Tell us about your PDF experience.

Build desktop apps for Windows

This documentation provides the latest guidance about building desktop apps for Windows 11 and Windows 10.

Get ready for development

GET STARTED

Start here!

An overview of Windows development options

Learning paths and modules

Code samples

Design and UI

OVERVIEW

Design and code your app UI

Windows 11 design principles

Make your apps great on Windows 11

Windows developer FAQ

Deploy and publish

OVERVIEW

Deployment overview

Deploy apps

The MSIX app package format

Publish Windows apps and games

Artificial intelligence and machine learning

CONCEPT

Windows AI

GitHub Copilot extension for Visual Studio

Windows Machine Learning

Windows Machine Learning samples

Community and support

OVERVIEW

Microsoft Q&A Forum

@WindowsDocs

OneDevMinute on YouTube

Windows developer support

Platforms

ARCHITECTURE

Windows App SDK

.NET MAUI

Universal Windows Platform (UWP)

Windows Presentation Foundation (WPF)

Windows Forms

Other development technologies

API Reference

REFERENCE

WinRT API for the Windows SDK

Win32 API for the Windows SDK

WinRT API for the Windows App SDK

Win32 API for the Windows App SDK

WinUI 3 API reference

What's New for developers in Windows 11

Article • 01/28/2025

Following the announcements at Microsoft Build 2024, here are some of the latest highlights for Windows developers:

- Updated recommendations for app development
- New AI and machine learning capabilities supported by Windows
- Enhanced developer tools

Development recommendations

You are strongly encouraged to consider using either the Windows App SDK/WinUI or WPF as your development platform for creating Windows client applications.

Windows App SDK

Expand table

Feature	Description
Windows App SDK	The Windows App SDK is a set of developer components and tools that represent the next evolution in the Windows app development platform. The Windows App SDK provides a unified set of APIs and tools that can be used in a consistent way by any desktop app on Windows 11 (and downlevel to Windows 10, version 1809).
Windows App SDK release notes	Details on the latest stable release of the Windows App SDK, which can be used by apps in production environments and by apps published to the Microsoft Store.
Create a new app with the Windows App SDK	The Windows App SDK includes WinUI 3 project templates that enable you to create apps with an entirely WinUI-based user interface. When you create a project using these templates (see Create your first WinUI 3 project), the entire user interface of your application is implemented using windows, controls, and other UI types provided by WinUI 3.
Use the Windows App SDK in an existing project	If you have an existing project in which you want to use the Windows App SDK, you can install the latest version of the Windows App SDK NuGet package in your project. Unpackaged apps must follow this procedure to use the Windows App SDK, but packaged apps can do this too.

Feature	Description
Download the Windows App SDK	There are several packages and release channels for the Windows App SDK. The Download the Windows App SDK page provides guidance on which ones you need, download links, and installation instructions.

WinUI

Expand table

Feature	Description
WinUI	WinUI is the native UI platform component that ships with the Windows App SDK (completely decoupled from Windows SDKs). The Windows App SDK provides a unified set of APIs and tools that can be used to create production desktop apps that target Windows 10 and later, and can be published to the Microsoft Store.

Windows AI

Expand table

Feature	Description
Windows AI	Enhance your Windows apps with AI through local APIs and Machine Learning models.
Windows Copilot Runtime Overview	Windows Copilot Runtime introduces new ways of interacting with the operating system that utilize AI, such as Phi Silica, the Small Language Model (SLM) created by Microsoft Research that is able to offer many of the same capabilities found in Large Language Models (LLMs), but more compact and efficient so that it can run locally on Windows.

Developer tools

Expand table

Feature	Description
Dev Home Overview	Dev Home was an experimental control center for Windows, but will no longer be supported beginning May 2025. Dev Home provided the ability to monitor projects in your dashboard using customizable widgets, set up your dev environment by downloading apps, packages, or repositories, connect to your

Feature	Description
	developer accounts and tools (such as GitHub), and create a Dev Drive for storage all in one place.
WSL (Windows Subsystem for Linux)	Windows Subsystem for Linux (WSL) is a feature of Windows that allows you to run a Linux environment on your Windows machine, without the need for a separate virtual machine or dual booting.

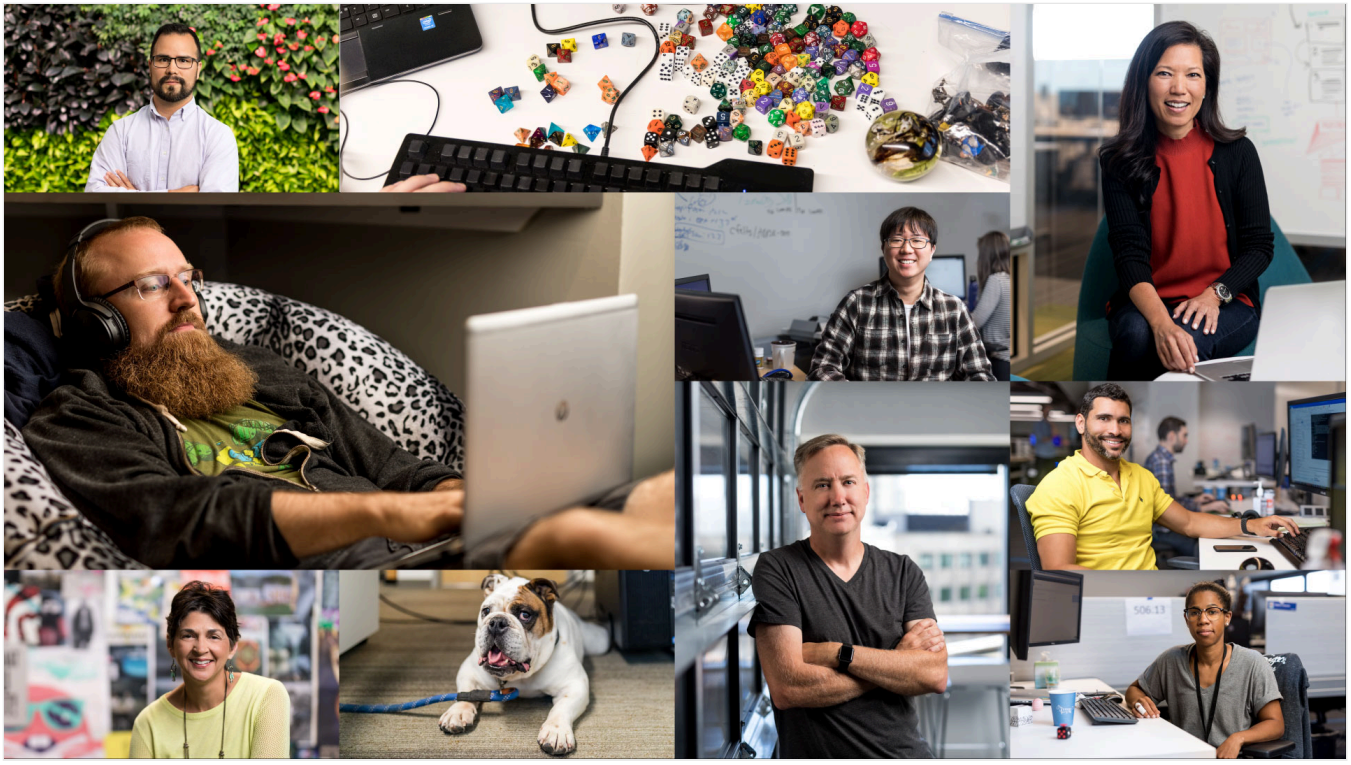
Samples

The WinUI 3 Gallery on GitHub is updated regularly to showcase the latest additions and improvements to WinUI in the Windows App SDK. The gallery app can also be downloaded from the Microsoft Store.

Windows developer community

Article • 04/24/2025

Stay updated with news, blogs, tutorials, and social channels relevant to Windows developers.



📣 Social media channels

X (formerly Twitter)

Expand table

Account	Link
Windows Developer	@windowsdev
Windows Developer Docs	@windowsdocs
Windows	@Windows
Microsoft Developer	@msdev
Microsoft Docs	@docsmsft
XAML Llama	@XAMLLlama
.NET	@dotnet

Bluesky

Expand table

Account	Link
Windows Developer Docs	microsoftdevdocs.bsky.social
Windows Insider Program	windowsinsider.bsky.social
Microsoft Developer	developer.microsoft.com
.NET	dot.net
DotNetBot	dotnetbot.bsky.social

LinkedIn

Expand table

Account	Link
Windows Developers Showcase	View
Microsoft Developers Showcase	View

Instagram

Expand table

Account	Link
@windowsdeveloper	View
@microsoftdeveloper	View
@dotnet.developers	View

Videos and tutorials

Expand table

Channel	Link
Windows Developer	YouTube
Microsoft Developer	Featured Content
.NET	YouTube

Blogs

- Windows Developer Blog

Keep in touch

- Windows App Development Discord

Overview of framework options

Article • 03/03/2025

This article contains the information you need to get started building apps for Windows.

Windows offers a wide range of languages, frameworks, and tools for building apps, including WinUI, React Native for Desktop, WPF, C++, C#, .NET, and a variety of cross-platform frameworks. Here, we provide information to help you decide which option is best for you.

WinUI

We recommend WinUI and the Windows App SDK to create apps that look great and take advantage of the latest Windows releases. If you're new to Windows development, or starting work on a new Windows app, WinUI provides the resources you need to create great apps for Windows 11.

WinUI is a XAML markup-based user interface layer that contains modern controls and styles for building Windows apps. As the native UI layer for the Windows App SDK, it embodies Fluent Design, giving each Windows app the polished feel that customers expect.

Get started with WinUI

> **Note**
>
> The Windows App SDK is a set of new developer components and tools that represent the latest evolution in the Windows app development platform. The Windows App SDK provides a unified set of APIs and tools that can be used in a consistent way by desktop apps on Windows 11 and downlevel to Windows 10, version 1809.
>
> While WinUI is the native UI layer, you can use the Windows App SDK with WPF, WinForms, or Win32 apps. If you've developed apps for Windows before, but are looking to get started with the Windows App SDK in an existing app, see Framework-specific guides.

React Native for Desktop

React Native is a development platform which allows building cross-platform apps. React Native for Desktop encompasses React Native for Windows and macOS, bringing React Native support to the Windows SDK. React Native for Desktop lets you use JavaScript to build native Windows apps for all devices supported by Windows 10 and Windows 11. This includes PCs, tablets, 2-in-1s, Xbox, Mixed Reality devices, etc.

With React Native for Desktop, you write most or all of your app code in JavaScript - or TypeScript - and the framework produces a native UWP XAML application. If your app needs to call a platform API, you can usually do so through one of the many community modules, or if a module does not yet exist, you can easily write a native module to expose it.

Here are some reasons to choose React Native for Desktop:

- You want to share code across platforms as much as possible, or you have web properties that you want to share code with.
- Improved developer productivity and inner loop, thanks to fast refresh.
- Your app's fundamentals (performance, accessibility, internationalization) are as good as a native UWP app.
- You have experience with and a preference for JavaScript or TypeScript
- You would like to leverage JavaScript-only libraries on npmjs.com, and many native libraries too.
- Your app will use the native controls, visual appearance, animations and colors, and therefore will feel integrated into the design language used in Windows. In addition, React Native for Desktop apps do not have to compromise on the set of APIs they can call, as the framework allows you to call platform APIs as well as write your own view managers and native modules.
- Large and growing community momentum, with lots of community modules.

Get started with React Native for Desktop

For more information about React Native for Desktop, see the following links:

- React Native for Windows repo on GitHub
- React Native for macOS repo on GitHub
- API reference
- React Native for Desktop resources

WPF

WPF is a well-established framework for Windows desktop applications with access to .NET or the .NET Framework. Like WinUI, it also uses XAML markup to separate UI from code. WPF provides a comprehensive set of application development features that include controls, data binding, layout, 2D and 3D graphics, animation, styles, templates, documents, media, text, and typography. WPF is part of .NET, so you can build applications that incorporate other elements of the .NET API.

Additionally, you can now integrate a sandbox environment into your packaged WPF applications, providing an additional layer of security. This enhancement requires little to no change to your code, thanks to the new Win32 App Isolation security feature.

> **Tip**
>
> If you've already invested in WPF, you can continue to use it and take advantage of the modernization options in .NET 9. You can build your apps knowing that Microsoft is continuing to invest in WPF. See the **Windows developer FAQ** for more information.

Get started with WPF

If you have a WPF .NET app, you also have access to modern Windows platform features and APIs provided by the **Windows App SDK**. For more information, see Use the Windows App SDK in a WPF app and Modernize your desktop apps.

> **Tip**
>
> If you need more help deciding which framework is the best choice for your app, see the **Choose the best application framework for a Windows development project** training module.

Other native platform options

Many apps for Windows are written using Win32, Windows Forms, or UWP. Each of these frameworks is supported and will continue to receive bug, reliability, and security fixes, but varying levels of investment for new features and styles. For more information about these app types see the following tabs.

Win32

Win32 desktop apps (also sometimes called *classic desktop apps*) are the original app type for native Windows applications that require direct access to Windows and hardware. This makes Win32 the app type of choice for applications that need the highest level of performance and direct access to system hardware.

Using the Win32 API with C++ makes it possible to achieve the highest levels of performance and efficiency by taking more control of the target platform with un-managed code than is possible on a managed runtime environment like WinRT and .NET. However, exercising such a level of control over your application's execution requires greater care and attention to get right, and trades development productivity for runtime performance.

Here are a few highlights of what the Win32 API and C++ offers to enable you to build high-performance applications.

- Hardware-level optimizations, including tight control over resource allocation, object lifetimes, data layout, alignment, byte packing, and more.
- Access to performance-oriented instruction sets like SSE and AVX through intrinsic functions.
- Efficient, type-safe generic programming by using templates.
- Efficient and safe containers and algorithms.
- DirectX, in particular Direct3D and DirectCompute.
- Use C++/WinRT to create modern desktop Win32 apps with first-class access to Windows Runtime (WinRT) APIs.

Additionally, you can now integrate a sandbox environment into your Win32 applications, providing an additional layer of security. This enhancement requires little to no change to your code, thanks to the new Win32 App Isolation security feature.

Get started with Win32

You also have access to modern Windows platform features and APIs provided by the **Windows App SDK**. For more information, see Use the Windows App SDK in an existing project and Modernize your desktop apps.

Other cross-platform options

If you need your app to be cross-platform, in addition to React Native for Desktop, you should consider .NET MAUI or a Progressive Web App (PWA). There are many other choices available (here's a list of popular options), but these are some good starting points.

.NET MAUI harnesses the power of WinUI on Windows, while also enabling execution on other operating systems. Another cross-platform option, Progressive Web Apps (PWAs), are websites that function like installed, native apps on Windows and other supported platforms, while functioning like regular websites on browsers.

For more information, see the following tabs.

.NET MAUI

.NET Multi-platform App UI (MAUI) is an open-source, cross-platform framework for building Android, iOS, macOS, and Windows applications that leverage the native UI and services of each platform from a single .NET code base. Because .NET MAUI favors platform native experiences, it uses WinUI and the Windows App SDK so apps get the latest user experience on Windows. This gives your apps access to everything you get with WinUI plus the ability to reach to other platforms.

.NET MAUI for Windows is a great choice if:

- You want to share as much .NET code as possible across mobile and desktop applications.
- You want to ship your application beyond Windows to other desktop and mobile targets with native platform experiences.
- You want to use C# and/or XAML for building cross-platform apps.
- You're using Blazor for web development and wish to include all or part of that in a mobile or desktop application.

Get started with .NET MAUI

For more information about .NET MAUI, see the following links:

- .NET MAUI documentation
- .NET MAUI on GitHub
- .NET MAUI Product Roadmap
- Build Windows apps with .NET MAUI
- Resources for learning .NET MAUI
- Video Series - .NET MAUI for Beginners
- Build 2022: Build native apps for any device with .NET and Visual Studio

App development framework feature comparison

There is a wide range of options for developing applications for Windows. The best option for you depends on your application requirements, your existing code, and your familiarity with the technology. The following table lists the most popular app development frameworks available on Windows and the features supported by each framework.

Expand table

Feature	.NET MAUI	Blazor Hybrid	React Native for Desktop	UWP XAML (Windows.UI.Xaml)	Win32 (MFC or ATL)	Windows Forms	WinUI 3	WPF
Language	C#	C#	JavaScript, TypeScript	C#, C++, Visual Basic	C++, Rust	C#, Visual Basic	C#, C++	C#, Visual Basic
UI language	XAML/Code	Razor	JSX	XAML	Code	Code	XAML	XAML

Feature	.NET MAUI	Blazor Hybrid	React Native for Desktop	UWP XAML (Windows.UI.Xaml)	Win32 (MFC or ATL)	Windows Forms	WinUI 3	WPF
UI designer (drag & drop)	Not supported	Not supported	Not supported	**Supported**	Not supported	**Supported**	Not supported	**Supported**
UI debugging	Hot Reload	Hot Reload	Fast Refresh	Hot Reload	-	Hot Reload	Hot Reload	Hot Reload
Fluent Design	**Supported**	**Supported**	**Supported**	**Supported** (via WinUI 2)	Not supported	Not supported	**Supported**	Not supported
.NET	.NET	.NET	N/A	.NET Core & .NET Native	N/A	.NET & .NET Framework	.NET	.NET & .NET Framework
Windows App SDK	**Supported** (more info)	**Supported** via MAUI	**Supported** (more info)	Not supported	**Supported**	**Supported** (more info)	**Supported**	**Supported** (more info)
Great for touch	**Supported**	**Supported**	**Supported**	**Supported**	Not supported	Not supported	**Supported**	Not supported
Cross-platform	**Supported**	**Supported**	**Supported**	Not supported	Not supported	Not supported	Not supported	Not supported
Xbox/HoloLens apps	Not supported	Not supported	**Supported**	**Supported**	Not supported	Not supported	Not supported	Not supported
Sandboxing (AppContainer)	Not supported	Not supported	**Supported**	**Supported**	Not supported	Not supported	Not supported	Not supported
Currently supported	**Supported**	**Supported**	**Supported**	**Supported**	**Supported**	**Supported**	**Supported**	**Supported**
Receiving updates	**Supported**	**Supported**	**Supported**	**Supported** (security & bugfix)	**Supported**	**Supported**	**Supported**	**Supported**
Roadmap	GitHub	GitHub	GitHub	n/a	n/a	GitHub	GitHub	GitHub

Learn more about each of these options:

- Windows developer FAQ
- .NET Multi-platform App UI (.NET MAUI)
- ASP.NET Core Blazor Hybrid
- React Native for Desktop
- Universal Windows Platform (UWP)
- Recommendations for Choosing Between ATL and MFC
- Windows Forms
- Windows Presentation Foundation (WPF)
- WinUI in the Windows App SDK (WinUI 3)

Next steps

- Use WinUI to start developing apps for Windows

> WinUI is our recommended platform for Windows apps, and these steps will quickly get you started.

- Set up your development environment on Windows

  > Windows isn't just great for developing apps that run on Windows, it's also a powerful environment for developing apps for any platform. Learn more about the tools and options available to maximize your development.

Start developing Windows apps

Article • 03/28/2025

Welcome to Windows app development. This guide will take you through the steps needed to begin creating apps using the latest Windows development frameworks: the Windows App SDK and WinUI. It will also point you to resources that will help you learn more about Windows development. If you are already comfortable developing apps for Windows, but want to know more about the latest tools, please see Develop Windows desktop apps.

> **Tip**
>
> Microsoft Copilot is a great resource if you have questions about getting started writing Windows apps.

1. Enable Developer Mode

Windows has a special mode for developers that adjusts security settings in order to let you run the apps you're working on. You'll need to enable Developer Mode before you can build, deploy, and test your app using Visual Studio.

> **Tip**
>
> If you don't enable it now, you'll be prompted to enable it when you try to build your app in Visual Studio.

To enable Developer Mode:

- Open Windows Settings and navigate to the **System > For developers** page.
- Toggle the Developer Mode switch to **On** and confirm your choice in the confirmation dialog.

For more information about Developer Mode, see Enable your device for development.

2. Install Visual Studio

You'll use Visual Studio, Microsoft's comprehensive integrated development environment (IDE), to create your WinUI app. It's the preferred development tool of many Windows developers and it will help you write, debug, and deploy your apps. The

project templates in Visual Studio will quickly get you started with projects for Windows and many other platforms.

> **Tip**
>
> Before installing these tools, make sure your development computer meets the system requirements for Windows app development.

Use the link below to download and install the latest Visual Studio. The installer will walk you through the steps, but if you find you need detailed instructions, see Install Visual Studio.

Download Visual Studio

The free Visual Studio Community Edition includes everything you need to create your apps. If you're working with a development team or enterprise, you might need Visual Studio Professional or Visual Studio Enterprise. See What is Visual Studio? for more info.

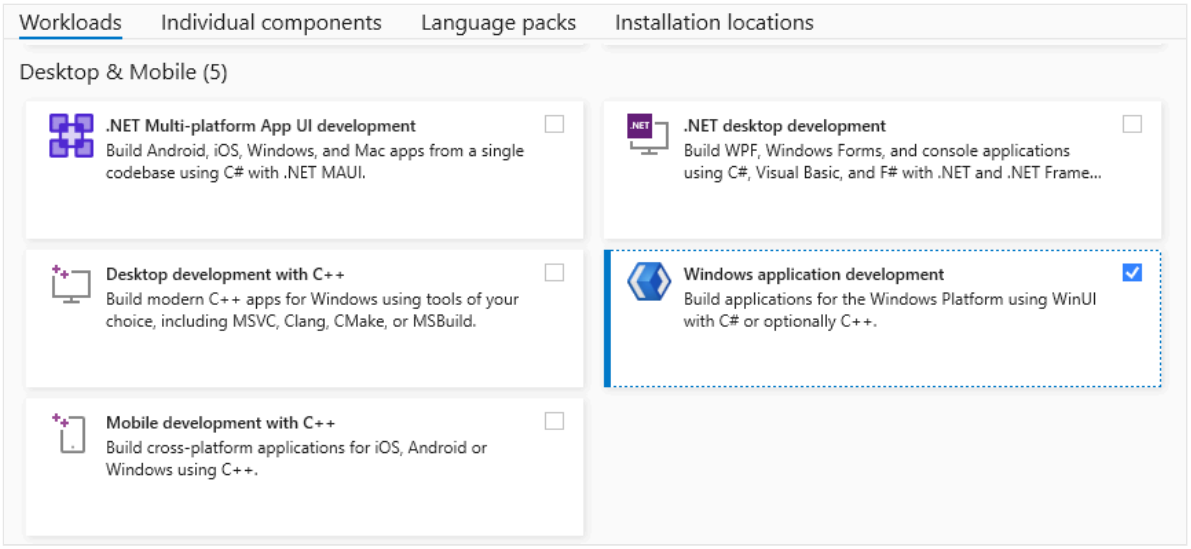
2.2 Required workloads and components

While installing Visual Studio, you need to install the workloads and components required for developing with WinUI and the Windows App SDK. After installation, you can open the Visual Studio Installer app and select **Modify** to add workloads and components.

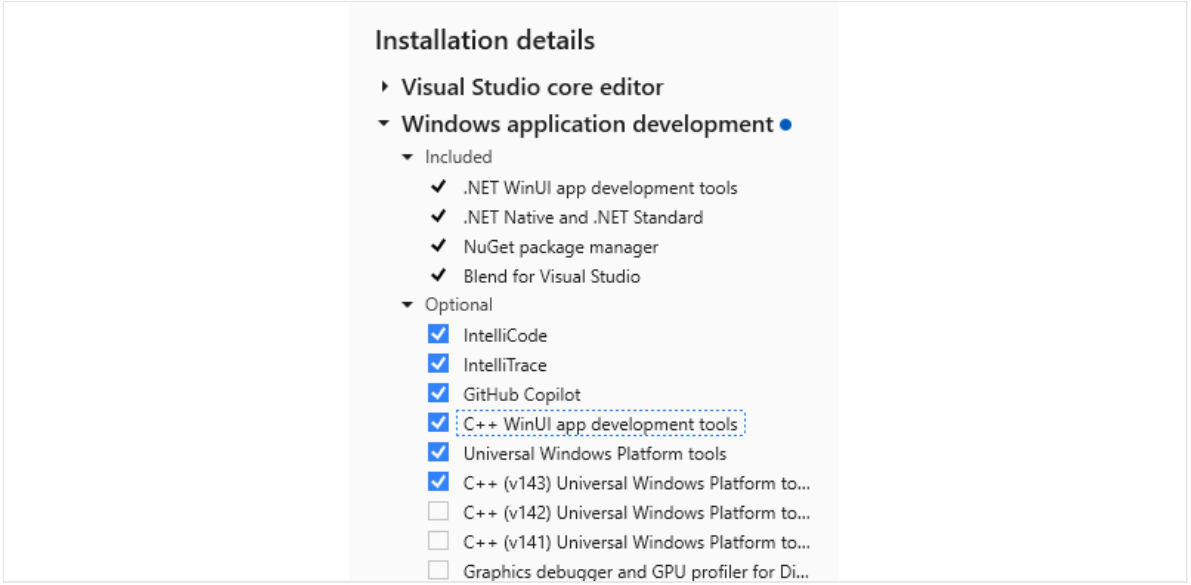
On the **Workloads** tab of the Visual Studio Installer app, select the following workloads and components:

Visual Studio 2022 version 17.10 and later

- For **C#** app development using the Windows App SDK, select **Windows application development**.



- For **C++** app development using the Windows App SDK, select **Windows application development**.
  - Then in the **Installation details** pane, under the **Windows application development** node, select **C++ WinUI app development tools**. (This will also select any additional required components.)

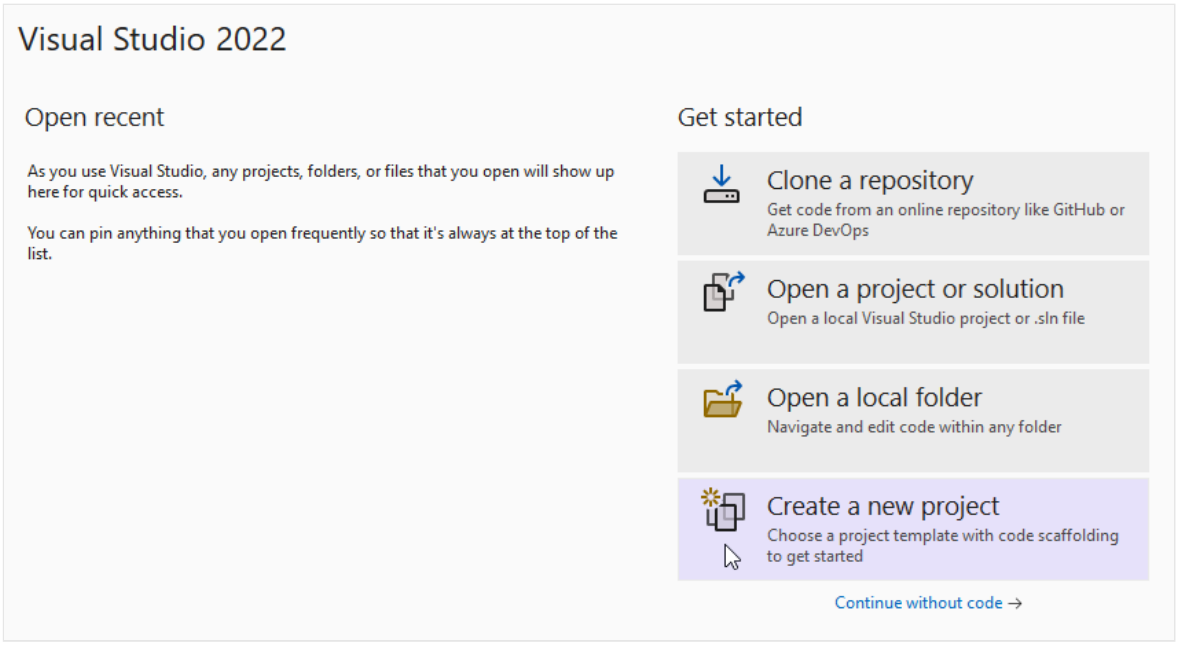


3. Create and launch your first WinUI app

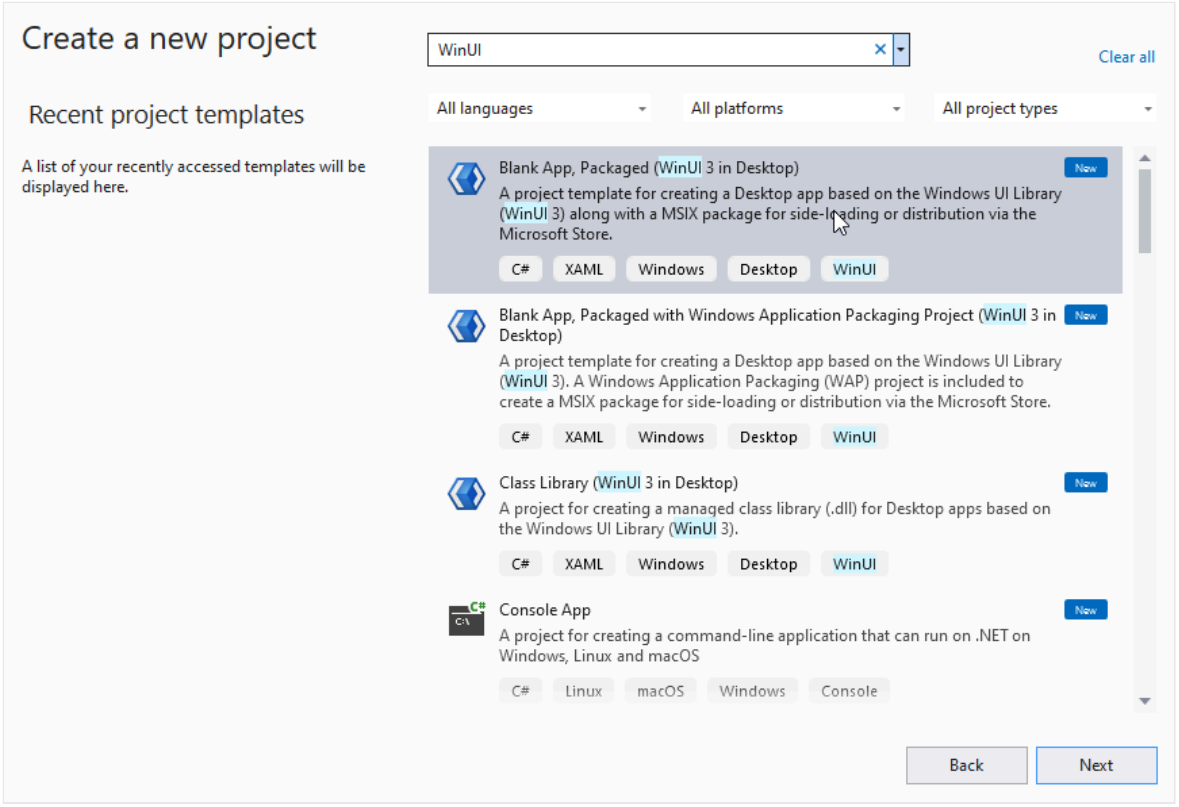
Visual Studio project templates include all the files you need to quickly create your app. In fact, after you create your project from a WinUI app template, you'll already have an app that you can run, and then add your code to.

To create a new project using the WinUI C# Blank App project template:

1. Open Visual Studio and select **Create a new project** from the launch page. (If Visual Studio is already open to the editor, select **File** > **New** > **Project**):

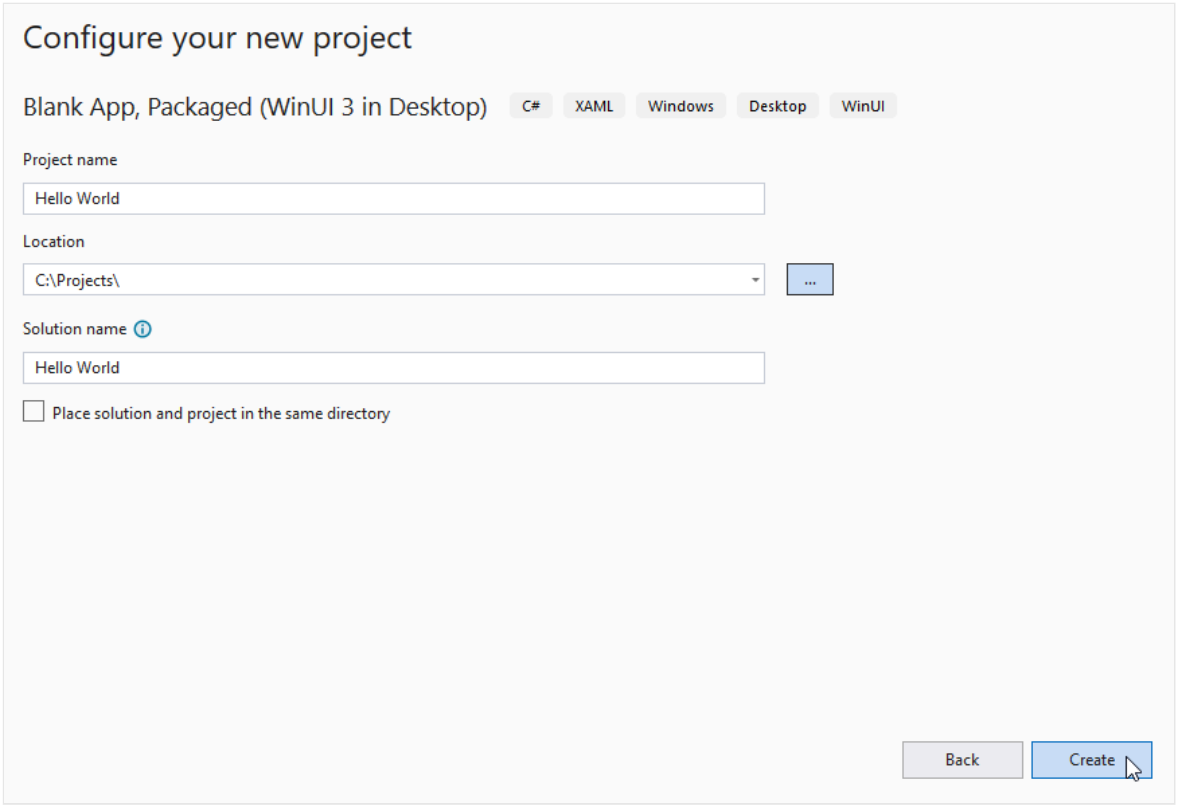


2. Search for `WinUI` and select the `Blank App, Packaged (WinUI 3 in Desktop)` C# project template, then click **Next**:



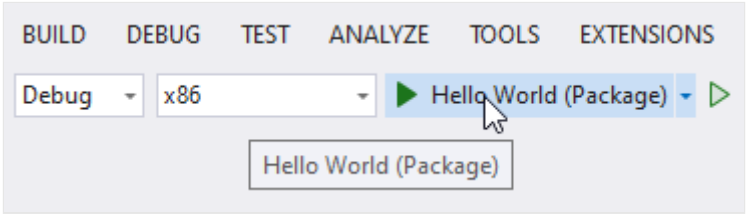
3. Specify a project name, then click **Create**. You can optionally specify a solution name and directory, or leave the defaults. In this image, the `Hello World` project

belongs to a `Hello World` solution, which will live in `C:\Projects\`:

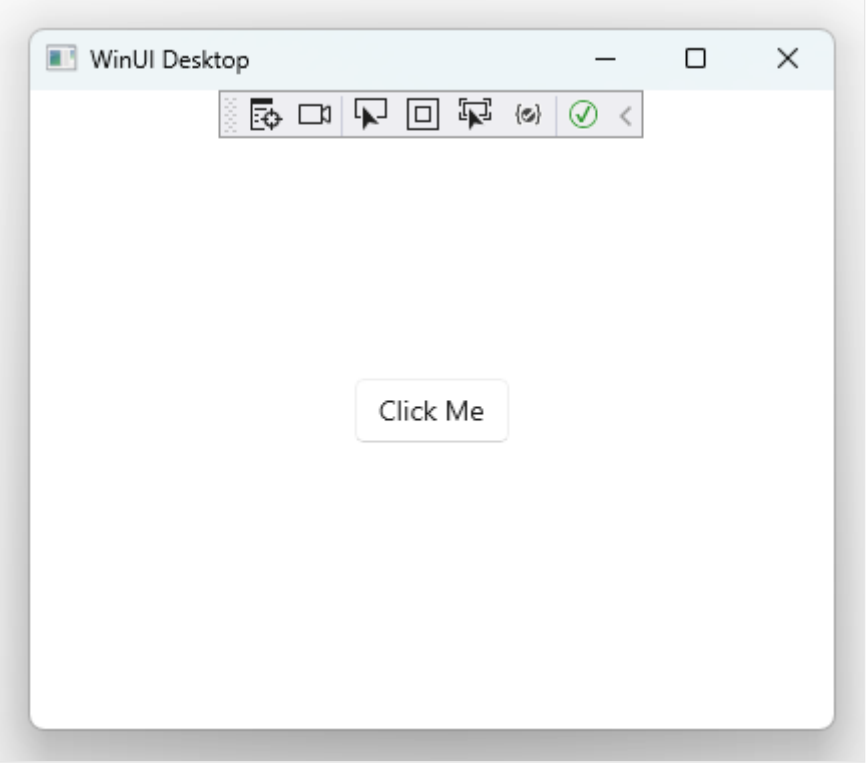


> **Note**
>
> If you want to use this project to build the complete app in the Next steps section, name the project `WinUINotes`.

4. Click the Debug "Start" button to build and run your project:



Your project will build, be deployed to your local machine, and run in debug mode:



5. To stop debugging, close the app window, or click the debug "Stop" button in Visual Studio.

4. Delete sample code

The `MainWindow` class included with the project template includes some sample code that needs to be removed to make room for your content.

1. Double-click `MainWindow.xaml` in **Solution Explorer** to open it. You should see XAML markup for a `StackPanel` control.

2. Delete the XAML for the `StackPanel`. (You'll add your own content in it's place as you create your app.)

   XAML

   ```
   <!-- ↓ Delete this. ↓ -->
   <StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
   VerticalAlignment="Center">
       <Button x:Name="myButton" Click="myButton_Click">Click Me</Button>
   </StackPanel>
   ```

   If you try to run your app now, Visual Studio will throw an error along the lines of `The name 'myButton' does not exist in the current context`. This is because you

deleted the `Button` control named `myButton`, but it's still referenced in the `MainPage.xaml.cs` code-behind file. Delete the reference in the code file, too.

3. Double-click `MainWindow.xaml.cs` in **Solution Explorer** to open it.

4. Delete the `myButton_Click` event handler.

```
C#
```

```
public sealed partial class MainWindow : Window
{
    public MainWindow()
    {
        this.InitializeComponent();
    }

    // ↓ Delete this. ↓
    private void myButton_Click(object sender, RoutedEventArgs e)
    {
        myButton.Content = "Clicked";
    }
    // End delete.
}
```

5. Save the file by pressing `CTRL + SHIFT + S`, clicking the Save All icon in the tool bar, or by selecting the menu **File** > **Save All**.

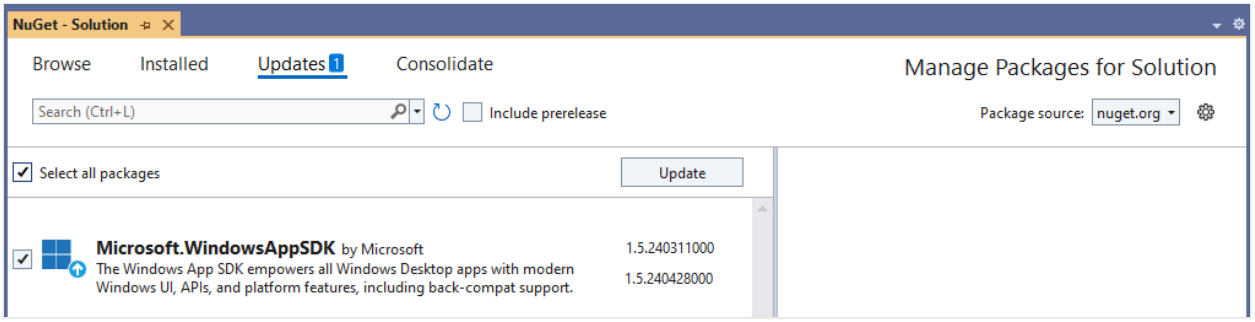
5. Update to the latest WinUI/Windows App SDK

The Windows App SDK (and WinUI, which is part of it) is distributed as a NuGet package. This means updates can be released out-of-sync with Windows and Visual Studio. As a result, the Visual Studio template you used to create your project might not reference the latest Windows App SDK NuGet package. To ensure you have the latest features and fixes, you should update your NuGet packages every time you create a new project in Visual Studio.

To update the Windows App SDK NuGet package for your project:

- In Visual Studio, with your project loaded, select **Tools** > **NuGet Package Manager** > **Manage NuGet Packages for Solution...**.
- If an update is available, it will appear on the Updates page. Check the box next to the listed update. (To include prerelease updates, check the "Include prerelease" option. To learn more about what's included in an update, see the release notes.)

- Click the **Update** button, then click **Apply** in the Preview changes dialog, then accept the license terms to finish installing the update.



Now your project is using the latest WinUI features that are available, and it's ready for you to make it your own.

Next steps

Complete a tutorial to create a note app with WinUI

- To get an idea of what WinUI has to offer, check out the WinUI Gallery app.

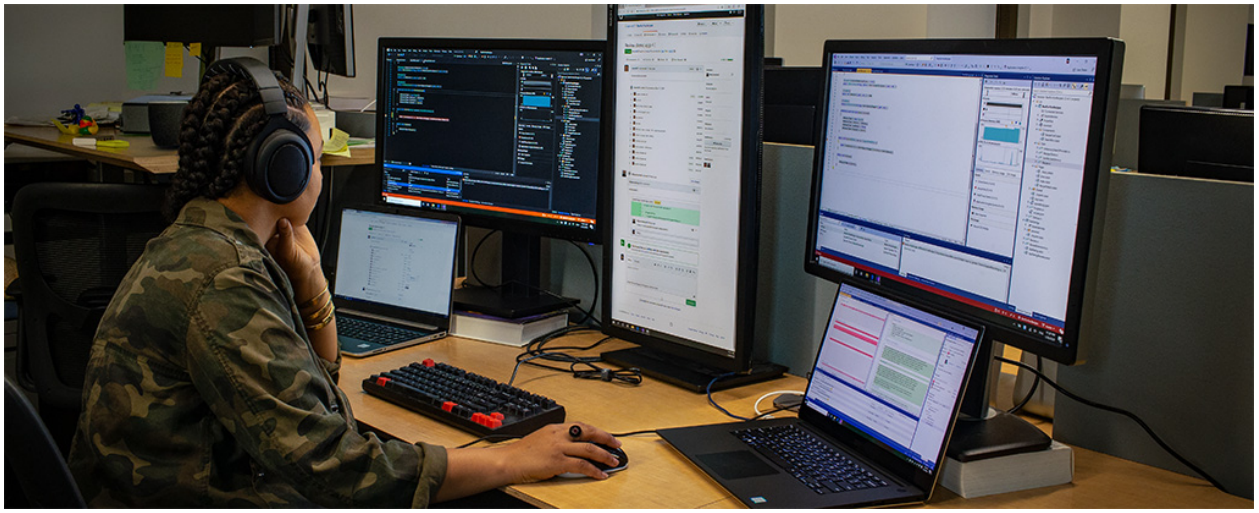
  > The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

- Learn more about WinUI fundamentals.
- Explore Fluent Design principles.
- Find samples and tools to help you develop apps more efficiently.

Samples and resources

Article • 08/19/2024

This page contains links to resources that can make you more efficient as you develop your apps for Windows. It includes sample apps and toolkits; code samples that demonstrate specific tasks, features, and API usage patterns; and tools to enhance your development environment.



Sample gallery apps

WinUI 3 Gallery

The WinUI 3 Gallery app showcases all the XAML UI components in an interactive format. This app is the interactive companion to the Fluent Design Guidelines and has examples of WinUI 3 controls, features, and styles.

You can get the app from the Microsoft Store or get the source code on GitHub at microsoft/WinUI-Gallery.

Input & Composition Gallery

Use this sample app to browse through a collection of samples showing creative ways to customize and differentiate your app, then check out the accompanying code on GitHub to see how it was made. It showcases a variety of Microsoft.UI.Composition and Microsoft.UI.Input APIs, the building blocks that make up the Fluent Design System.

You can get the app from the Microsoft Store or get the source code on GitHub at microsoft/WindowsCompositionSamples.

Toolkits and helpers

Windows Community Toolkit

The Windows Community Toolkit is a collection of helper functions, custom controls, and app services. It simplifies and demonstrates common developer tasks when building apps for Windows. See Get started with the Windows Community Toolkit for detailed information about using the toolkit.

You can get the *Windows Community Toolkit Gallery* app from the Microsoft Store to see the controls in an actual app or get the source code on GitHub at CommunityToolkit/Windows.

Template Studio

Template Studio for WinUI accelerates the creation of new WinUI apps using a wizard-based UI. It's available to create apps using C# and the MVVM Toolkit.

You can get the extension from the Visual Studio Marketplace or get the source code on GitHub at microsoft/TemplateStudio.

> **Important**
>
> Template Studio for WinUI only creates apps that use the MVVM design pattern. It does not support creating apps that use the standard code-behind pattern.

Windows App SDK / WinUI 3 samples

The samples in this list demonstrate features from the Windows App SDK and WinUI 3. See our Samples Browser for a more extensive catalog of samples.

Most of the samples identified in this list contain Solution (`.sln`) files that can be opened in Visual Studio. Refer to each resource's `Readme` for additional instructions.

You can find these sample in the main Windows App SDK samples repository on GitHub at microsoft/WindowsAppSDK-Samples

Samples that demonstrate specific tasks

Expand table

Sample	Task
Activation	**Handle app activation kinds**. App activiation APIs control the way that

Sample	Task
	your app handles activation kinds like `Launch`, `File`, and `Protocol`.
App Instancing	**Determine application instancing behavior**. App instancing APIs control whether or not users can run multiple instances of your application at the same time.
Power Notifications	**Use state notifications**. State notifications allow you to detect when the user's device enters specific states, such as low power mode.
Restart	**Programatically restart your app**. This sample makes use of the Windows App SDK Restart APIs.
Resource Management	**Tailor your app to the user and device settings**. MRT Core resource management APIs allow you to adapt resource utilization to specific situations.
Deployment Manager	**Call the Deployment API** to ensure that Windows application framework components are up to date.
Installer	**Launch the Windows App SDK installer** without using a console window.
TextRendering with DWriteCore	**Render text** using DWriteCore APIs.
Unpackaged	**Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged**, an alternative to packaged app deployment architecture.
Dynamic Dependencies	Demonstrates the techniques shown in Use the dynamic dependency API to reference MSIX packages at run time.
Push Notifications	**Add push notifications to your app**. Push Notifications can be used to send device-native app notifications to users.
App Notifications	**Add toast notifications to your app**. See App Notifications to learn more.
Custom Controls	**Add custom C#/WinRT controls to your app** using the patterns demonstrated in Author Windows Runtime components with C#/WinRT.
Windowing	**Add windowing support to your app** to create/hide new windows, customize titlebars, and more. See Manage App Windows to learn more.

Development environment

If you've completed the steps in Get started with WinUI, then you've already installed Visual Studio and enabled Developer Mode. These links provide more information about

these tools to take you beyond the basics.

Developer Mode

Windows has a special mode for developers that adjusts security settings in order let you to run the apps you're working on. For more information, see Enable your device for development and Developer Mode features and debugging.

Visual Studio

The preferred development tool of many Windows developers, Visual Studio will let you create projects for Windows, and many other platforms. It is a powerful IDE that can help you write, debug, and deploy your apps. For more information about the many things you can do with Visual Studio, see the Visual Studio documentation.

Visual Studio Code

A highly extensible editor, Visual Studio Code can be customized to support almost any kind of development you can think of. It is a great choice for writing apps for Windows, and other platforms. For more information, see the Visual Studio Code documentation.

Additional tools

These additional tools will help you set up your development environment on Windows.

Dev Drive

In order to speed up common development tasks, you can create a specially formatted drive (a Dev Drive) that is used to store your projects.

Windows Terminal

Windows Terminal is a modern host application for the command-line shells you already love, like Command Prompt, PowerShell, and bash (via WSL).

Windows Subsystem for Linux (WSL)

WSL lets you run Linux distributions on Windows. This is a great way to use open source tools to develop apps for Windows.

Language documentation

Most WinUI apps are written in C# or C++. These guides contain articles, tutorials, and code samples to help you get started with these languages.

- C# language documentation
- C++ language documentation
- C++/WinRT language documentation

Next steps

Now is also a good time to create a developer account. Among other things, this will allow you to publish your apps to the Microsoft Store. Here is how you do that: Create a developer account

Also see:

- Windows Application Development - Best Practices
- Windows Developer FAQ

Windows Application Development - Best Practices

Article • 03/15/2023

The best practices in this document will help you build great Windows apps that reach and delight ~1.5 billion diverse PC users around the world. This document is divided into the following sections:

1. **User experience**: Guidance in this section will help you improve the look, feel, and usability of your apps.
2. **Performance and fundamentals**: Guidance in this section will help you improve your app's performance and resource utilization.
3. **Operating system / hardware optimization**: Guidance in this section will help you optimize your packaging and distribution for a variety of hardware configurations.
4. **Application discovery and management**: Guidance in this section will make it easier for users to discover, install, update, and uninstall your app.
5. **Accessibility**: Guidance in this section will help you build accessible and inclusive experiences.
6. **Security and privacy**: Guidance in this section will help you mitigate security risks and meet your users' privacy needs.

User experience (UX)

Windows 11 marks a visual evolution of the Windows operating system that improves the look, feel, and usability of Windows. Our studies show that users have high expectations for Windows apps:

- They expect Windows apps to work with a complete range of inputs.
- They expect design and interaction patterns that look and feel native on current and future devices.
- They expect support for modern windowing workflows and shell integration points.

When applications adhere to Windows styles and standard Windows behaviors, users don't have to re-learn interaction patterns. This makes it much easier for users to use your app. An app that looks great can create a great first impression, but an app that's also easy to use and helps the user accomplish their goals will create a great lasting impression.

Windows 11 is built on the Windows 11 design principles. Following these guidelines as you build your apps will help you meet your customers' expectations of a great app

experience. When thinking about incorporating the latest and recommended Windows application UI/UX patterns into your Windows applications, we recommend that you focus on these 5 areas:

- Layout
- UI interaction
- Visual style
- Window behavior
- Shell integration points

WinUI 3 provides built-in support for many of these experiences and styles through its **common controls**. If you aren't able to use WinUI 3, consider emulating the styles demonstrated in our design toolkits and WinUI Gallery.

Layout

Windows applications run on a variety of configurations that match users' needs. Test your application's panes and/or pages across a variety of dimensions, devices, window sizes, DPI settings, and scale settings. Your application should work as expected even when resized down to small dimensions.

DPI awareness

WinUI applications automatically scale for each display that they're running on. Other Windows programming technologies (Win32, WinForms, WPF, etc.) don't automatically handle per-monitor DPI scaling. Without additional work to support per-monitor DPI scaling for these technologies, applications may appear blurry or incorrectly-sized. See High DPI Desktop Application Development on Windows for more information.

Responsive layout

Use responsive design techniques to optimize your app pages for different window sizes. Follow the guidelines for panning or scrolling to ensure that users can always access your content, no matter how small the app window gets.

UI interaction

Windows users can choose from a wide variety of input devices to interact with your application, and Windows has specific system experiences that people are accustomed to using. When your application adheres to these experiences, your users can use your

application reliably. When your app doesn't follow these conventions, users may find it confusing or frustrating.

On-object commanding

Use on-object commanding such as context menus, swipe commands, and keyboard shortcuts. Windows 11 improves the behavior of the right-click context menu, so if your app creates context menus, refer to the latest context menu integration guidance. WinUI text controls automatically expose cut/copy/paste commands, but other controls may need extra work to support these commands.

Text interaction

Whenever there is text in an application, users expect that they can select and copy it. If the text is editable, they expect that they can cut and paste, as well. By providing consistent shortcuts to users, you let them complete their tasks more efficiently. Ensure that these actions can be performed using keyboard, mouse or trackpad, touch, and pen.

Panning & Scrolling

It is uncommon for an application's UI to fit entirely inside a single page that does not need to scroll. Even if there are only a few UI elements, users can freely resize the app Window and cause some UI elements to be hidden. Ensure that your application's UI properly supports scrolling and panning (using keyboard, mouse or trackpad, touch, and pen) to let users access any UI elements that might have moved out of the visible window area.

Visual style

Windows 11 is built on the Windows 11 design principles: Effortless, Calm, Personal, Familiar, and Complete + Coherent. We believe experiences that follow these principles bring great user experiences on Windows.

Materials: Acrylic and Mica

Acrylic and Mica are visual materials that give interactive UI controls a distinct "occluded" visual style. Use Acrylic to apply a semi-transparent style to transient surfaces like context menus, flyouts, and other elements that can be light-dismissed. Use Mica to add a subtle adaptive tint to long-lived UI surfaces like the title bar.

More information about Acrylic and Mica materials can be found in Things you can do to make your app great on Windows 11.

Dark and Light themes

Dark and Light themes give users a way to adapt your app to their visual preferences. Windows 11 updates the color tones to be softer on the eyes by avoiding pure white and black, which makes the colors much more delightful. WinUI supports switching between Dark and Light themes by default (see XAML theme resources). For Win32 apps, see Support Dark and Light themes in Win32 apps. (The title bar in Win32 apps does not automatically adapt to the Dark theme. Be sure to follow the title bar guidance in the article).

Refreshed UI elements

Windows 11 geometry has been crafted to support modern app experiences. Progressively rounded corners, nested elements, and consistent gutters combine to create a soft, calm, and approachable effect that emphasizes unity of purpose and ease of use.

The visual and behavioral changes are built in to WinUI 3. Use WinUI 3 where you can to take advantage of the work that has already been done. If you aren't able to use WinUI 3, consider emulating the styles demonstrated in our design toolkits and WinUI Gallery.

Context menu

A context menu is a shortcut menu that the user invokes with a right-click or tap & hold action to reveal a menu of commands relevant to the context of the control the user is interacting with. Users expect the appearance and behavior of context menus to be coherent across Windows. Use platform-provided context menus whenever possible to keep them consistent with the rest of the system.

Iconography and typography

Windows 11 has updated icons ("Segoe Fluent Icons"), improved support for animated icons, and a new UI font ("Segoe UI Variable"). We recommend that you use these new icons and font whenever possible to be coherent on Windows 11. The new font brings much softer geometry and makes the text much more legible.

More information about iconography and typography on Windows can be found in Things you can do to make your app great on Windows 11.

Window behavior and style

Applications run in a frame provided by Windows, and users expect the built-in Windows look and behaviors to be consistent across app windows. Consider supporting the features listed here to ensure that your app looks and functions as users expect on Windows 11.

Snap Layout

Window snapping is greatly enhanced in Windows 11, and the Snap Layout menu is a new feature to help users discover and use the power of window snapping. Use the Snap Layout menu to test your app in different Snap Layouts and ensure your app supports different snap sizes, like 1/2, 1/3, and 1/4 screen.

If the Snap Layout menu doesn't appear for your app by default, see Support snap layouts for desktop apps on Windows 11 for some steps you can take to enable it.

Title bar and caption buttons

The title bar and caption buttons (minimize, maximize, close) are how users interact with Windows to resize, move, and close app windows. Having a consistent experience will help people use your application smoothly. See Windows app title bar to learn about title bar and caption button design for Windows.

You can use the Windows App SDK APIs to integrate app content with the title bar in WinUI 3, .NET, WinForms, and WPF apps.

Rounded corners

In most cases, your app's window will have rounded corners by default on Windows 11. If you've customized your app window and don't have rounded corners, see Apply rounded corners in desktop apps for Windows 11 for some things you can do. You should also avoid customizing window borders and shadows, which can prevent the system from rounding the window corners.

Shell integration points

Windows shell integration lets users benefit from your app even when its not running in the foreground or even visible on-screen. When your app integrates well with Windows, it becomes part of the users workflow with other apps and helps create a seamless experience.

Toast notifications

Toast notifications are the Windows notifications that appear at the bottom of the user's screen and in the Notification Center.

- Notifications should be personalized, actionable, and useful to your users. Try to give your users what they want, not what you want them to know.
- Notifications shouldn't be noisy. Too many interruptions from your app leads to users turning off this critical communication channel for your app.
- Respond to the user's intent. Selecting a notification should launch your app in the notification's context. The only exception to this guideline is when the user selects a button on your notification that's attached to a background task, such as a quick reply.
- Provide a consistent Notification Center experience. Keep Notification Center tidy by clearing out old notifications.

For more information about toast notifications, see Notifications design basics.

Performance & fundamentals

Windows users expect Windows apps to exhibit great performance and fundamentals. As you design and build your app, it is important to keep in mind optimizing for memory usage, power consumption, responsiveness, reliability, and the impact on long-term sustainability. Allocating time to test and measure your application's fundamentals and performance will ensure that your users have a first-class experience.

Following the best practices in this section will help you meet your customers' expectations across these criteria.

- Minimize application memory usage:
  - Reduce foreground memory usage.
  - Minimize background work.
  - Release resources while in the background.
  - Ensure your application does not leak memory.
- Make efficient use of the disk footprint:
  - Enable "pay for play" for optional functionality.

    - Ensure any caches are sized efficiently.
    - Implement new experiences in a disk-efficient manner.
    - Optimize individual binary sizes where possible.

- Improve power consumption and battery life by minimizing background work:
    - Do not wake the CPU or use system resources while in the background.

- Improve the responsiveness of your app's launch and key interactions
    - Define your key interaction scenarios and add ETW events to measure.
    - Set goals based on the interaction class associated with user expectations.

To learn more, see the Performance and fundamentals overview, which will cover questions such as "What is application performance and why is it important?" or "What tools can I use to measure Windows application performance?", as well as linking to case studies, related blogs, support communities, and information on how performance engineering intersects with sustainability--reducing the impact your application will have on our planet.

Operating system / hardware optimization

Windows apps can be built, packaged, and delivered in a variety of ways. The best practices in this section will help you optimize these aspects of your application across hardware configurations.

MSIX app attach and Azure Virtual Desktop

If you want your app to run best in an enterprise environment, add support for MSIX app attach.

MSIX app attach lets you deliver MSIX applications to both physical and virtual machines. It's made specifically for Azure Virtual Desktop (AVD), a desktop and app virtualization service that runs on the cloud. Using MSIX app attach with AVD can help you improve sign-in times for users, and it can reduce infrastructure costs for your enterprise.

Windows on Arm

Windows can run on Arm devices. Arm PCs benefit from extended battery life and integrated support for mobile data networks. These PCs also provide great application compatibility and allow you to run your existing `x86` and `x64` applications unmodified.

For best performance, you should enable your apps to take full advantage of the energy-efficient Arm processor architecture by either building a full Arm version or by optimizing the parts of the codebase that would benefit most from native performance. For more information on these techniques, refer to Windows on Arm and Arm64EC for Windows 11 apps on Arm.

Push notifications

Push notifications allow you send information from your cloud service to your app in a performance-optimized way. Push notifications include raw notifications, badge notifications, and toast notifications sent from your cloud service.

- Use push notifications to wake up the app/client rather than always keeping it running to optimize performance on the user's device.
- Notification channels are not meant to be used to send advertisements.
- Respect `retry-after` headers – this protects our service and ensures notification delivery success.
- Remove expired/revoked channels from the system. Windows Notification Service (WNS) does not process requests for expired/revoked channels.
- Avoid sudden, large bursts of requests to WNS. This can lead to throttled responses.
- Utilize the `MS-CV` header. This will help with end-to-end traceability and diagnostics.
- Have a back-up mechanism for when notifications don't work.
- Use Azure Notification Hubs (ANH). ANH gives you access to engagement features like targeting audiences, scheduling notifications, and broadcasting notifications. If you're a Windows-only developer today, using ANH will make it easy for you to transition your notifications infrastructure to other platforms in the future.

Application discovery and management

Reliable installation, update, and uninstallation experiences are important pieces of a consistent, high-quality user experience. The following best practices will help ensure that your application leaves a good impression when discovered and managed by users:

Application discovery

- Listing your app on Microsoft Store can make your app more discoverable for users.

- If you're hosting your app across multiple channels (for example - on a website and on the Microsoft Store), your application should have a consistent application identity and update mechanism across all channels.
- Distribute your app through the Microsoft Store to make it more discoverable for users. Note that Store apps are made available to users through the Windows Package Manager WinGet. If you don't publish to the Microsoft Store, you can still make your app easily discoverable in WinGet via the WinGet repository.

Installation and uninstallation

- Support a per-user install. This will enable users to install more easily and avoid UAC prompts.
- Ensure that your application's installation is error free, transparent, and thoughtful about its file management. Your application's installation shouldn't leave any temporary files behind.
- Avoid requiring elevated permissions to install and requiring operating system reboots when possible.
- Support silent installation. This is important for app manageability in enterprise environments.
- Ensure your app is listed in the Apps -> Installed Apps list.
- Consider using MSIX to ensure that users experience a seamless installation, update, and uninstallation experience. MSIX automatically removes the app binaries and data. For information about how packaged apps handle files and registry entries, see Understanding how packaged desktop apps run on Windows.
- For unpackaged apps, ensure that your application can be easily uninstalled through the Apps -> Installed Apps list in Settings. When your application is uninstalled, ensure that Start menu entries, files, directories, registry entries, and temporary files are also removed. Consider giving your users the option to preserve their data when they uninstall your application.
- Ensure that during uninstallation your app removes all binaries and application data. User-created content should be stored in locations like `Documents`, which can then be retained by users even after the app is uninstalled.
- Avoid installing or updating system binaries that may require a reboot.
- Integrate with RestartManager to save and restore state between OS updates.

Updates

- Support an update mechanism that allows your app to restart when its convenient for the user. Consider using the Windows App SDK Restart APIs to manage app behavior for WinUI 3 apps.

- Ensure that your update mechanism downloads only the essential changed components that need to be updated. This can minimize the network bandwidth required.
- Ensure that you provide a way to update and repair your app. Consider MSIX, which automatically handles update repair. For more information, see Auto-update and repair apps.
- Consider push notification-based updates or checking for available updates at app startup or at restart.

Additional resources

- MSIX documentation
- Windows Installer Best Practices

Accessibility

Accessible Windows applications support rich and inclusive experiences for as many people as possible, including those with disabilities (both temporary and permanent), personal preferences, specific work styles, or situational constraints (such as shared work spaces, driving, cooking, glare, and so on).

In fact, the World Health Organization defines disability not as a personal characteristic, but rather as a mismatched interaction between a person and the physical and digital world around them.

> ## Accessibility is good for both people and business
>
> **Accessibility is a responsibility**
>
> More than 1 billion people worldwide experience some form of disability. However, only 1 in 10 have access to the assistive technology needed to fully participate in our economies and societies. Typically, the unemployment rate for people with disabilities is twice that of people without a disability. And disabilities—whether situational, temporary, or permanent—can affect any of us at any time.
>
> **Accessibility is an opportunity**
>
> According to the Microsoft Accessibility Approach Datasheet: Inclusive organizations that embrace best practices for employing and supporting persons with disabilities in the workplace outperform their peers and do better at attracting and keeping top talent. Millennials, who will be 75% of the global workforce by

> 2020, typically choose employers who reflect their values. Diversity and inclusion top that list.

Incorporating accessibility

Incorporating accessibility into your Windows apps can maximize user engagement, increase product satisfaction, and encourage product loyalty. Proactively designing and implementing accessible experiences typically results in reduced development and maintenance costs over the long-term.

For detailed guidance on building accessible Windows apps, see Accessibility in Windows 11 and Windows 10.

Accessibility testing

Accessibility Insights is a powerful suite of tools for developers to test the accessibility of their apps and services. Here are some tools to leverage in testing accessibility:

1. Inspect in Accessibility Insights for Windows. Inspect the accessibility tree to find low-hanging fruit like hints in labels, incorrect roles, etc.
2. Event monitoring in Accessibility Insights for Windows · Accessibility Insights. See Supporting UI Automation Control Types for more info on event monitoring.
3. Run Accessibility Insights automated checks in your PRs or CI/CD. For more info, see axe-pipelines-samples.
4. Fix all bugs you find, they all have direct impact on accessibility.

Security and privacy

An insecure application can be an entry point that allows an attacker to perform malicious activities. Even if your app doesn't have security bugs, attackers can use your app to initiate their attacks through phishing and other forms of social engineering that violate security and privacy boundaries. The best practices in this section will help you mitigate risks related to security and user privacy.

Security guidelines

- Follow the Security Development Lifecycle for all development.
  - Threat modeling can help you avoid security flaws.
  - Using secure libraries, languages, and tools minimizes implementation flaws.
  - Secure defaults can prevent security issues caused by user error.
- Don't require administrative privileges to *install* your app.

    - Ideally, your app should support both administrative installs and per-user installs.
    - Using MSIX packaging is one way to achieve this.
- Don't require administrative privileges to *run* your app.
    - If there are certain features that need administrative privileges, consider separating them into their own processes to reduce attack surface.
- Prefer to use languages with guaranteed memory safety (such as C#, JavaScript, or Rust), especially for risky code paths (like parsing untrusted data).
- Use all security mitigations provided by your compiler and toolset (see Security Features In Microsoft Visual C++ for Visual C++).
- Always use your chosen language or framework's standard libraries for cryptography and other security-sensitive code. *Do not try to build your own*.
- Digitally sign all components of your application – not just the installer, but also the uninstaller (if you have one). Also sign all the EXE, DLL, and other executable files that make up your app.
    - Digital signatures enable the user to verify the authenticity of your app and allow Enterprise admins to secure their devices using Windows Defender Application Control.
    - Using MSIX packaging is one way to achieve this.
- Ensure all network communication is over a secure transport, such as SSL.
- Provide guardrails or other mitigations that can help protect users from accidentally performing harmful actions, even when coerced into doing so by attackers.
    - Simple "Are you sure you want to do *X*? *Yes / No*" dialogs are typically not effective, because users have been conditioned to click "Yes."

Most modern apps collect and use a large amount of data – including personal data – for various reasons. Telemetry, product improvement, and monetization are three common reasons for using data, but users and regulators alike are becoming more sensitive to the privacy implications of these practices. They expect transparency and control over the data collected and used by apps. Use the following tips to help meet the privacy needs of your users.

Privacy guidelines

- Ensure that your app provides an accurate Privacy Policy. Ideally, provide both a summary document written for a casual audience (your users) in addition to a long-form legal policy (written for your lawyers).
- Familiarize yourself with privacy regulations in the markets where your app will be available, and ensure your app meets or exceeds any requirements for disclosure, usage rights, deletion requests, etc.

- Ensure you're collecting the least amount of personal data needed to complete your app's experiences.
  - Don't collect data "just in case" – there should be a valid reason for collecting all data, e.g. to improve the customer's experience or to facilitate monetization.
- Always get the user's consent before collecting and storing personal data and provide the user with an easy way to revert their decision in the future. Avoid "dark patterns" such as making the "Yes" button larger or more prominent than the "No" button in a consent dialog.
  - Consult with applicable regulations to determine what specific disclosures and consent is required for specified kinds of data. For example, some regions may allow users to view, change, or delete the data you have stored about them.
- If you must transmit data over the network, always use secured connections, e.g. over TLS.
- Avoid storing personal data in a centralized location (e.g. website). If you must store personal data, minimize the amount of data you store, store it only for as long as strictly necessary, and ensure it is securely encrypted.
- Verify that any 3rd-party libraries or SDKs you use also have good privacy practices. Note this is *not* limited to just advertising SDKs – any library that connects to the internet may impact the privacy of your app's users.

Related articles

- Create your first WinUI 3 project
- Windows Developer FAQ
- Things you can do to make your app great on Windows 11

Windows developer FAQ

FAQ

This article is meant to promote a common understanding of the Windows development experience and product direction.

Today's Windows app development landscape offers a variety of frameworks and technologies to choose from. This FAQ provides guidance on how to decide which framework to use for your Windows app development projects. It covers the following topics:

- Getting started and the Windows app development landscape.
- Native Windows-only app development with WinUI, Windows Presentation Foundation (WPF), and Windows Forms (WinForms).
- The Windows Software Development Kit (SDK) and Windows App SDK.
- Targeting Windows as part of your cross-platform development strategy.
- Hybrid and web app development with .NET MAUI, Blazor, and ASP.NET Core.
- How to choose an approach while understanding Microsoft's investments.

The Windows app development landscape

Where can I find a straightforward overview of Windows development technologies?

For a great review of today's options for Windows developers, watch the most recent Windows Dev Chat episode, Choosing your ideal dev platform, where they discuss WinUI, .NET MAUI, React Native, Blazor, and Progressive Web Apps (PWAs).

You can also refer to the overview of app development options for Windows developers.

Why is client app development still crucial for modern digital transformation in the era of cloud services?

In the age of cloud services, client app development remains a critical component of modern digital transformation. For developers, building client applications is essential not just for reach but for delivering responsive, meaningful interactions on user devices.

Here's why client apps still matter:

- **Device Reach:** With over 1.5 billion Windows devices and more than 5 billion Android and iOS devices globally, client apps let you bring your applications directly to users on their devices of choice.
- **Gateway to Intelligent Services:** Client apps are often the first interaction users have with your services. They offer a rich, interactive interface that allows you to showcase intelligent features and differentiate your product from others.
- **Scalability with Cloud Integration:** A well-integrated client app can sync effortlessly with backend cloud services, enabling real-time data access and seamless scalability as your user base grows.
- **Enhanced Productivity and User Loyalty:** A thoughtfully designed app can enhance productivity and keep users engaged with your product or service over time.

Native Windows-only app development

What is the Windows App SDK?

The Windows App SDK is a Windows application development platform that allows you to create beautiful, modern, backwards-compatible (down to Windows 10 1809) desktop apps. WinUI 3 is the UI framework that ships with Windows App SDK.

What's the difference between the Windows App SDK and the Windows SDK?

Both are software development kits (SDKs) that let you build Windows apps.

The **Windows App SDK** is a new development platform that lets you build modern desktop apps that can be installed across Windows versions (down to Windows 10 1809). When you use Windows App SDK to build an app, you'll have access to the latest Windows development platform features. The Windows App SDK includes WinUI 3.

The **Windows SDK** is a development platform that lets you build UWP apps and Win32/desktop apps. It's designed around Windows APIs that are coupled to particular versions of the OS.

The Windows App SDK doesn't replace the Windows SDK. Instead, the Windows App SDK is a complement to the Windows SDK. It provides a convenient, OS-decoupled abstraction around the rich catalogue of Windows OS APIs that you can already access using the Windows SDK. When you build apps with Windows App SDK, you might end

up using some Windows SDK APIs depending on the functionality that you need. Over time, more Windows SDK functionality will be lifted into the Windows App SDK.

I'm building a new team to develop a Windows-only app. Why should I choose to develop with a native Windows framework like WinUI, WPF, or WinForms?

Here are some reasons to choose a native Windows framework for your Windows-only app:

- **Performance:** Native Windows frameworks are optimized to leverage the full power of today's Windows hardware and provide fast and responsive user experiences.
- **Integration:** Windows ships with wide varieties of APIs to build sophisticated experiences only available on Windows. The native frameworks provide deep integration with these features and APIs.
- **Native user experience:** Native Windows frameworks provide a consistent user experience across Windows devices, ensuring that your app looks and works great on all Windows devices.
- **Offline support:** Native Windows frameworks provide support for offline scenarios, enabling your app to work even when the user is not connected to the internet.
- **Monetization:** Native Windows frameworks are actively maintained and supported by Microsoft, ensuring that you have access to the latest updates and features.

Which framework should I use to leverage Microsoft's latest investments in Windows app development?

If you're building a new Windows-only app, we recommend using WinUI. WinUI is the latest native UI framework for Windows app development and is designed to work across a wide range of Windows devices. WinUI provides a modern and flexible UI framework that enables you to create visually appealing and interactive Windows apps. WinUI is part of the Windows App SDK and is designed to work best with the latest versions of Windows.

Can I use Windows App SDK / WinUI in my existing Windows app?

Note that WinUI (a UI framework) ships with Windows App SDK (a Windows platform development framework).

Generally speaking, you can't use WinUI unless you're ready to migrate your UI framework entirely. We're working on a feature called XAML islands that lets you host WinUI content in other UI frameworks (WPF, Win32).

You should be able to use elements of the Windows App SDK in any desktop app, depending on how your existing app was built. UWP apps are not supported by Windows App SDK.

This means that **WPF/MFC/WinForms** apps can use Windows App SDK APIs that are not related to WinUI. Examples of such APIs include App Lifecycle, Windowing, and Toast Notifications.

See Use the Windows App SDK in an existing project for more information.

Do I need to use Visual Studio to build WinUI apps?

We strongly recommend using Visual Studio 2022 versions 17.10 and above to develop WinUI apps. Using the latest version of Visual Studio will give you access to rich development features like Hot Reload. The latest Visual Studio installers include a **Windows application development** workload, making it easy to get started with WinUI development.

You may be able to use other IDEs and development workflows, but Visual Studio is currently the only officially supported IDE for WinUI. Note that MSBuild is required to compile projects that use XAML or WinUI.

When I build an app using Windows App SDK and WinUI 3, am I building a "WinUI app"?

Yes - "WinUI app" is the term that we recommend you use. WinUI 3 apps are generally referred to as "WinUI apps", as WinUI 2 is not a type of app but a set of components that can be used in UWP apps.

Can I incrementally update my UWP app with WinUI 2 controls to WinUI 3 by gradually

replacing WinUI 2 components with WinUI 3 components?

No. Windows App SDK can't be used in UWP apps, and WinUI 2 can't be mixed with WinUI 3. See Migrate from UWP to the Windows App SDK.

How hard is it to migrate a UWP app to WinUI?

Migrating UI components is generally straightforward (for C# and C++/WinRT). Otherwise, the cost of migrating from UWP to WinUI will depend primarily on these factors:

1. **Project file and MSBuild customization:** Migrating your project might require a significant level of effort, depending on whether you use advanced MSBuild features.
2. **.NET API migration:** If your UWP app relies on .NET, then you'll need to upgrade to .NET 6 or later. In most cases, adopting .NET 6 is straightfoward.
3. **UI component libraries:** If you use UI component libraries, then you'll need new versions of them that target WinUI 3.
4. If your UWP source code is written in the now-superseded C++/CX, then some source code porting will be involved. See Move to C++/WinRT from C++/CX.

For more info about UWP migration, see Migrate from UWP to the Windows App SDK.

If I have an existing UWP app in the Store, can I publish a new packaged WinUI app using the same identifiers?

Yes, upgraded apps can be published without having to update your application's identity. Users who have the old version will get updated to the new version. This guidance applies only to desktop apps. Xbox, HoloLens, and Surface Hub apps can't migrate to WinUI.

How do I package/distribute my WinUI app?

See Deployment overview.

Where can I find Windows App SDK migration guidance?

See Migrate from UWP to the Windows App SDK.

Do I need to use XAML markup if I want to use WinUI?

No. UI controls can be created in code. But there are many benefits to representing your WinUI user interface in the form of declarative XAML markup, such as improved developer experience.

If you're migrating from UWP to WinUI, you'll probably be able to reuse a lot of your XAML markup and UI-related code (but you'll have to update some of the syntax). If you're migrating from WPF to WinUI, you'll be able to reuse a lot of the concepts, but the control set and APIs will be different.

Does Visual Studio have a design surface / UI designer for WinUI?

Not yet. We recognize that this is a gap in the WinUI developer experience. Tools like XAML Hot Reload can help in many scenarios. Work has started on a Visual Studio UI designer for WinUI in Windows App SDK 1.7, but there isn't a timeline for the release of this feature yet.

Does Windows App SDK include WinUI 3?

Yes. WinUI 3 ships as a part of the Windows App SDK.

Does Windows App SDK include WinUI 2?

No. WinUI 2 is part of the UWP platform.

Are WinUI 2 and WinUI 3 built on the same technology?

Not quite. Although WinUI 3 started from the WinUI 2 codebase, they are distinct technologies. Both WinUI 2 and WinUI 3 are XAML-based UI frameworks that work across .NET and C++. Note that WinUI 2 and WinUI 3 are not compatible with each other.

Can I use WinUI 3 without using Windows App SDK?

No. WinUI 3 ships as part of the Windows App SDK.

Can I use WinUI 3 in an unpackaged app?

Yes. All technologies in the Windows App SDK work in unpackaged apps, including WinUI 3.

What's the difference between XAML Islands and WinUI 3?

XAML Islands allow you to host modern WinUI controls alongside existing Win32 UI from other frameworks, such as WinForms and WPF. Today, XAML Islands are supported with *most* system XAML and WinUI 2 controls. See Host WinRT XAML controls in desktop apps (XAML Islands) for more information. XAML Islands support for WinUI 3 controls is available starting in Windows App SDK 1.4.

If I create a WinUI app, will it look modern on both Windows 11 and Windows 10?

Yes, your app's UI will inherit the latest Fluent UI design principles on all supported versions of Windows 11 and Windows 10 down to version 1809 in both packaged and unpackaged scenarios.

Can I use Mica or Acrylic backgrounds in apps built with Windows App SDK?

Yes. See Apply Mica or Acrylic materials in desktop apps for Windows 11.

Where can I find WinUI samples?

See Sample and resources. Some notable repositories:

- WindowsAppSDK-Samples : Demonstrates how to use specific Windows App SDK API sets.
- WinUI 3 Demos : Contains demos used during Microsoft's WinUI presentations.

- WinUI Gallery: Showcases WinUI and Windows App SDK. You can also get the WinUI Gallery in the Microsoft Store.

If I have already invested heavily in WPF, should I continue to use WPF or should I consider migrating to WinUI?

If you have already invested heavily in WPF, you can continue to use WPF for your existing apps. WPF is a mature and stable framework that is widely used by developers to build Windows desktop apps.

Consider leveraging the .NET Upgrade Assistant to migrate your .NET Framework WPF apps to the latest .NET platform. The .NET Upgrade Assistant is a tool that helps you migrate your .NET Framework apps to the latest supported versions of .NET. The .NET Upgrade Assistant analyzes your existing codebase and provides guidance on how to update your code.

If I build a new WPF app, will it look dated compared to other new Windows apps?

When developing a WPF application with .NET 9 or later, you can ensure your app matches the sleek, modern look of Windows 11. The new Fluent theme for WPF introduces a contemporary Windows 11 aesthetic to WPF applications, complete with integrated Light/Dark mode and system accent color support. This update not only modernizes the appearance of your app but also enhances user engagement by providing a polished and cohesive user experience.

My team is comfortable building WinForms apps, and it suits our needs. Should we consider migrating to WinUI or another framework?

If your team is comfortable building WinForms apps and WindForms meets your needs, you can continue to use WinForms for your existing apps. WinForms is a mature and stable framework that is widely used by developers to build Windows desktop apps.

The WinForms team is investing across features and has an active community of contributors. Some current areas of investment include:

- Async support with common controls
- Dark mode

- Layout flexibility
- Desktop security features like clipboard access

Cross-platform native development

What are some reasons for building cross-platform, native apps that target Windows?

If you're targeting users across multiple OS platforms, building cross-platform apps with .NET MAUI or React Native can provide several benefits:

- **Reach:** Building cross-platform apps allows you to reach a larger audience of users on different platforms.
- **Code reuse:** Building cross-platform apps allows you to reuse code across different platforms, reducing development time and costs. It can be prohibitively expensive to build separate apps for Windows, iOS, Android, macOS, etc.
- **Consistent user experience:** Building cross-platform apps allows you to provide a consistent user experience across different platforms, ensuring that your app looks and works great on all devices.
- **Integration:** Building cross-platform apps allows you to integrate with different platforms and services, enabling you to provide a more comprehensive user experience.

Can I be confident that .NET MAUI apps will run well on Windows?

When you build a .NET MAUI app for Windows, your output is a WinUI app. While developing, .NET MAUI provides a single .NET experience across all platforms, including Windows, but under the hood, it generates platform-specific code. This ensures that your .NET MAUI app runs well on each platform and provides a native user experience.

How can .NET MAUI provide native device APIs across every platform?

.NET MAUI provides a single .NET experience across all platforms, including Windows, iOS, Android, and macOS. .NET MAUI abstracts over 60 platform-specific APIs into a single, cross-platform API that you can use in your .NET MAUI app. These APIs cover access to storage, networking, device-specific sensors, and more. If needed, you can also

access additional platform-specific APIs using dependency injection to abstract the code for each platform.

Can I start with WinUI, and later integrate .NET MAUI if I eventually want to target cross-platform scenarios?

Not at this time. Although .NET MAUI apps use WinUI when running on Windows, we recommend starting with .NET MAUI or React Native for Desktop if you anticipate the need to target multiple platforms.

Our team has strong web front-end development skills. Should we consider using React Native for Desktop?

If your team has strong web development skills, you may want to consider using React Native for Desktop. React Native for Desktop encompasses the React Native for Windows and macOS frameworks. The motto of React Native is "Learn once, write anywhere," which means that you can use your existing web development skills to build native Windows apps using React Native. React Native for Desktop is an open-source project that enables you to build native Windows and macOS apps using React Native. React Native for Desktop provides a set of APIs that allow you to access desktop OS-specific features and capabilities in your React Native apps.

Your team can leverage their JavaScript, TypeScript, and React skills to build the UI layer, which is rendered directly to native primitives. This provides native app performance and access to the capabilities of the native platform.

Read more about getting started with React Native development for Windows in the React Native for Desktop documentation.

Are any other Windows devices supported by React Native for Desktop?

React Native apps can be deployed to all devices supported by Windows 10 and higher including PCs, tablets, 2-in-1s, Xbox, and Mixed reality devices.

What should I use if I want to build apps that work on Windows and Xbox?

If your app needs to support Xbox, HoloLens, or IoT, we recommend using UWP. Windows App SDK doesn't support these platforms. For game development, we recommend using Microsoft Game Development Kit.

What should I use if I want to build apps that work on Windows and Surface Hub?

If you're targeting both Windows and Surface hub, we recommend using UWP.

Hybrid and web development

What are hybrid apps, and why should I consider building one?

Hybrid apps blend the best of web and native app development. The core of the app is built using web technologies like HTML, CSS, and JavaScript, and then wrapped in a native container that allows the app to leverage certain native platform features and hardware. They can also be distributed through app stores.

The main advantage of hybrid apps is that they allow you to build a single app that can run on multiple native platforms and on the web, reducing development time and costs. Some examples of hybrid app development platforms answer:

- Electron for desktop apps
- Ionic for mobile apps
- .NET MAUI Blazor hybrid for cross-platform apps

How do I build native-feeling progressive web apps (PWAs) on Windows?

See Web development on Windows and Overview of Progressive Web Apps.

What is a .NET MAUI Blazor hybrid app?

With .NET MAUI, Blazor apps can also run natively on Windows, iOS, Android, and macOS. This means you can create hybrid client apps that combine Blazor and .NET

MAUI components together into a single, native client app. This provides full access to the same native platform capabilities available to any other .NET MAUI apps.

To read more about how Blazor can hosted within a .NET MAUI app, see ASP.NET Core Blazor Hybrid.

Do the web components of a .NET MAUI hybrid app need to be created with Blazor?

No, the web components of a .NET MAUI hybrid app do not need to be created with Blazor. Starting with .NET 9, .NET MAUI provides a HybridWebView control that allows you to host other JavaScript UIs in your native app.

This means you can take your Angular, React, Vue or other HTML & JavaScript web app and host it within your .NET MAUI app. The hybrid control provides interop between the C# and JavaScript layers so you can call JavaScript functions from C# and vice versa.

Can any other native app types host Blazor hybrid components?

Yes, WPF and WinForms apps can also host Blazor hybrid components. This allows you to add modern web UI components to your existing WPF and WinForms apps. Note, that this is not possible with WPF or WinForms apps built on .NET Framework.

Does my entire app need to be a hybrid app, or can I mix and match native and hybrid components?

You can mix and match native and hybrid components in your app. For example, you can build the core of your app using .NET MAUI components and then add hybrid components to provide additional functionality. This allows you to leverage the best of both worlds: the performance and capabilities of native components and the flexibility and cost savings of hybrid components.

What are my choices for building .NET-based web apps that look great on modern browsers on Windows?

Web apps have the broadest reach of any client app platform. If you want to create beautiful .NET web apps on Windows, you have several options:

- ASP.NET Core apps with Razor Pages
- ASP.NET Core MVC apps
- ASP.NET Core Blazor apps, with hosting model options including:
  - Blazor WebAssembly apps
  - Blazor Server apps

Note that the hosting model for Blazor can be configured at the component level now. So, you can have a Blazor WebAssembly component hosted in a Blazor Server app.

Read more about ASP.NET Core development options in the ASP.NET Core documentation.

Choose an approach and understand Microsoft's investments

There are so many framework options for building apps that target Windows! How do I decide?

Windows is an open platform that supports many technologies. Here are some criteria that can help you decide which platform to use:

- Are you building Windows-first or cross platform?
- Do you have experience with .NET? JavaScript? Other languages?
- Do you need to access Windows-specific APIs?
- Which framework's capabilities best match your app's requirements?
- See this table for other decision factors.

When it comes to business apps, most teams want to choose based on existing skills and what the team is comfortable using.

How do I choose the best development approach for my web app?

Some things to consider when choosing a development approach for your web app include:

- Blazor is recommended for building front-end web apps with .NET. By using Blazor, you can now build your entire front-end and back-end with .NET, saving time and money. It is particularly well-suited for enterprise line-of-business apps on today's devices.
- JavaScript web apps still make sense if you want to leverage your team's existing skills or investments in JavaScript, or if you need to integrate with existing JavaScript libraries or frameworks.
- Existing apps that use older frameworks, like Web Forms, MVC, or Razor Pages, are still supported and can continue to be developed and maintained using those frameworks.

Who is building apps with WinUI today?

Many customers are building with WinUI today, including Adobe and Apple:

- Adobe Fresco, a free drawing and painting app for Windows.
- Apple created the Apple Music, Apple TV, and Apple Devices apps with WinUI and the Windows App SDK.

Microsoft has also created several apps with WinUI, including the Windows 11 File Explorer and Photos apps.

Who is building .NET MAUI apps today?

Many customers are using .NET MAUI to build their cross-platform apps today, including Microsoft. For example, the Microsoft Azure mobile app was built using .NET MAUI.

Find out who else is building apps with .NET MAUI on the .NET customers showcase site.

Who is building WPF apps today?

Most of Microsoft Visual Studio's user interface is built with WPF. The Visual Studio IDE is a great example of a complex, high-performance WPF app.

Who is building Blazor apps today?

GE Digital's FlightPulse airline system puts sensor data and analytics into the hands of pilots to improve safety and efficiency. The backend configuration of everything the pilots see is all done with Blazor.

You can read more Blazor customer stories on the .NET site.

UWP and WinUI 2

Can UWP apps be distributed outside of the Microsoft Store?

Yes. If your MSIX package is signed, the signing certificate must be valid and trusted on the target device.

Can I mix UWP XAML UI controls with Win32, WPF, or WinForms UI controls?

Yes - XAML Islands lets you do this. Learn more about XAML Islands.

Packaging, deployment, and updates

What's the difference between apps that are packaged, unpackaged, and packaged with external location?

For definitions of apps that are packaged, unpackaged, and packaged with external location, see Deployment overview. That topic also explains the advantages and disadvantages of each option.

Will my WinUI app automatically update for end-users?

A WinUI app can be delivered through the Store, an .appinstaller file, or in your existing MSI or setup.exe package. The Store and the AppInstaller support automatic updates for end-users who have automatic updating enabled, but your MSI/setup.exe app needs to have its own updater.

Can I use Windows App SDK without using MSBuild?

Generally speaking, no. WinUI and Windows App SDK require MSBuild, which is why Visual Studio is a prerequisite to developing with WinUI and Windows App SDK. While it's technically possible to build Windows App SDK apps that don't use WinUI using other toolchains, this isn't supported.

Performance and optimization

What can I do to make my Windows app feel great to end-users?

See Make apps great for Windows.

Compatibility

Will my users ever have to update Windows to use my WinUI app?

Users who have Windows 10, version 1809 and beyond will be able to install your WinUI apps without updating their OS.

Can I target Arm64 with my WinUI app?

Yes.

Deprecations and migrations

Are UWP / WinUI 2 deprecated?

No. UWP and WinUI 2 are still supported and will receive bug, reliability, and security fixes. However, most new features and capabilities will only be added to WinUI 3.

> **Note**
>
> UWP support for .NET 9 is available in preview. This preview support provides a path for UWP apps to modernize using the latest version of .NET.
>
> There are two main goals of the .NET 9 UWP support. First, it provides a better migration path for UWP developers who want to migrate to WinUI 3. Second, it

enables developers to remove dependencies on .NET Native, so they can benefit from the latest .NET and C# features, no matter what their app model is.

For more information, see the Modernize your UWP app with preview UWP support for .NET 9 and Native AOT blog post.

When should I migrate a UWP / WinUI 2 app to WinUI 3?

UWP developers should feel no pressure to migrate their apps to the WinUI 3, if they are happy with UWP and its feature set. The best choice for some apps may be to never migrate. For apps that want to benefit from the latest Windows platform and .NET investments from Microsoft, these apps should consider moving to the Windows App SDK. See Migrate from UWP to the Windows App SDK).

When should I *not* migrate a UWP + WinUI 2 app to WinUI 3?

We recommend continuing to use UWP if you're building for Xbox, Surface Hub, or HoloLens.

Is WPF deprecated?

No. WPF is still supported, recommended, and receives feature updates as well. See the WPF roadmap on GitHub.

Is WinForms deprecated?

No. WinForms is still supported and receives feature updates. See the Windows Forms Roadmap on GitHub.

Is the Windows Runtime (WinRT) deprecated?

No. WinRT refers to an application binary interface (ABI) that lets you interop between multiple languages. WinRT is the evolution of COM. Windows App SDK provides most of its functionality through WinRT APIs.

Release notes

Where can I find release notes for Windows App SDK?

See Stable channel release notes and Preview channel release notes.

Related content

- Windows developer glossary
- Overview of app development options

Windows developer glossary

Article • 12/17/2024

The following glossary of terms is meant to promote a common vocabulary among Windows developers.

App lifecycle management (ALM)

Describes the management of your application's execution state: not running, running in background, running in foreground, suspended, and so on. See Windows 10 universal Windows platform (UWP) app lifecycle.

Application model

Often referred to as "app model". The combination of deployment, isolation, lifecycle, and presentation components that are unique to a given application development technology. For example: Windows App SDK / WinUI 3 apps run on the Win32 app model, while UWP / WinUI 2 run on the UWP app model.

Application packaging

Describes the way in which your app is packaged before being deployed and installed by users. An app can be packaged, unpackaged, or packaged with external location (see the Windows developer FAQ).

Bootstrapper

A redistributable component providing an API to find and load the Windows App SDK framework package for the calling process. In a packaged with external location or unpackaged app, you can opt to load the Windows App SDK framework package explicitly by calling Bootstrapper APIs such as MddBootstrapInitialize. Also see Reference the Windows App SDK framework package at run time.

C++/WinRT

C++/WinRT is a standard C++17 language projection for Windows Runtime (WinRT) APIs, implemented as a header-file-based library, and designed to provide you with first-class access to modern Windows APIs. C++/WinRT.

Dynamic Dependencies

Dynamic Dependencies makes framework packages accessible to all kinds of apps: packaged and unpackaged.

Fluent Design

Fluent Design is a design system that lets you create reusable cross-platform user experiences. Fluent 2 is the latest design system for Windows and is used by WinUI.

GitHub Copilot

An AI pair programmer that helps you write code faster and with less work in Visual Studio or Visual Studio Code (VS Code). See AI-assisted development in Visual Studio for more information.

Hot Reload

An app development feature that allows you to update your application's code and observe your changes while your application runs, eliminating the need to stop, rebuild, and re-run your apps while developing. See Write and debug running code with Hot Reload.

Hybrid app

An app that uses multiple technologies. For example, a .NET MAUI app that uses Blazor to render web content in a WebView2 control. See ASP.NET Core Blazor Hybrid for more information.

Hybrid CRT linkage

A C/C++ runtime library linkage technique that simplifies deployment. Also referred to simply as *Hybrid CRT*. See Hybrid C/C++ runtime library linkage (hybrid CRT linkage).

Managed apps

"Managed" refers to the "managed runtime" of .NET, which provides managed services such as garbage collection and security assurances. If you're building an app with .NET, you're building a managed app.

Microsoft Foundation Classes (MFC)

You can use Microsoft Foundation Classes (MFC) to create complex user interfaces with multiple controls. You can use MFC to create applications with Office-style user interfaces. See: MFC desktop applications.

MSIX (Microsoft Installer package format)

MSIX is a Windows app package format that combines the best features of MSI, .appx, App-V, and ClickOnce to provide a modern and reliable packaging experience. It's a modern application package format that lets you easily deploy your Windows applications. MSIX can be used to package apps built using Windows App SDK, Win32, WPF, or Windows Forms. When you use MSIX to deploy your apps, your app is a *packaged* app. A packaged app can check for updates, and can control when updates are applied. What is MSIX?.

Native apps

Traditionally, "native" refers to applications built without using the .NET runtime. In this case, "native" is synonymous with "unmanaged", and can be used to describe apps that manage their own memory and security concerns. Alternatively, some developers use "native" to indicate that an application has been built to run specifically on Windows, calling Windows APIs directly.

.NET MAUI

.NET Multi-platform App UI. A cross-platform framework for creating native mobile and desktop apps with C# and XAML. An evolution of `Xamarin.Forms` extended from mobile to desktop scenarios, with UI controls rebuilt from the ground up for performance and extensibility. What is .NET MAUI?.

Packaged app

For definitions of apps that are packaged, unpackaged, and packaged with external location, see Deployment overview. That topic also explains the advantages and disadvantages of each option.

Packaged app with external location

For definitions of apps that are packaged, unpackaged, and packaged with external location, see Deployment overview. That topic also explains the advantages and disadvantages of each option.

Progressive web app (PWA)

An application that you build by using web technologies, and that can be installed and can run on all devices, from one codebase. See Overview of Progressive Web Apps (PWAs) for more information about building PWAs.

Project Reunion

The codename for the Windows App SDK. No longer in use.

React Native

React Native is a development platform from Meta which allows developers to build fully native cross-platform apps using JavaScript, TypeScript, and React.

React Native for Desktop

React Native for Desktop brings React Native support to the Windows 10 and Windows 11 SDKs, enabling developers to use JavaScript to build native Windows apps for all devices supported by Windows 10 and Windows 11. This includes PCs, tablets, 2-in-1s, Xbox, Mixed reality devices, etc. The term React Native for Desktop encompasses both React Native for Windows and React Native for macOS.

Responsible AI (RAI)

A set of recommended responsible development practices to use as you create applications and features. See Developing Responsible Generative AI Applications and Features on Windows to learn more about the RAI principles and how they apply to Windows development.

Universal Windows Platform (UWP)

An application development platform and application model that uses Windows Runtime (WinRT) APIs to deliver packaged apps. UWP apps run in a sandboxed environment, and they inherit the security of the UWP platform. Learn more about UWP.

> **Note**
>
> The preferred approach for writing Windows apps is to use the Windows App SDK and WinUI. Another option is WPF.

Unmanaged app

Apps that aren't managed by the .NET runtime. If you're handling your own memory management, you're building an unmanaged app.

Unpackaged app

For definitions of apps that are packaged, unpackaged, and packaged with external location, see Deployment overview. That topic also explains the advantages and disadvantages of each option.

Visual Studio extension (VSIX)

Lets you create, package, and deploy Visual Studio extensions. Get started with the VSIX Project template.

WebView2

A control that allows app developers to embed web content (HTML/CSS/JS) in their native apps using the Microsoft Edge (Chromium) rendering engine. You can use WebView2 in WinUI 3, Win32 C++, WPF, and WinForms, and it offers a developer preview for WinUI 2 / UWP support. See Introduction to Microsoft Edge WebView2.

Windows API

Refers to the entire set of Windows APIs including Win32 APIs, COM APIs, UWP WinRT APIs, and the WinRT/Win32 APIs that are part of WinAppSDK and WinUI 3.

Windows App SDK

A set of new developer components and tools that represent the next evolution in the Windows app development platform. The successor to UWP / WinUI 2 for desktop application development. It lifts libraries from the OS into a standalone SDK that you

can use to build backwards-compatible desktop apps. See Overview of app development options.

Windows Forms

Also known as WinForms. A UI framework for building Windows desktop applications. It is a .NET wrapper over Windows user interface libraries, such as User32 and GDI+. It's a battle-tested way to create desktop applications using a visual designer within Visual Studio. See Desktop Guide (Windows Forms .NET).

Windows Presentation Foundation (WPF)

A UI framework for building Windows desktop applications. WPF applications are based on a vector graphics architecture. This enables applications to look great on high DPI monitors, as they can be infinitely scaled. See What is Windows Presentation Foundation (WPF)?.

Windows SDK

The Windows SDK is a collection of headers, libraries, metadata, and tools that allow you to build desktop and UWP Windows apps. The Windows SDK is not the same as the Windows App SDK.

WinUI

WinUI is the modern native user interface (UX) framework for both Windows desktop and UWP applications. WinUI.

WinUI 2

WinUI 2 is tightly integrated with Windows SDKs, and provides official native Windows UI controls and other user interface elements for UWP applications (and desktop applications using XAML Islands). See WinUI 2.

WinUI 3

The latest and recommended UI framework for Windows desktop apps. This framework is made available through the Windows App SDK, and has been decoupled from the Windows operating system. WinUI 3 uses Fluent Design to provide a native UX framework for Windows desktop apps. It will feel very familiar if you've worked with

WinUI 2. Note that WinUI 3 apps are commonly referred to as "WinUI apps". See WinUI 3.

XAML Islands

XAML Islands lets you host WinRT XAML controls in non-UWP desktop (Win32, WinForms, WPF) apps starting in Windows 10, version 1903. Host WinRT XAML controls in desktop apps (XAML Islands).

Related content

- Windows developer FAQ
- Overview of app development options

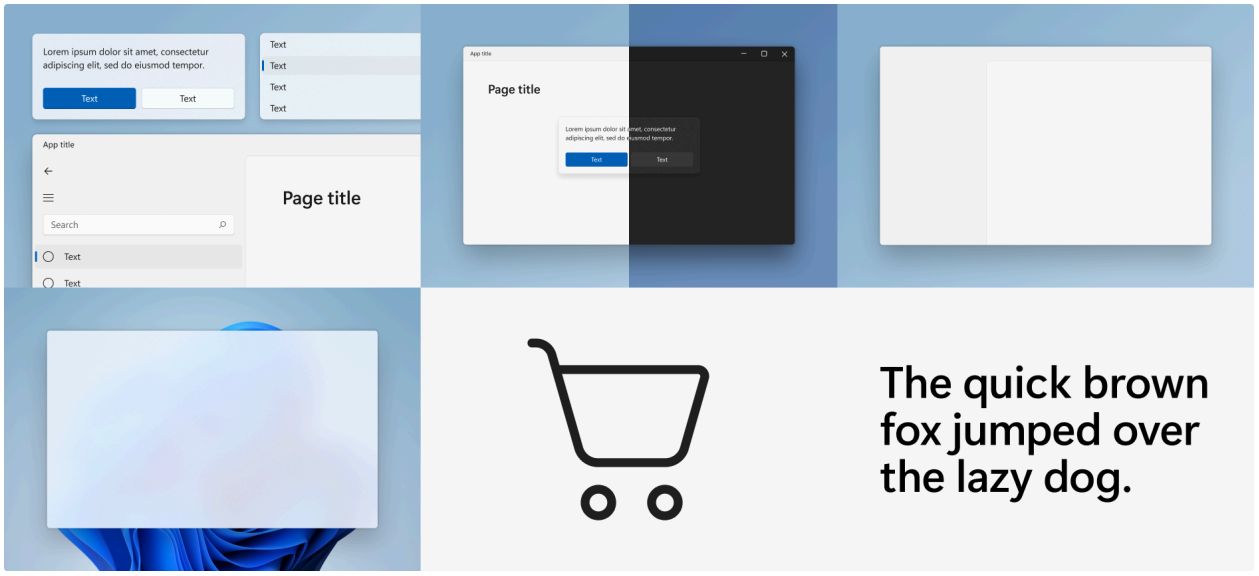
Design for Windows apps

Article • 11/05/2024

Here, you'll find design guidelines and examples for creating Windows app experiences. Windows 11 incorporates Fluent's design language and principles for a cohesive look and feel that's uniquely Microsoft.

To get the building blocks for crafting Windows experiences, use WinUI. These components incorporate Fluent's design language, so you can be confident you're building great experiences within the Fluent ecosystem.

Windows 11 design principles



Windows 11 marks a visual evolution of the operating system. We have evolved our design language alongside of Fluent to create a design that is human, universal, and truly feels like Windows.

The design principles below have guided us throughout the journey of making Windows the best-in-class implementation of Fluent.

Effortless

Windows 11 is faster and more intuitive. It's easy to do what I want, with focus and precision.

Calm

Windows 11 is softer and decluttered; it fades into the background to help me stay calm and focused. The experience feels warm, ethereal, and approachable.

Personal

Windows 11 adapts seamlessly to the way I use my device. It bends and flexes to my individual needs and preferences so that I can truly express myself.

Familiar

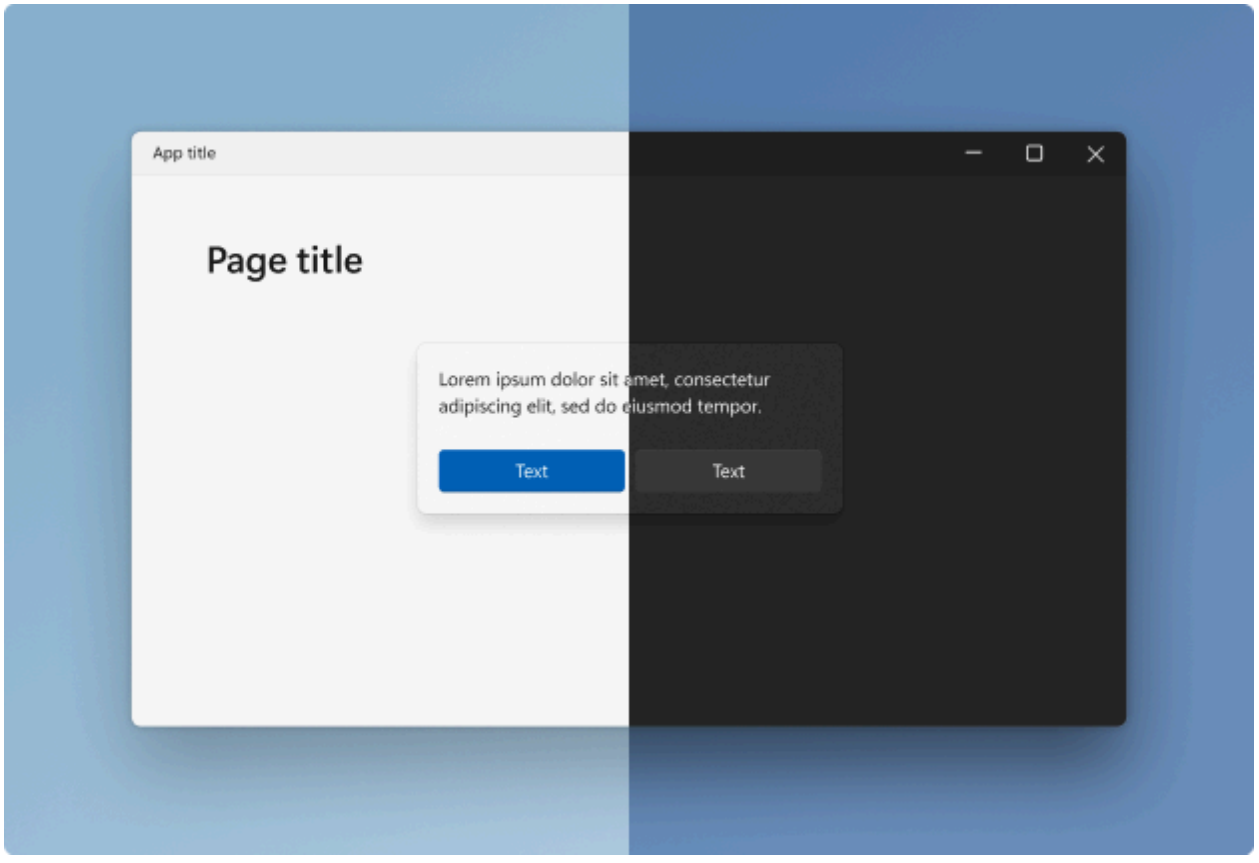
Windows 11 balances a new, refreshed look and feel with the familiarity of the Windows I already know. There is no learning curve; I can pick it up and go.

Complete + Coherent

Windows 11 offers a visually seamless experience across platforms. I can work in many platforms and still have a consistent Windows experience.

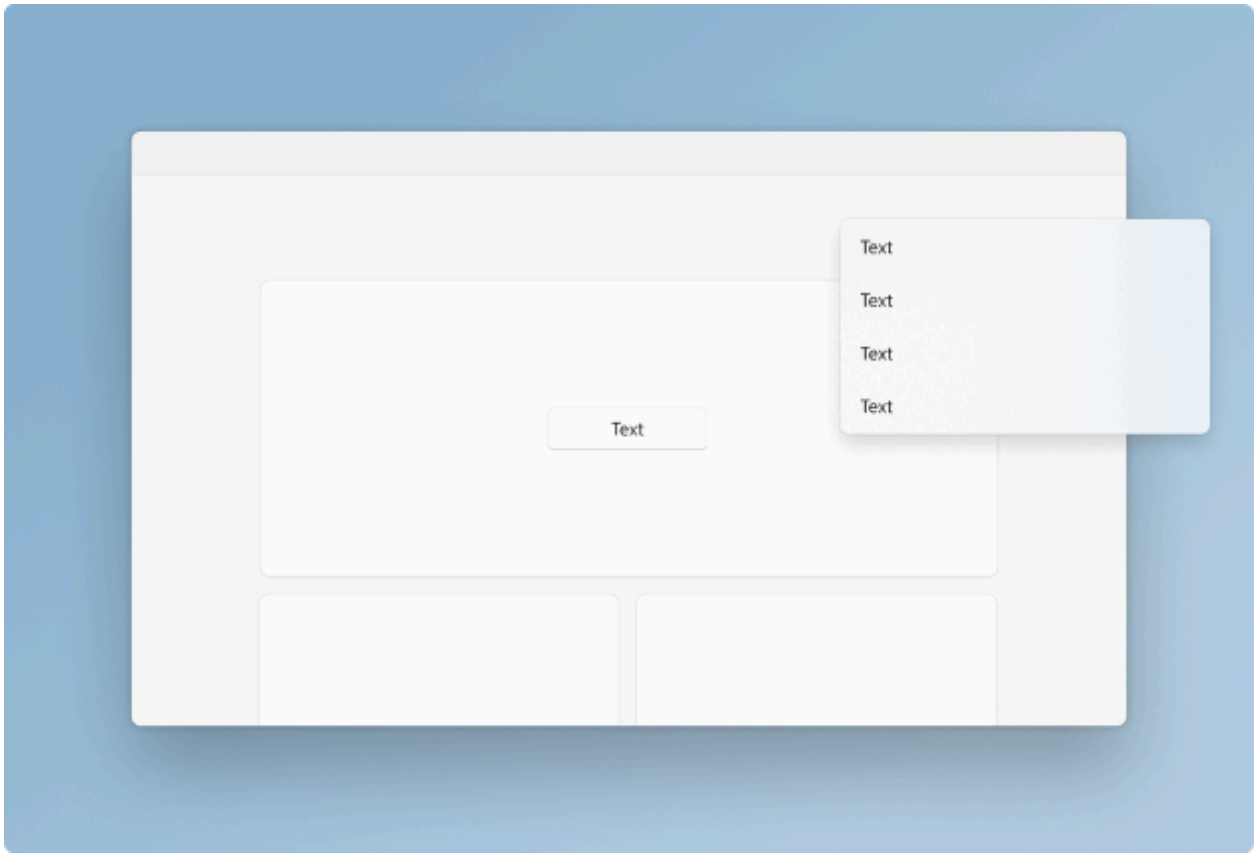
Windows 11 signature experiences

Signature experiences are the design elements Windows 11 uses to express its visual language, while maintaining a coherent look and feel across all Fluent experiences.



Color

Color helps users focus on their tasks by indicating a visual hierarchy and structure between user interface elements. Windows 11 uses color to provide a calming foundation, subtly enhancing user interactions and emphasizing significant items only when necessary.

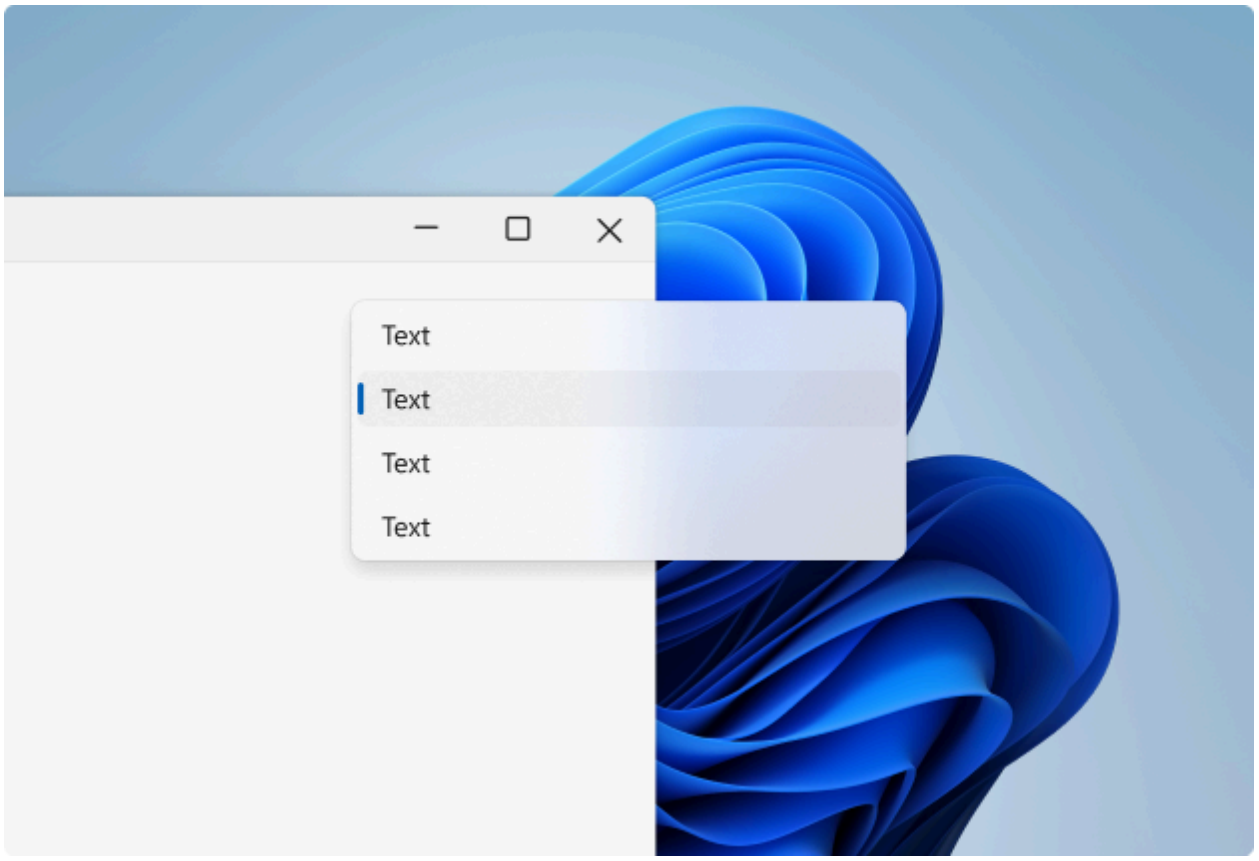


Elevation and layering

Elevation and layering is the concept of overlapping one surface with another, creating two or more visually distinguished areas within the same surface. Windows 11 uses elevation and layering as its foundation for app hierarchy.

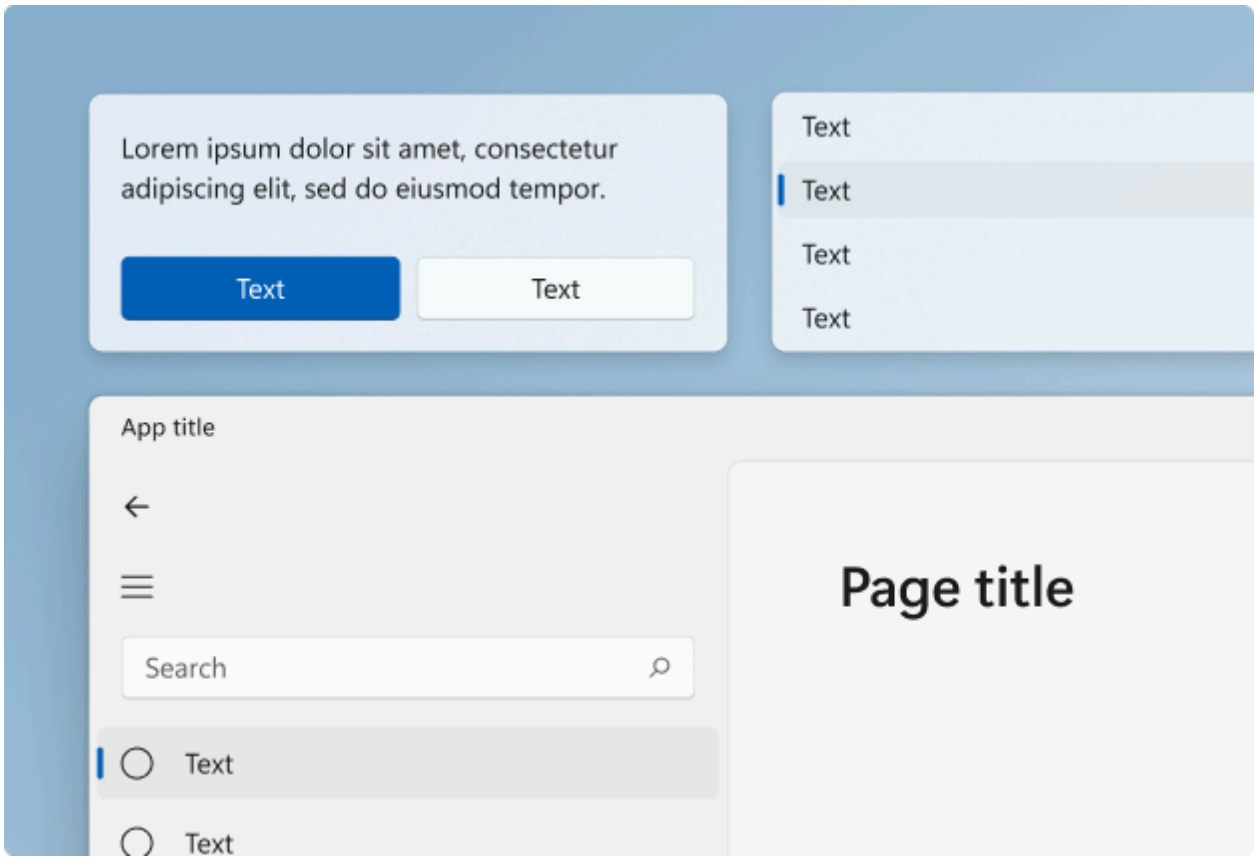
Iconography

Iconography is a set of visual images and symbols that help users understand and navigate your app. Windows 11 iconography has evolved in concert with our design language. Every glyph in our system icon font has been redesigned to embrace a softer geometry and more modern metaphors.



Materials

Materials are visual effects that make UI surfaces resemble real life artifacts. Windows 11 uses materials to keep the UI connected to its environment. Materials bring surfaces to life and help us distinguish between focused and unfocused applications.



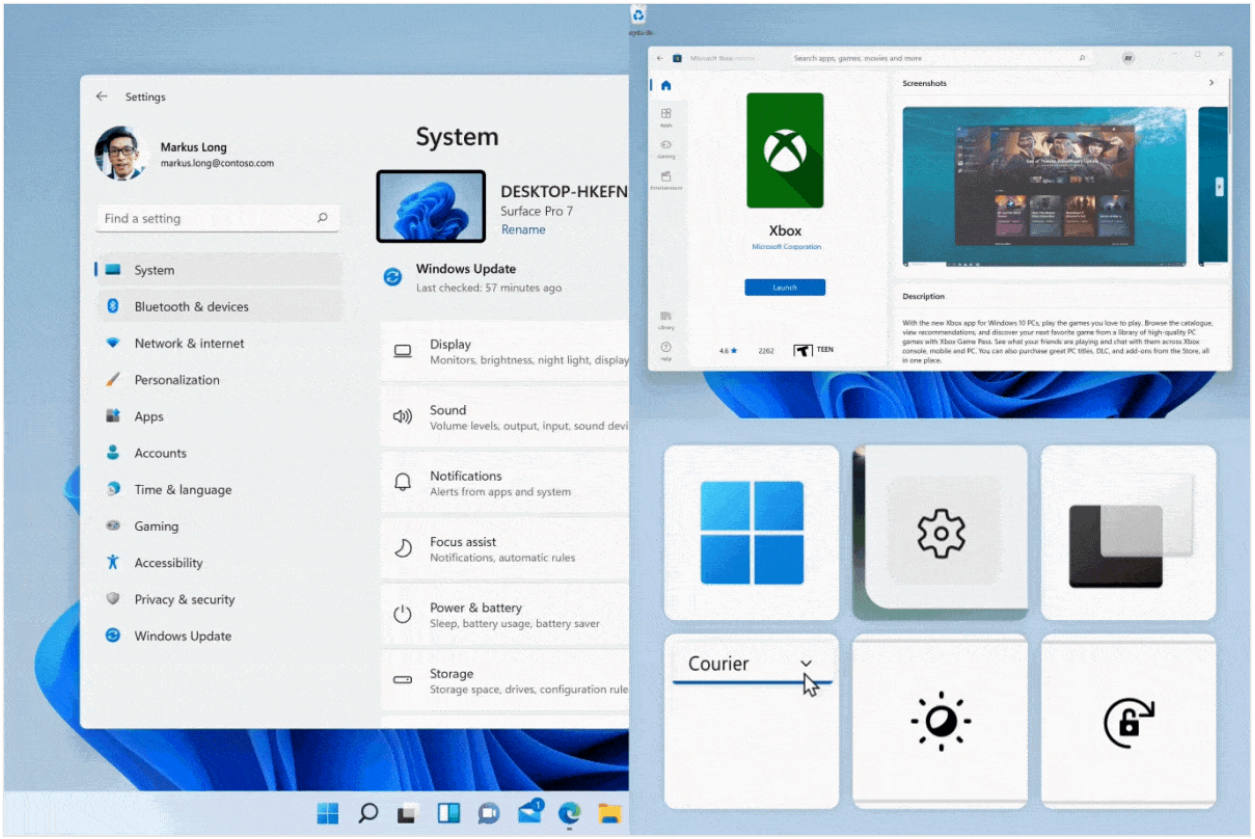
Shapes and geometry

Geometry describes the shape, size, and position of UI elements on screen. These fundamental design elements help experiences feel coherent across the entire design system. Windows 11 features updated geometry that creates a more approachable, engaging, and modern experience.

The quick brown fox jumped over the lazy dog.

Typography

As the visual representation of language, the main task of typography is to communicate information. The Windows 11 type system helps you create structure and hierarchy in your content in order to maximize legibility and readability in your UI.



Motion

Motion describes the way the interface animates and responds to user interaction. Motion in Windows is reactive, direct, and context appropriate. It provides feedback to user input and reinforces spatial paradigms that support way-finding.

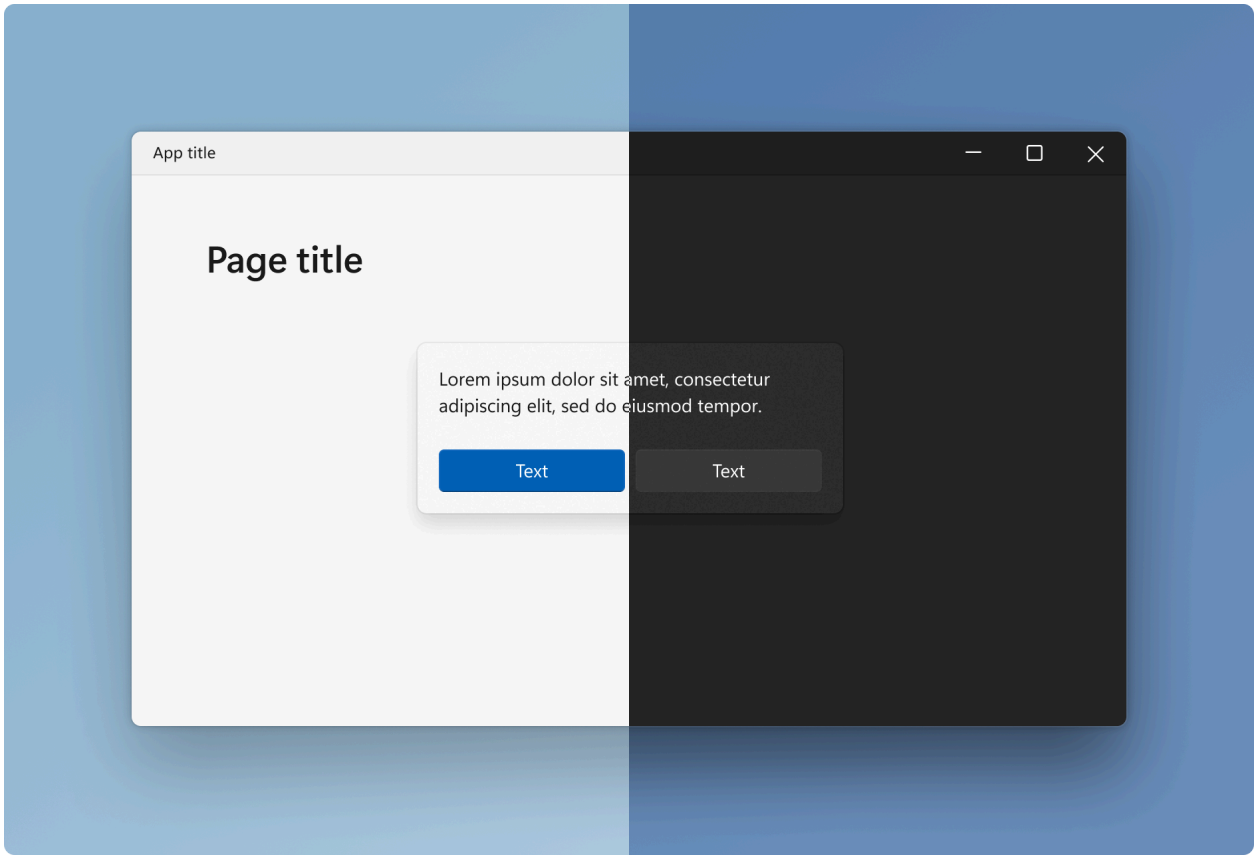
Color in Windows

Article • 11/05/2024

Windows employs color to help users focus on their tasks by indicating a visual hierarchy and structure between user interface elements. Color is context appropriate and used to provide a calming foundation, subtly enhancing user interactions and emphasizing significant items only when necessary.

> **Tip**
>
> This article describes how the Fluent Design language is applied to Windows apps. For more information, see Fluent Design - Color.

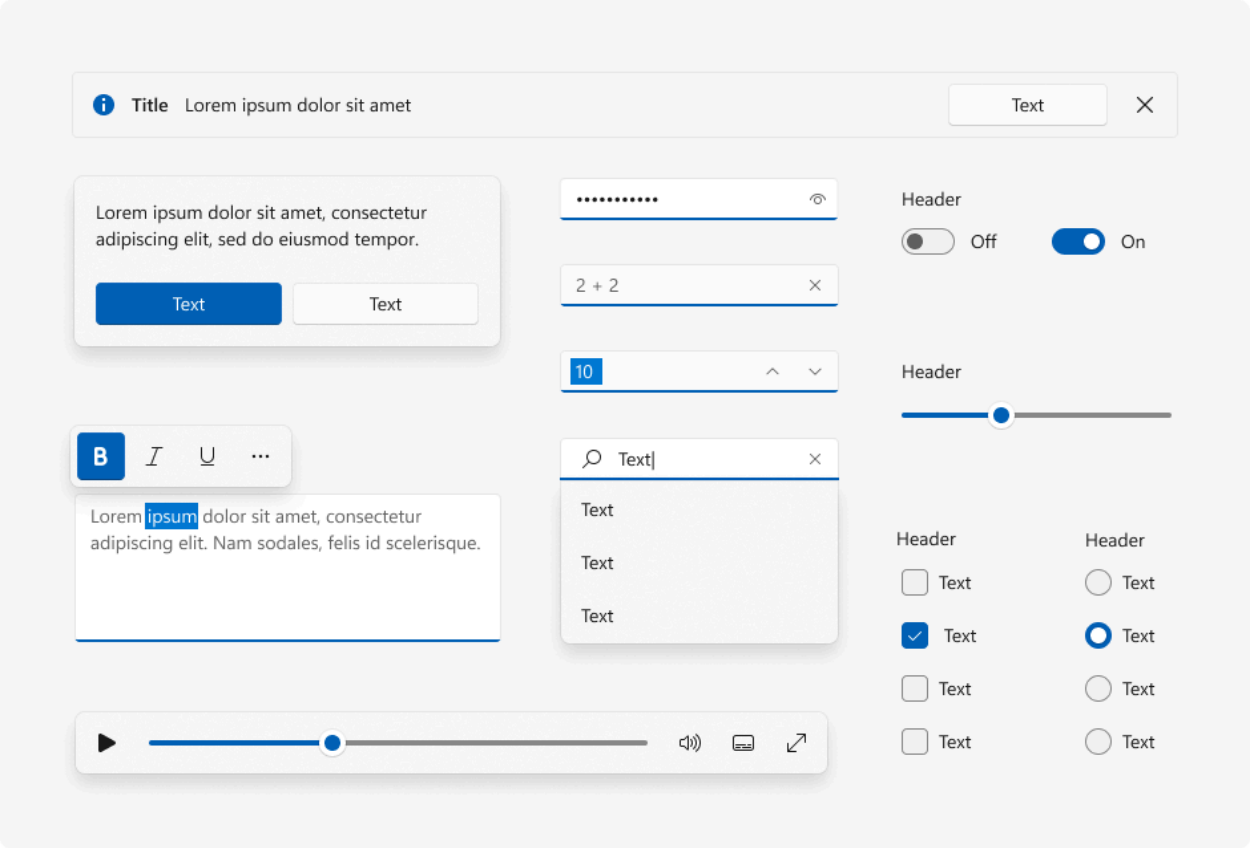
Color modes

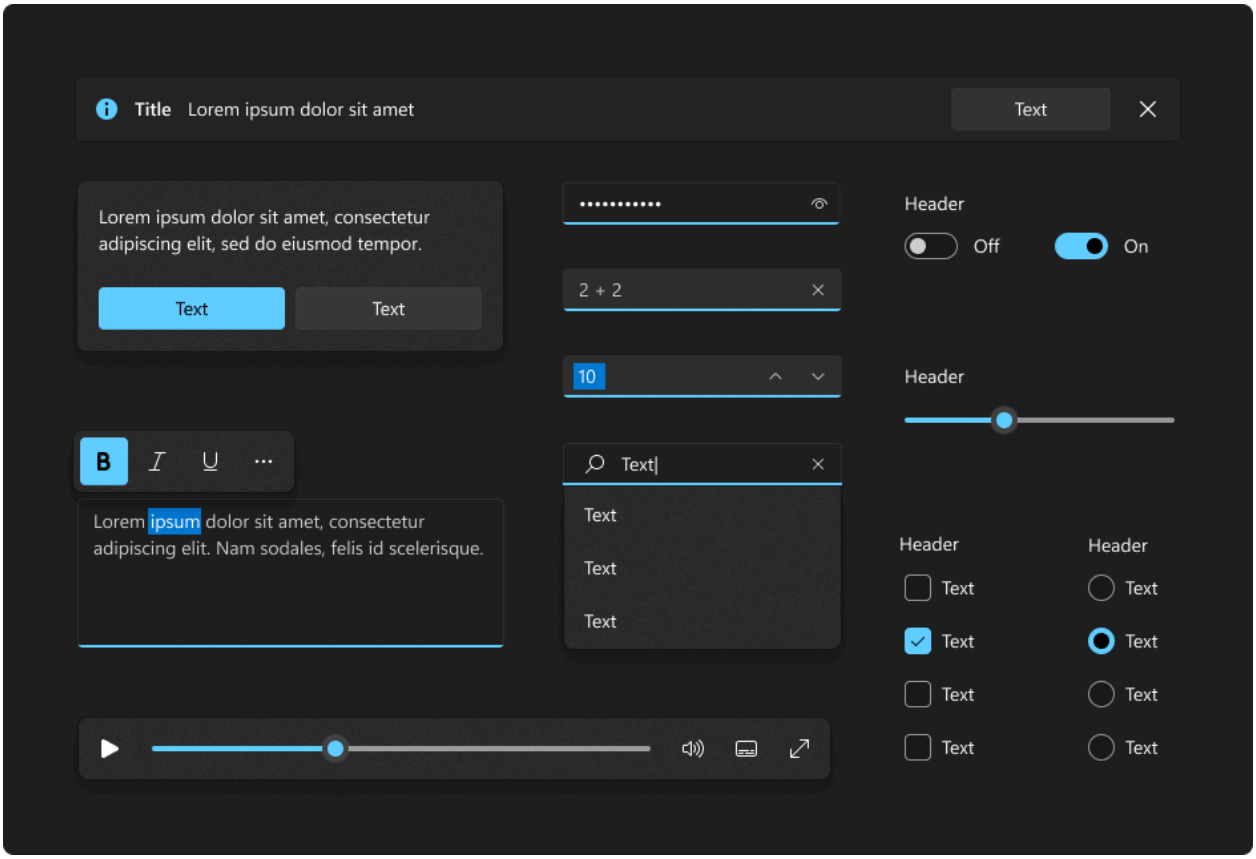


Windows supports two color modes, or themes: light and dark. Each mode consists of a set of neutral color values that are automatically adjusted to ensure optimal contrast.

In both light and dark color modes, darker colors indicate background surfaces of less importance. Important surfaces are highlighted with lighter and brighter colors. See layering & elevation for more information.

Accent color





Accent color is used to emphasize important elements in the user interface and to indicate the state of an interactive object or control. Accent color values are generated automatically and optimized for contrast in both light and dark modes. Accent colors

are used sparingly to highlight important elements and convey information about an interactive element's state.

Examples

Open the WinUI 3 Gallery app and see Color principles in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Color in Windows apps

Color provides an intuitive way of communicating information to users in your app: it can be used to indicate interactivity, give feedback to user actions, and give your interface a sense of visual continuity.

In Windows apps, colors are primarily determined by accent color and theme. In this article, we'll discuss how you can use color in your app, and how to use accent color and theme resources to make your Windows app usable in any theme context.

Color principles

Use color meaningfully. When color is used sparingly to highlight important elements, it can help create a user interface that is fluid and intuitive.

Use color to indicate interactivity. It's a good idea to choose one color to indicate elements of your application that are interactive. For example, many web pages use blue text to denote a hyperlink.

Color is personal. In Windows, users can choose an accent color and a light or dark theme, which are reflected throughout their experience. You can choose how to incorporate the user's accent color and theme into your application, personalizing their experience.

Color is cultural. Consider how the colors you use will be interpreted by people from different cultures. For example, in some cultures the color blue is associated with virtue and protection, while in others it represents mourning.

Themes

Windows apps can use a light or dark application theme. The theme affects the colors of the app's background, text, icons, and common controls.

Light theme

Dark theme

By default, your Windows app's theme is the user's theme preference from Windows Settings or the device's default theme. However, you can set the theme specifically for your Windows app.

Changing the theme

You can change themes by changing the **RequestedTheme** property in your `App.xaml` file.

XAML

```
<Application
    x:Class="App9.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:App9"
    RequestedTheme="Dark">
</Application>
```

Removing the **RequestedTheme** property means that your application will use the user's system settings.

Users can also select the high contrast theme, which uses a small palette of contrasting colors that makes the interface easier to see. In that case, the system will override your RequestedTheme.

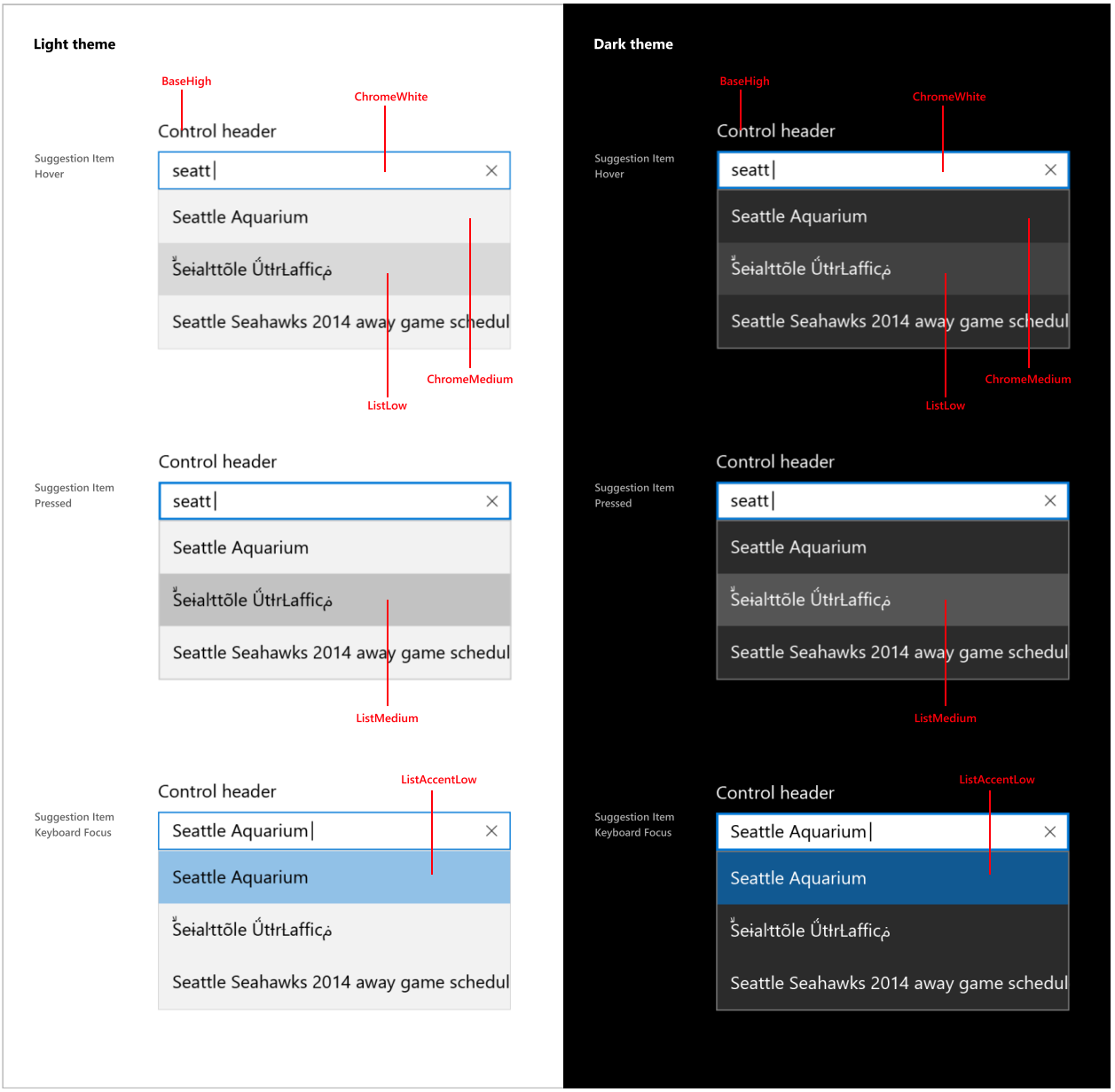
Testing themes

If you don't request a theme for your app, make sure to test your app in both light and dark themes to ensure that your app will be legible in all conditions.

Theme brushes

Common controls automatically use theme brushes to adjust contrast for light and dark themes.

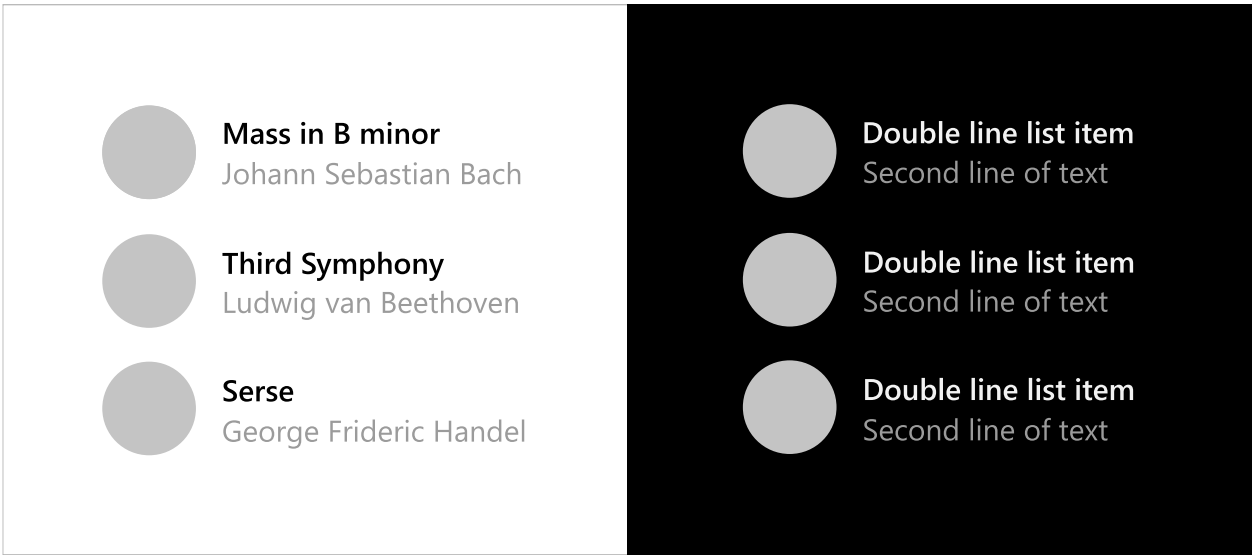
For example, here's an illustration of how the AutoSuggestBox uses theme brushes:



Using theme brushes

When creating templates for custom controls, use theme brushes rather than hard code color values. This way, your app can easily adapt to any theme.

For example, these item templates for ListView demonstrate how to use theme brushes in a custom template.



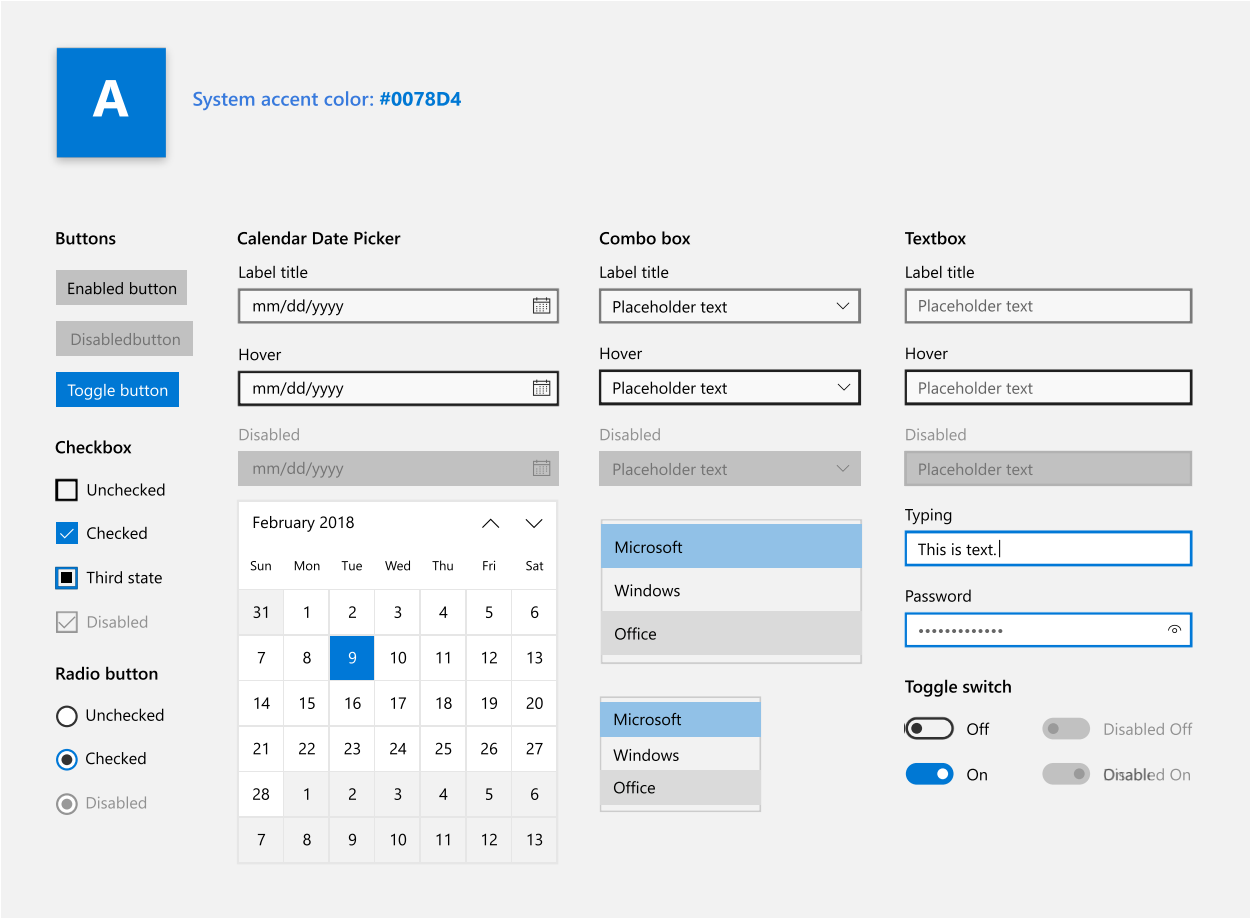
XAML

```
<ListView ItemsSource="{x:Bind ViewModel.Recordings}">
    <ListView.ItemTemplate>
        <DataTemplate x:Name="DoubleLineDataTemplate"
x:DataType="local:Recording">
            <StackPanel Orientation="Horizontal" Height="64"
AutomationProperties.Name="{x:Bind CompositionName}">
                <Ellipse Height="48" Width="48" VerticalAlignment="Center">
                    <Ellipse.Fill>
                        <ImageBrush ImageSource="Placeholder.png"/>
                    </Ellipse.Fill>
                </Ellipse>
                <StackPanel Orientation="Vertical"
VerticalAlignment="Center" Margin="12,0,0,0">
                    <TextBlock Text="{x:Bind CompositionName}" Style="
{ThemeResource BodyStrongTextBlockStyle}" Foreground="{ThemeResource
TextFillColorPrimaryBrush}" />
                    <TextBlock Text="{x:Bind ArtistName}" Style="
{ThemeResource BodyTextBlockStyle}" Foreground="{ThemeResource
TextFillColorTertiaryBrush}"/>
                </StackPanel>
            </StackPanel>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

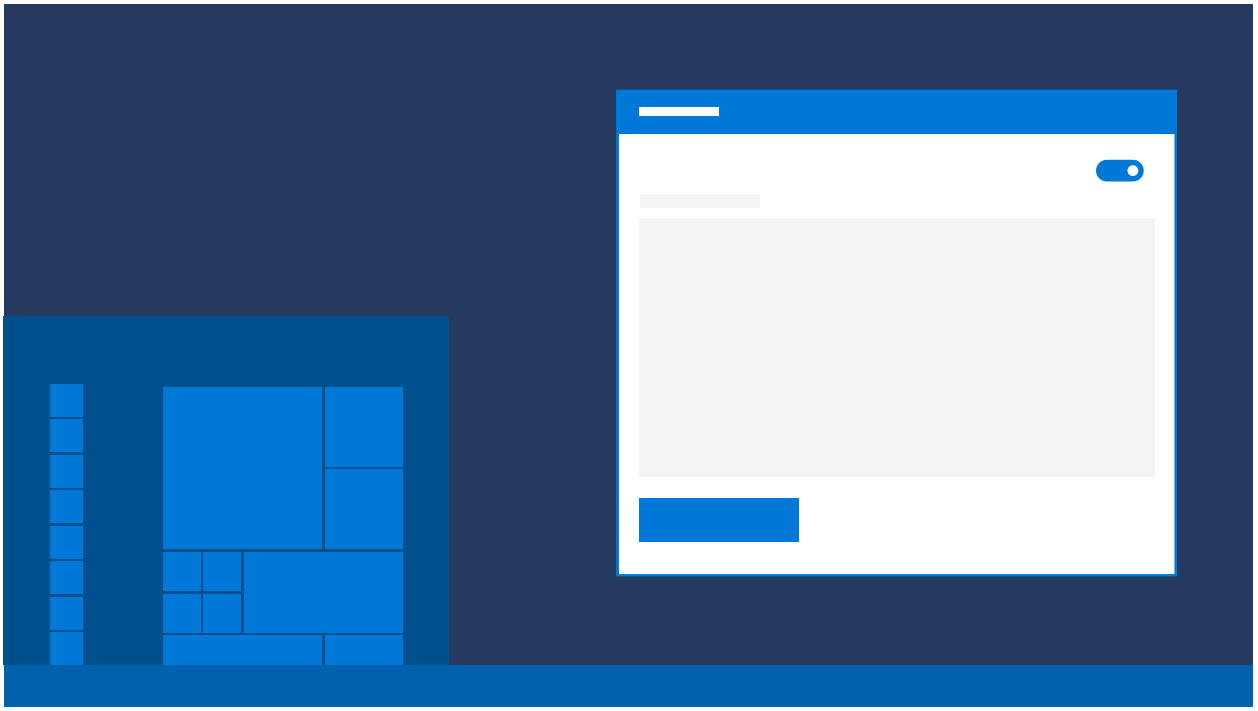
For more information about how to use theme brushes in your app, see Theme Resources.

Accent colors

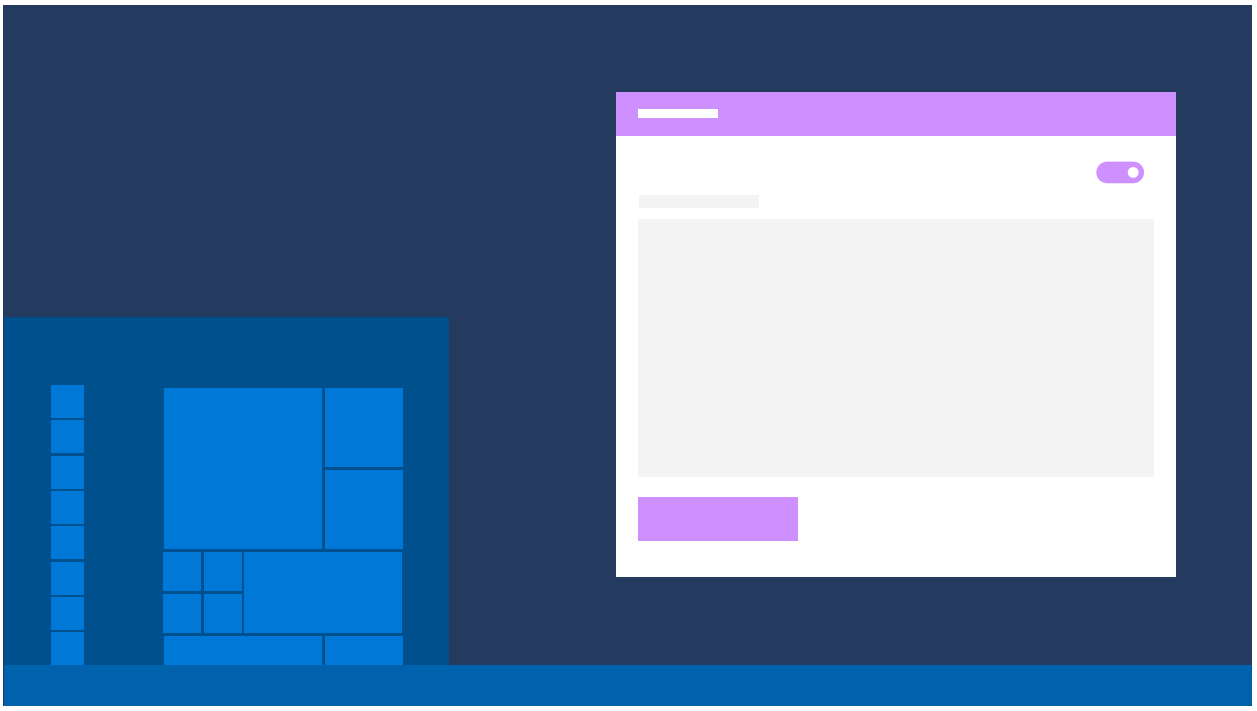
Common controls use an accent color to convey state information. By default, the accent color is the `SystemAccentColor` that users select in their Settings. However, you can also customize your app's accent color to reflect your brand.



A User-selected system accent color



C **Custom accent color**



Overriding the accent color

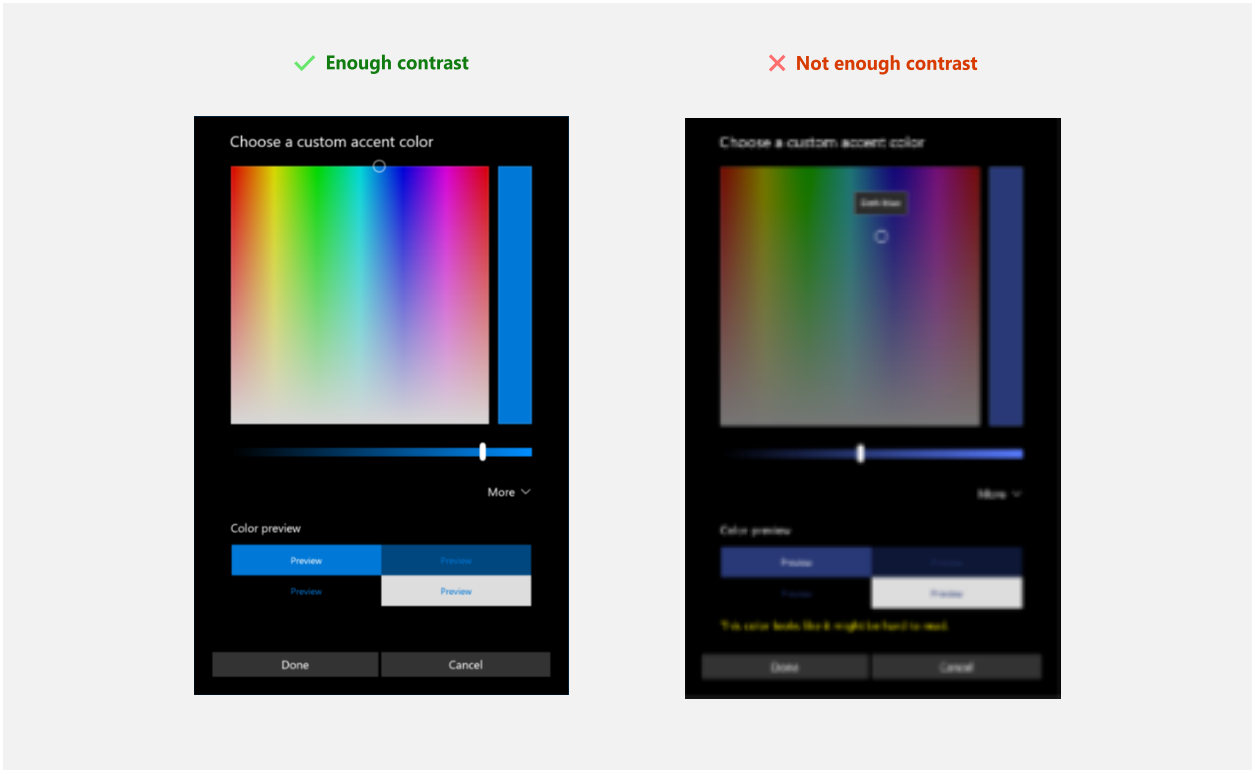
To change your app's accent color, place the following code in `app.xaml`.

XAML

```
<Application.Resources>
    <ResourceDictionary>
        <Color x:Key="SystemAccentColor">#107C10</Color>
    </ResourceDictionary>
</Application.Resources>
```

Choosing an accent color

If you select a custom accent color for your app, please make sure that text and backgrounds that use the accent color have sufficient contrast for optimal readability. To test contrast, you can use the color picker tool in Windows Settings, or you can use these online contrast tools.



Accent color palette

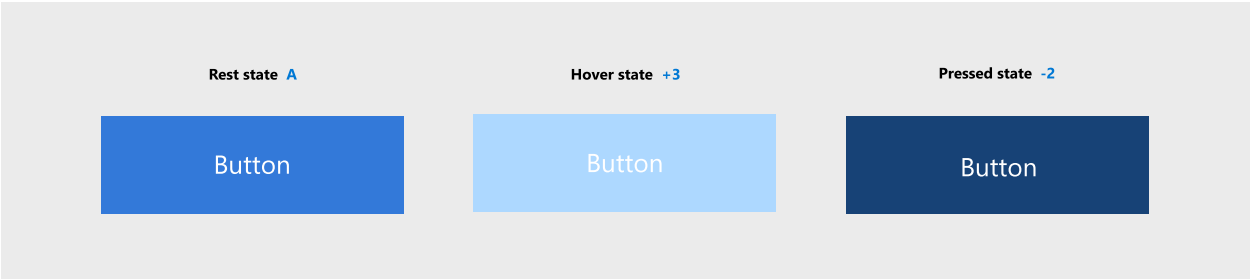
An accent color algorithm in the Windows shell generates light and dark shades of the accent color.

These shades can be accessed as theme resources:

- `SystemAccentColorLight3`
- `SystemAccentColorLight2`
- `SystemAccentColorLight1`
- `SystemAccentColorDark1`
- `SystemAccentColorDark2`
- `SystemAccentColorDark3`

You can also access the accent color palette programmatically with the **UISettings.GetColorValue** method and **UIColorType** enum.

You can use the accent color palette for color theming in your app. Below is an example of how you can use the accent color palette on a button.



XAML

```
<Page.Resources>
    <ResourceDictionary>
        <ResourceDictionary.ThemeDictionaries>
            <ResourceDictionary x:Key="Light">
                <SolidColorBrush x:Key="ButtonBackground" Color="
{ThemeResource SystemAccentColor}"/>
                <SolidColorBrush x:Key="ButtonBackgroundPointerOver" Color="
{ThemeResource SystemAccentColorLight1}"/>
                <SolidColorBrush x:Key="ButtonBackgroundPressed" Color="
{ThemeResource SystemAccentColorDark1}"/>
            </ResourceDictionary>
        </ResourceDictionary.ThemeDictionaries>
    </ResourceDictionary>
</Page.Resources>

<Button Content="Button"></Button>
```

When using colored text on a colored background, make sure there is enough contrast between text and background. By default, hyperlink or hypertext will use the accent color. If you apply variations of the accent color to the background, you should use a variation of the original accent color to optimize the contrast of colored text on a colored background.

The chart below illustrates an example of the various light/dark shades of accent color, and how colored type can be applied on a colored surface.

For more information about styling controls, see XAML styles.

Color API

There are several APIs that can be used to add color to your application. First, the **Colors** class, which implements a large list of predefined colors. These can be accessed automatically with XAML properties. In the example below, we create a button and set the background and foreground color properties to members of the **Colors** class.

XAML

```
<Button Background="MediumSlateBlue" Foreground="White">Button text</Button>
```

You can create your own colors from RGB or hex values using the **Color** struct in XAML.

XAML

```
<Color x:Key="LightBlue">#FF36C0FF</Color>
```

You can also create the same color in code by using the **FromArgb** method.

C#

```
Color LightBlue = Color.FromArgb(255,54,192,255);
```

C++/WinRT

```
Windows::UI::Color LightBlue =
Windows::UI::ColorHelper::FromArgb(255,54,192,255);
```

The letters "Argb" stands for Alpha (opacity), Red, Green, and Blue, which are the four components of a color. Each argument can range from 0 to 255. You can choose to omit the first value, which will give you a default opacity of 255, or 100% opaque.

> **Note**
>
> If you're using C++, you must create colors by using the ColorHelper class.

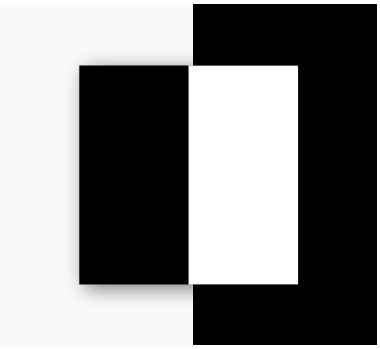
The most common use for a **Color** is as an argument for a SolidColorBrush, which can be used to paint UI elements a single solid color. These brushes are generally defined in a ResourceDictionary, so they can be reused for multiple elements.

XAML

```
<ResourceDictionary>
    <SolidColorBrush x:Key="ButtonBackgroundBrush" Color="#FFFF4F67"/>
    <SolidColorBrush x:Key="ButtonForegroundBrush" Color="White"/>
</ResourceDictionary>
```

For more information on how to use brushes, see XAML brushes.

Usability



Contrast

Make sure that elements and images have sufficient contrast to differentiate between them, regardless of the accent color or theme.

When considering what colors to use in your application, accessibility should be a primary concern. Use the guidance below to make sure your application is accessible to as many users as possible.

Lighting

Be aware that variation in ambient lighting can affect the usability of your app. For example, a page with a black background might unreadable outside due to screen glare, while a page with a white background might be painful to look at in a dark room.

Colorblindness

Be aware of how colorblindness could affect the usability of your application. For example, a user with red-green colorblindness will have difficulty distinguishing red and green elements from each other. About **8 percent of men** and **0.5 percent of women** are red-green colorblind, so avoid using these color combinations as the sole differentiator between application elements.

Related

- XAML Styles
- XAML Theme Resources

- WinUI 3 Gallery - Colors

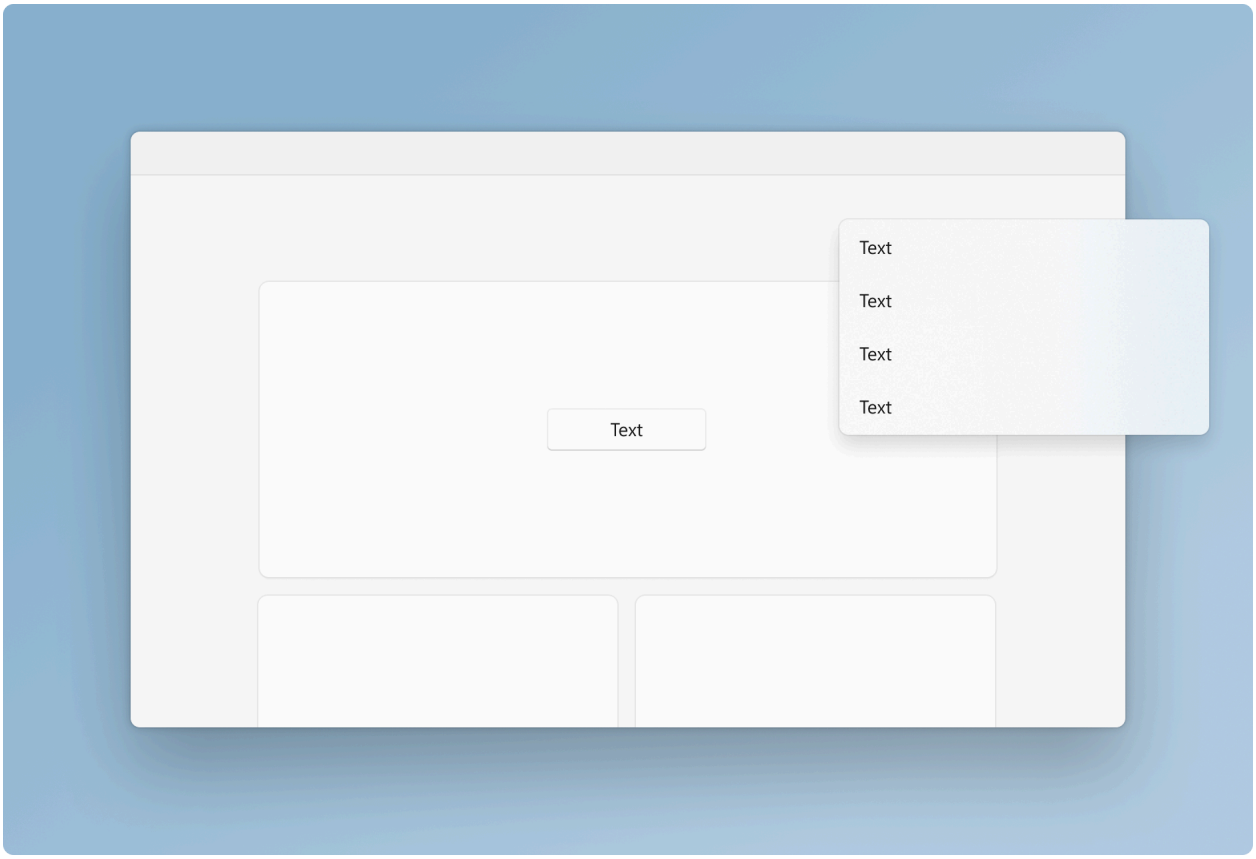
Elevation in Windows

Article • 11/05/2024

Windows 11 uses layering and elevation as its foundation for app hierarchy. Hierarchy communicates important information about how to navigate within an app while keeping the user's attention focused on the most important content. Layering and elevation are powerful visual cues that modernize experiences and help them feel coherent within Windows.

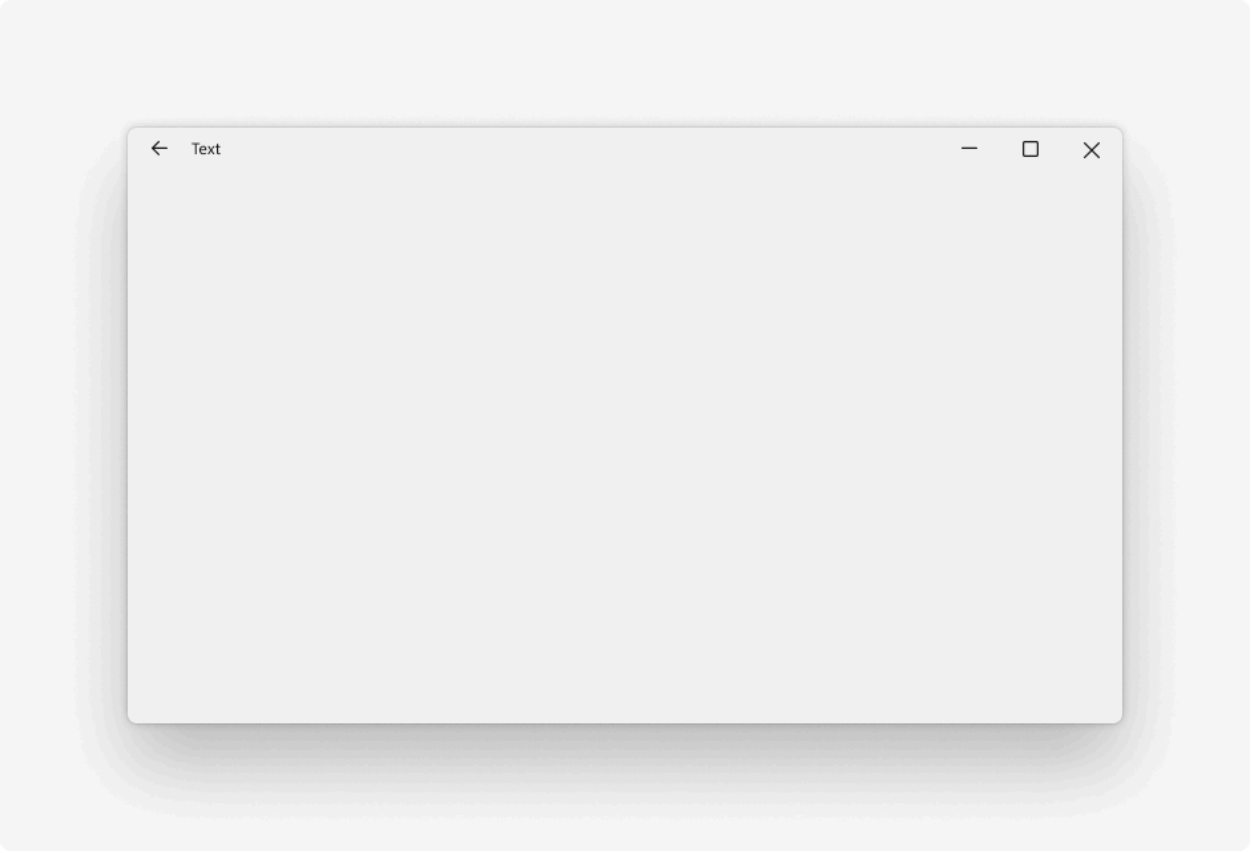
> **Tip**
>
> This article describes how the Fluent Design language is applied to Windows apps. For more information, see Fluent Design - Elevation.

Elevation

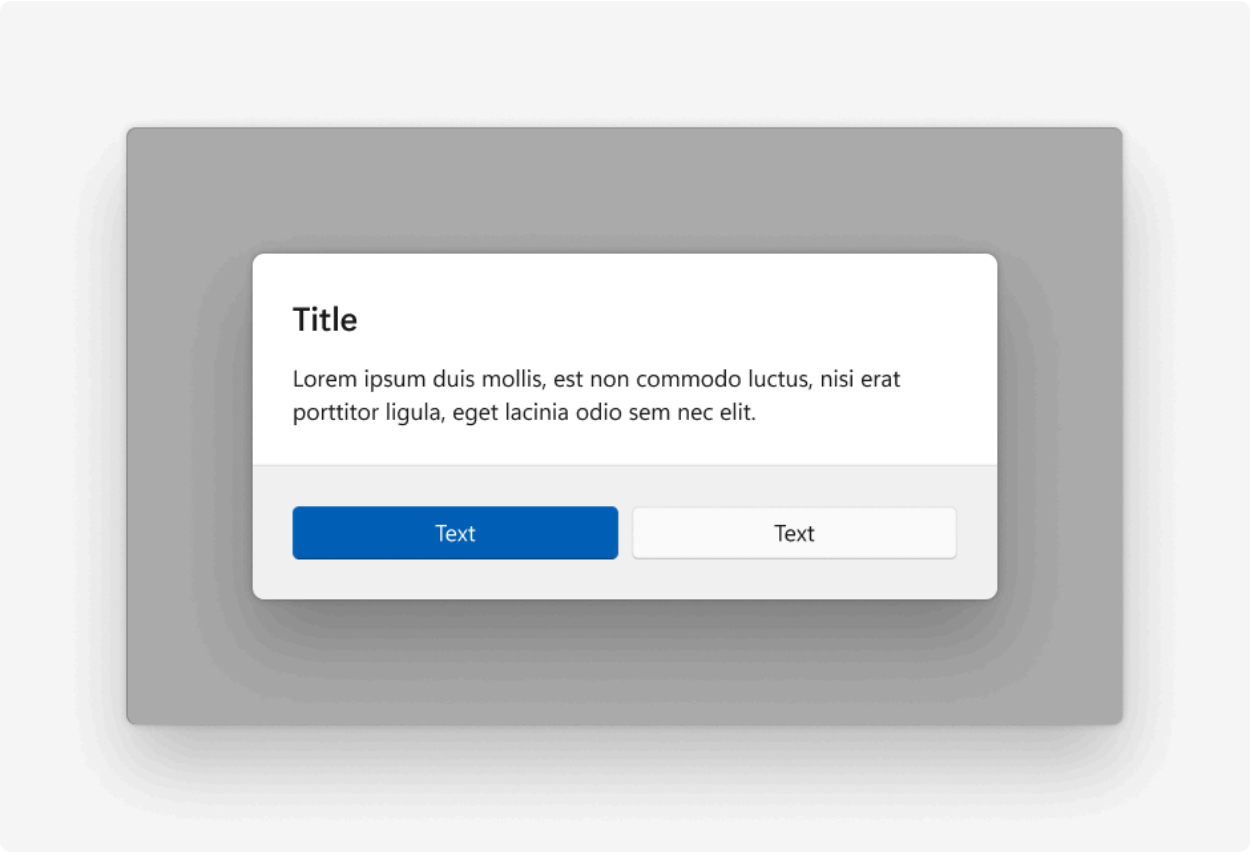


Elevation is the depth component of the spatial relationship one surface has to another with respect to their position on the desktop. When two or more objects occupy the same location on the screen, only the object with the highest elevation will be rendered at that location.

Shadows and contour (outlines) are used on controls and surfaces to subtly communicate an object's elevation, and to help draw focus where needed within an experience. Windows 11 uses the following values to express elevation with shadow and contour.



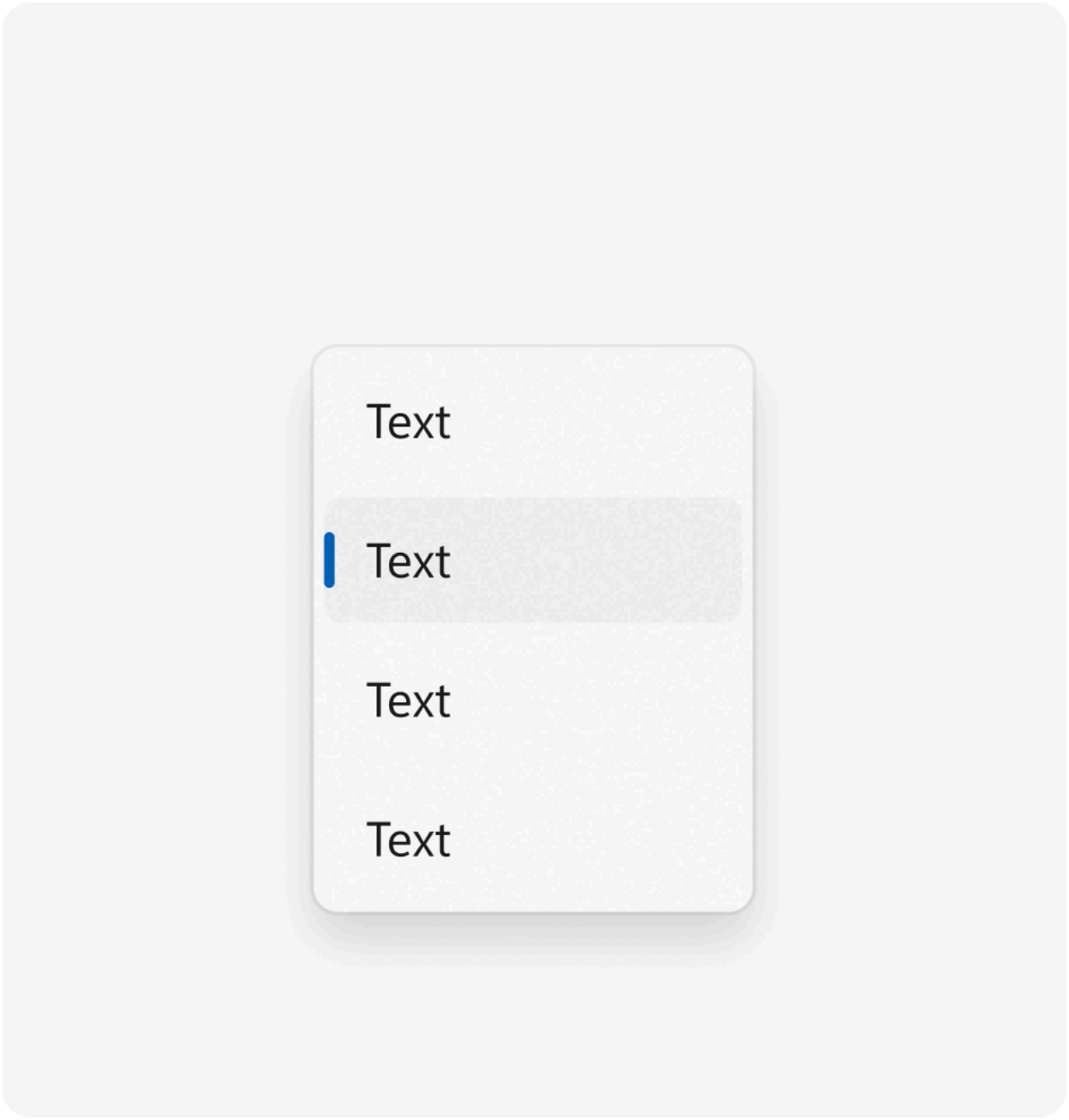
Window
Elevation value: 128
Stroke width: 1



Dialog

Elevation value: 128

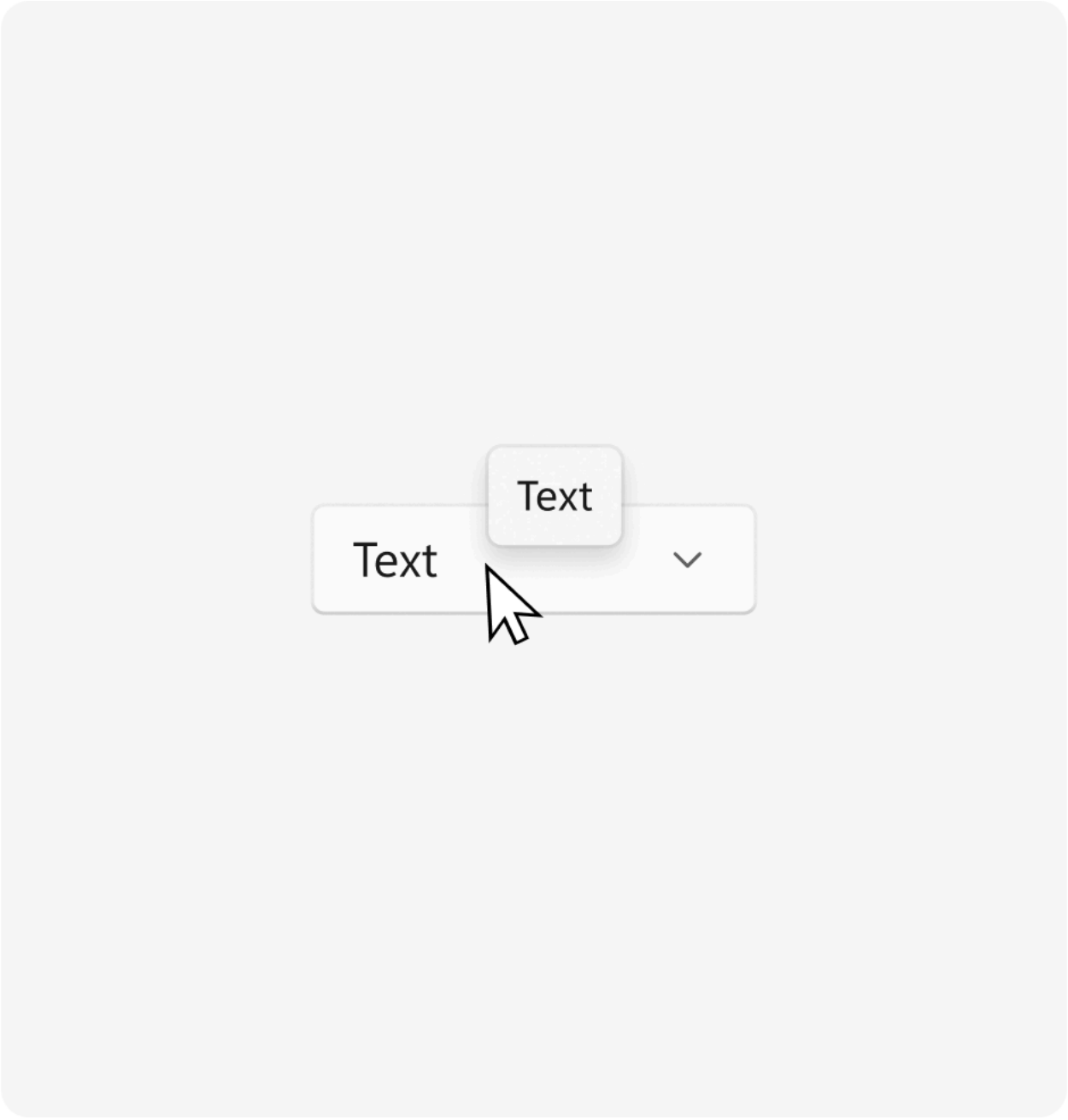
Stroke width: 1



Flyout

Elevation value: 32

Stroke width: 1



Tooltip

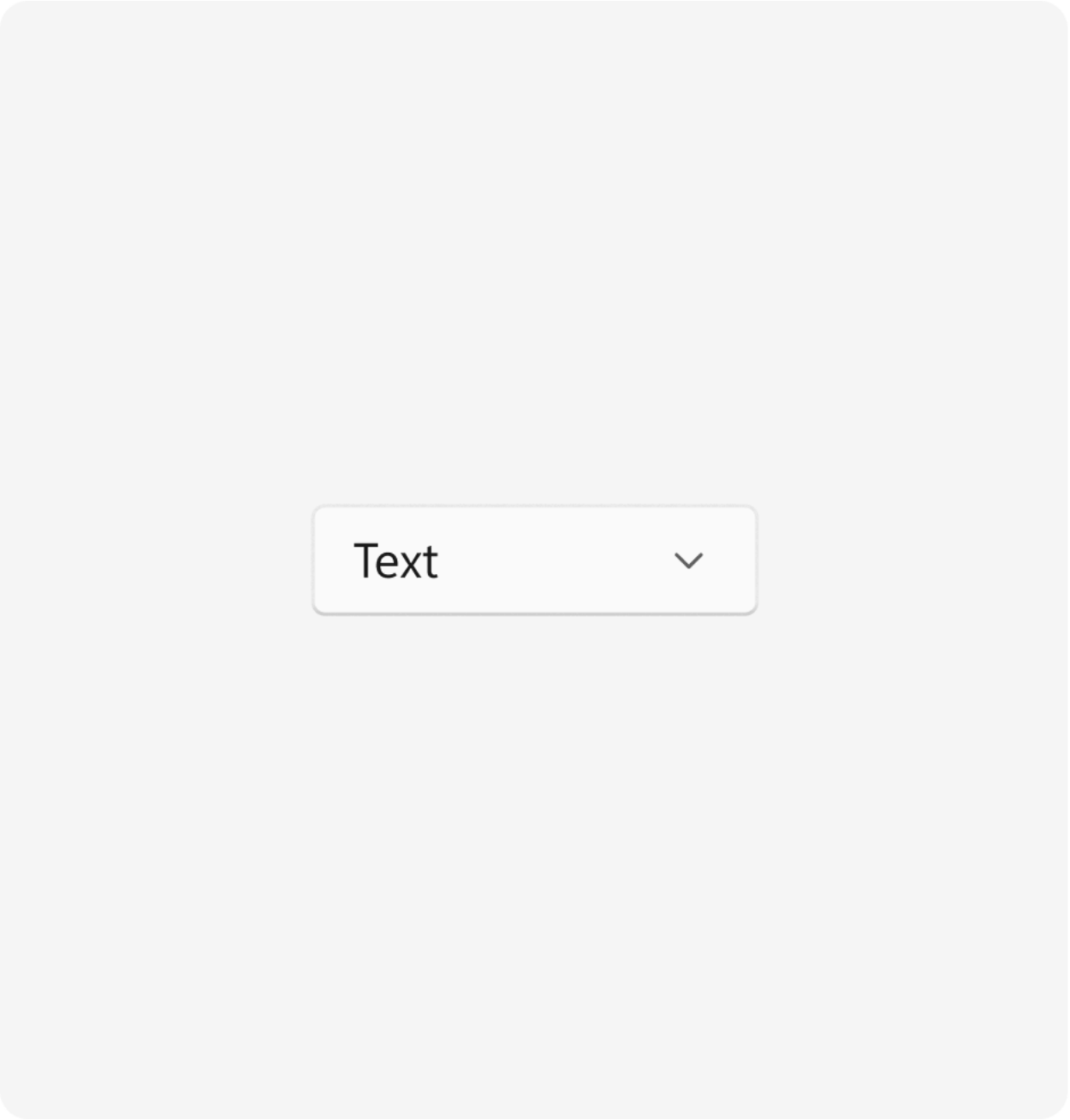
Elevation value: 16

Stroke width: 1

Card

Elevation value: 8

Stroke width: 1



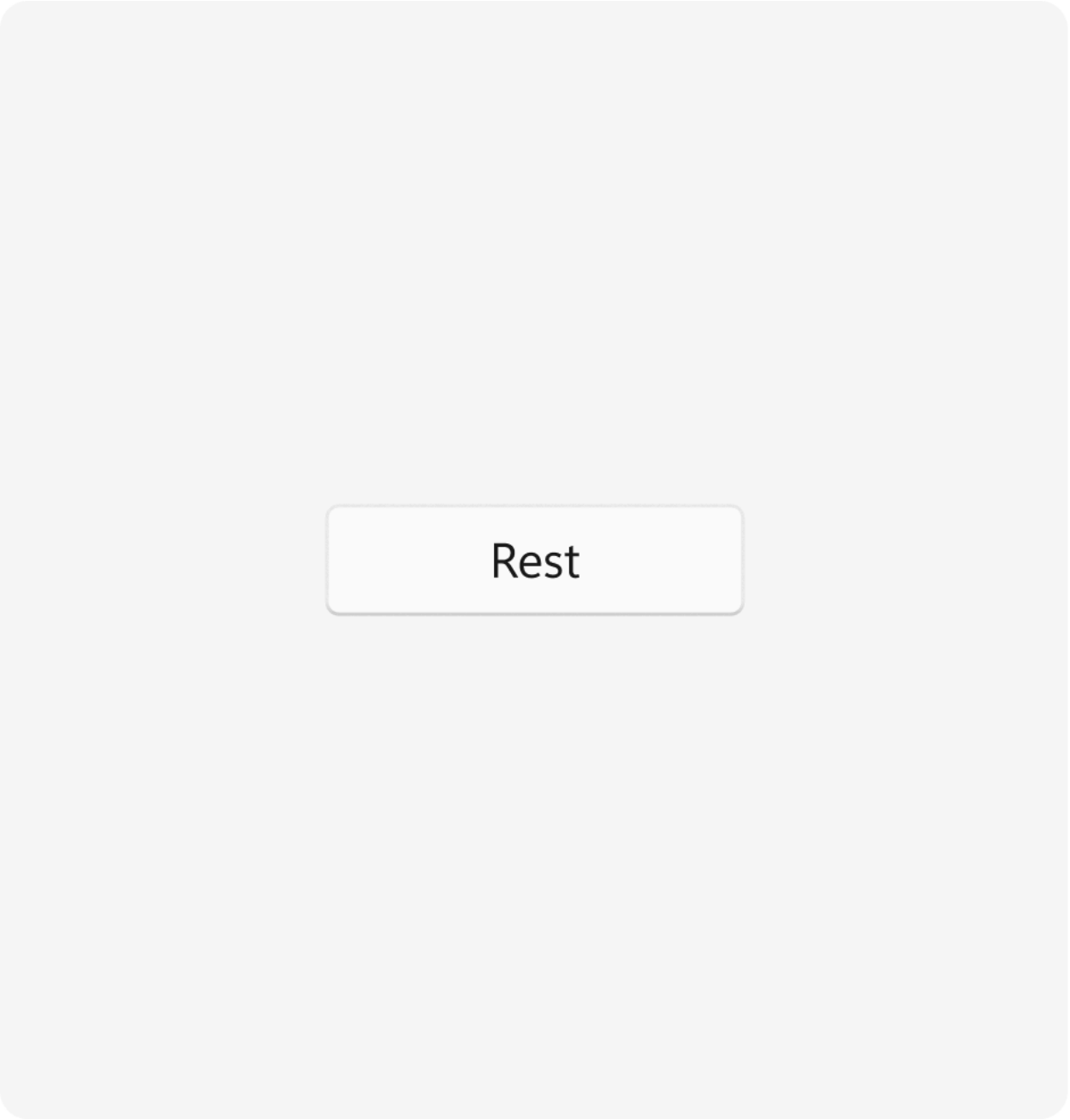
Control

Elevation value: 2

Stroke width: 1

Layer
Elevation value: 1
Stroke width: 1

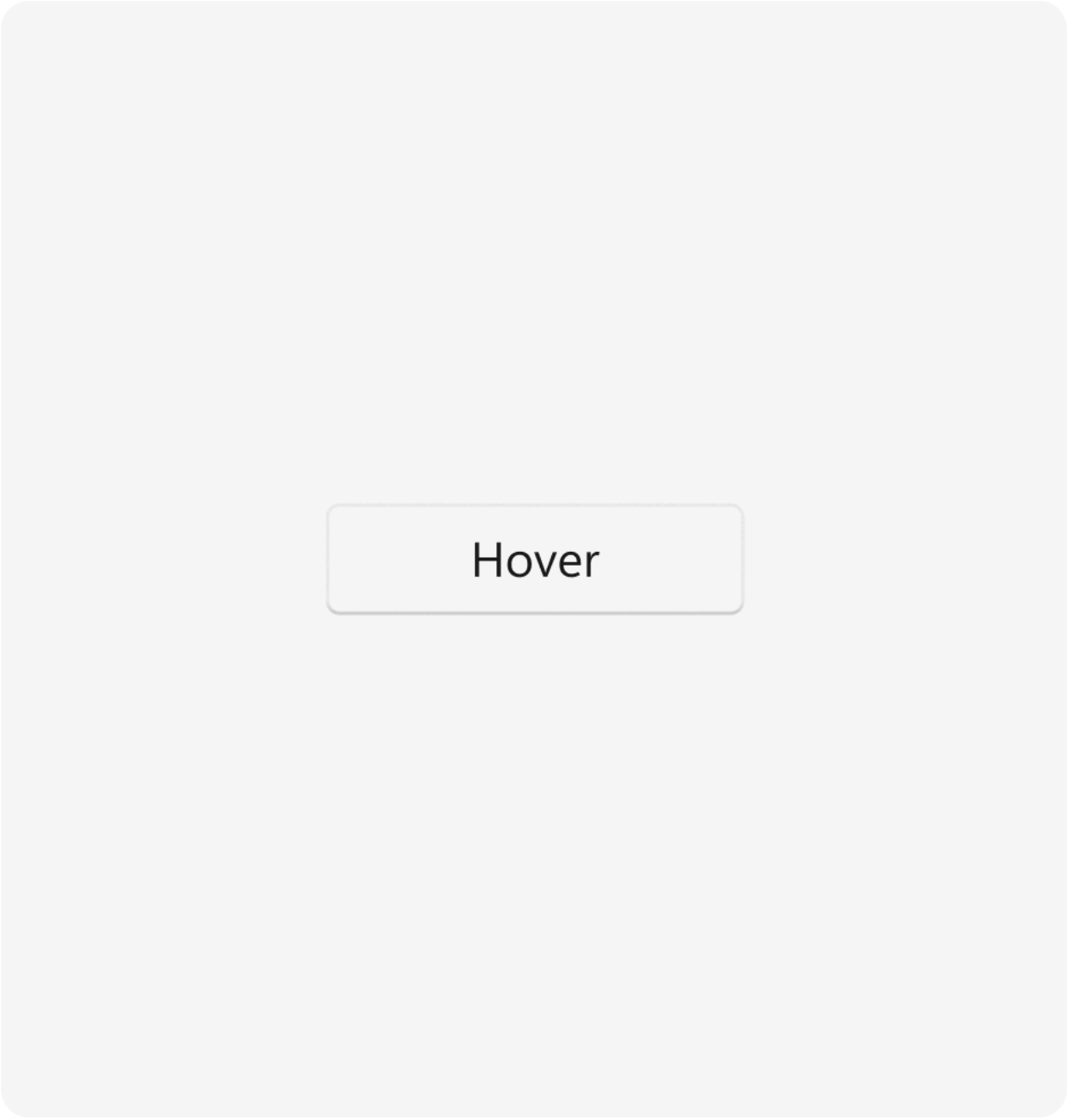
Controls in Windows 11 vary their elevation and contour to indicate state. The intensity of the rendered shadow changes depending on the theme at parity of value.



Rest

Elevation value: 2

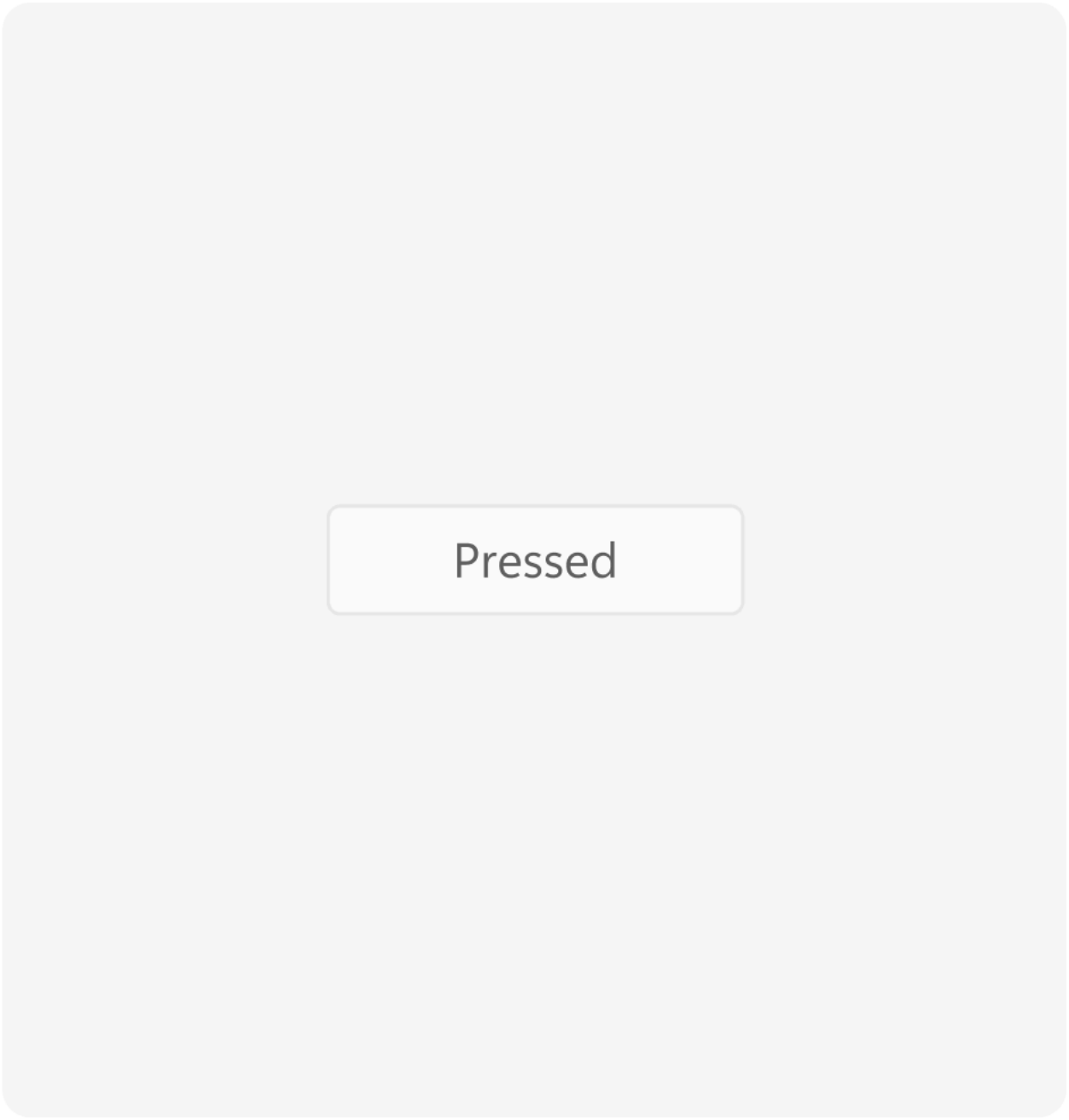
Stroke width: 1



Hover

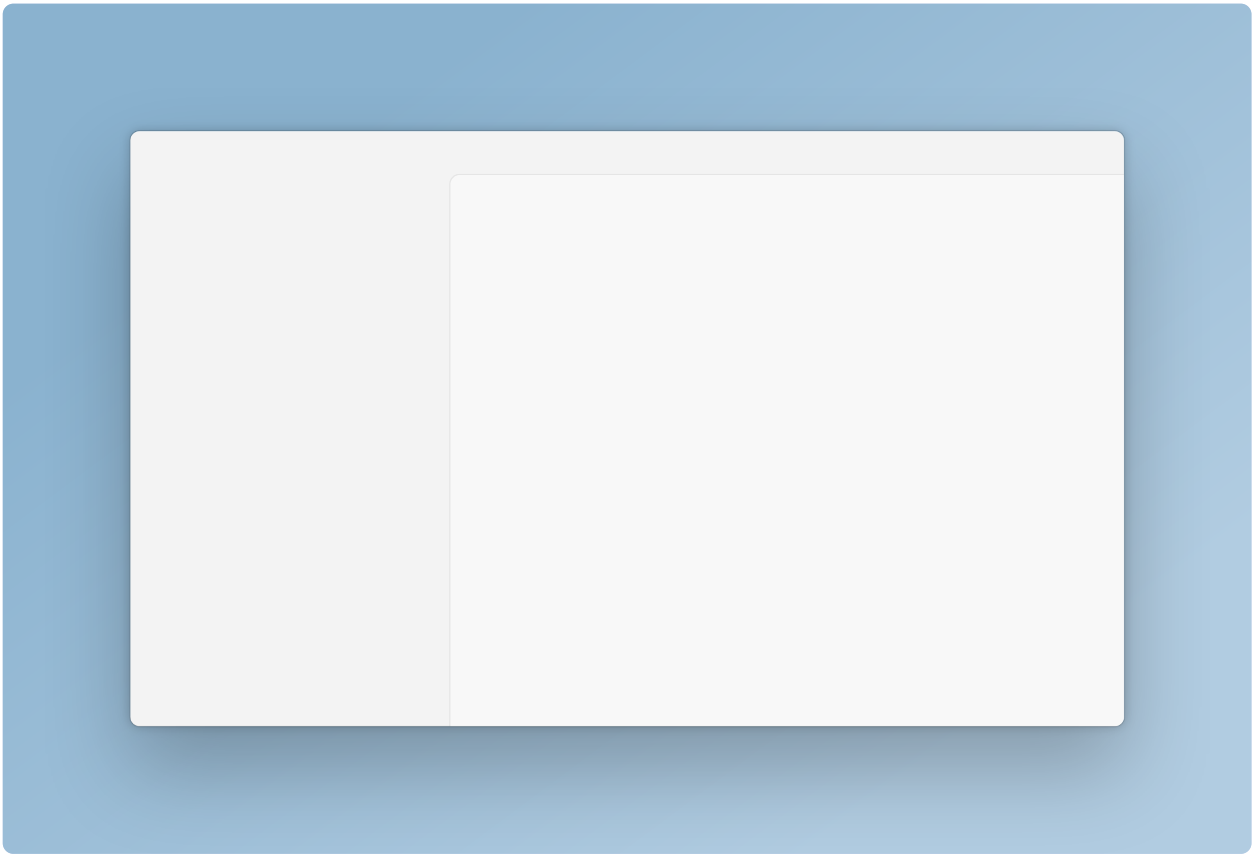
Elevation value: 2

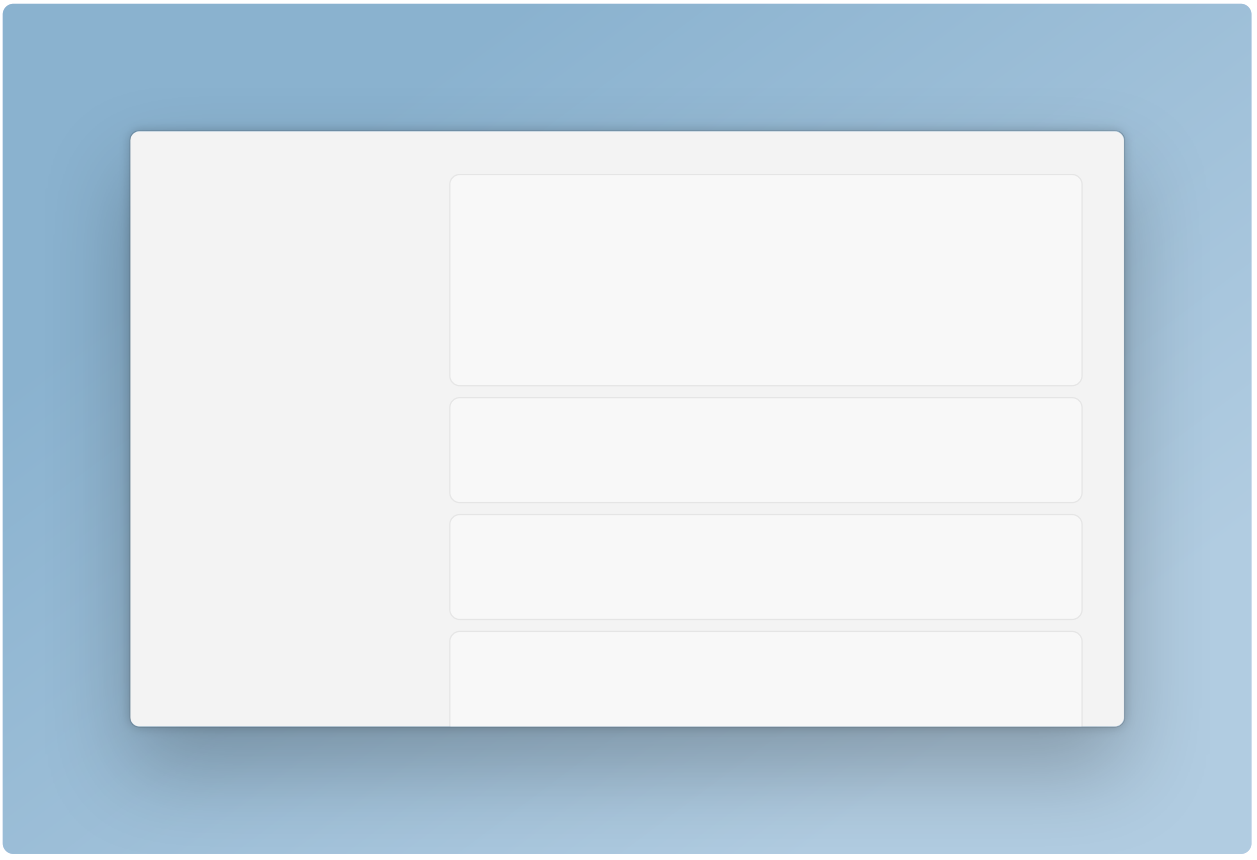
Stroke width: 1



Pressed
Elevation value: 1
Stroke width: 1

Layering





Layering is the concept of overlapping one surface with another, creating two or more visually distinguished areas within the same application.

Note

Layering in Windows is tightly coupled with the use of materials. Please reference the **materials section** for specific guidance on how those are applied.

Windows 11 uses a two-layer system for applications. These two layers create hierarchy and provide clarity, keeping users focused on what's most important.

- The **base** layer is an app's foundation. It is the bottommost layer of every app, and contains controls related to app menus, commands, and navigation.
- The **content** layer focuses the user on the app's central experience. The content layer may be on contiguous element, or separated into cards that segment content.

Z-depth and shadow

Article • 09/26/2024

Creating a visual hierarchy of elements in your UI makes the UI easy to scan and conveys what is important to focus on. Elevation, the act of bringing select elements of your UI forward, is often used to achieve such a hierarchy in software. This article discusses how to create elevation in a Windows app by using z-depth and shadow.

Z-depth is a term used amongst 3D app creators to denote the distance between two surfaces along the z-axis. It illustrates how close an object is to the viewer. Think of it as a similar concept to x/y coordinates, but in the z direction.

Windows apps use shadows to express depth and add visual hierarchy. To achieve this, the z-axis provides an easy coding path. However, the shadows are emulated; they are not displayed in the true 3D sense. This is so that we can achieve the feeling of depth without sacrificing the performance of your app's UI.

Why use z-depth?

In the physical world, we tend to focus on objects that are closer to us. We can apply this spatial instinct to digital UI as well. For example, if you bring an element closer to the user, then the user will instinctively focus on the element. By moving UI elements closer on the z-axis, you can establish visual hierarchy between objects, helping users complete tasks naturally and efficiently in your app.

What is shadow?

Shadow is one way a user perceives elevation. Light above an elevated object creates a shadow on the surface below. The higher the object, the larger and softer the shadow becomes. Elevated objects in your UI don't need to have shadows, but they help create the appearance of elevation.

In Windows apps, shadows should be used in a purposeful rather than aesthetic manner. Using too many shadows will decrease or eliminate the ability of the shadow to focus the user.

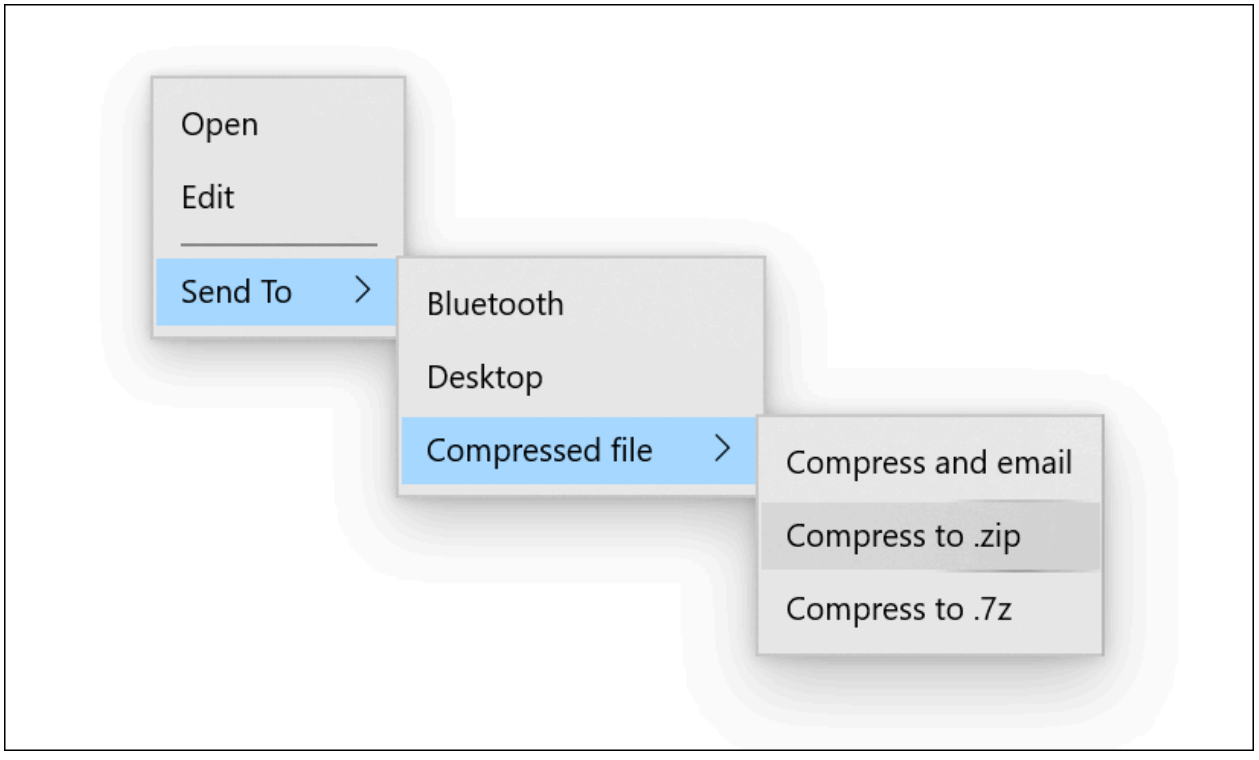
If you use standard controls, shadows are already incorporated into your UI. However, you can manually include shadows in your UI by using either the ThemeShadow or the DropShadow APIs.

ThemeShadow

The ThemeShadow type can be applied to any XAML element to draw shadows appropriately based on x, y, z coordinates.

- It applies shadows to elements based on z-depth value, emulating depth.
- It keeps shadows consistent throughout and across applications thanks to built in shadow aesthetics.

Here is how ThemeShadow has been implemented on a MenuFlyout. MenuFlyout has a built in shadow with a depth of 32px applied to the main menu and all nested menus.



ThemeShadow in common controls

The following common controls will automatically use ThemeShadow to cast shadows from 32px depth unless otherwise specified:

- Context menu, Command bar, Command bar flyout, MenuBar
- Dialogs and flyouts (Dialog at 128px)
- NavigationView
- ComboBox, DropDownButton, SplitButton, ToggleSplitButton
- TeachingTip
- AutoSuggestBox
- Calendar/Date/Time pickers
- Tooltip (16px)
- Number Box
- TabView
- Media transport control, InkToolbar
- BreadcrumbBar
- Connected animation

> **Note**
>
> ThemeShadow was introduced in Windows 10 version 1903 (SDK 18362). It is updated in Windows 11 to use ninegrid shadow instead of projected shadow for better performance.

ThemeShadow in Popups

It is often the case that your app's UI uses a popup for scenarios where you need user's attention and quick action. These are great examples when shadow should be used to help create hierarchy in your app's UI.

ThemeShadow automatically casts shadows when applied to any XAML element in a Popup. It will cast shadows on the app background content behind it and any other open Popups below it.

To use ThemeShadow with Popups, use the `Shadow` property to apply a ThemeShadow to a XAML element. Then, elevate the element from other elements behind it, for example by using the z component of the `Translation` property. For most Popup UI, the recommended default elevation relative to the app background content is 32 effective pixels.

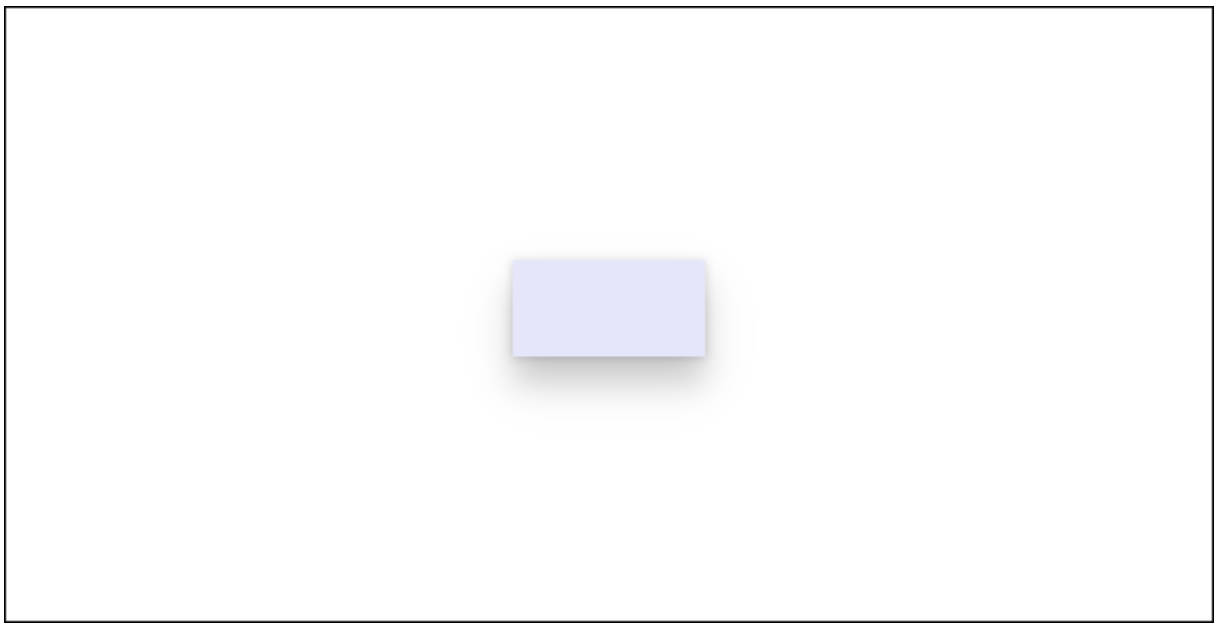
This example shows a Rectangle in a Popup casting a shadow onto the app background content and any other Popups behind it:

XAML

```
<Popup>
    <Rectangle x:Name="PopupRectangle" Fill="Lavender" Height="48"
Width="96">
        <Rectangle.Shadow>
            <ThemeShadow />
        </Rectangle.Shadow>
    </Rectangle>
</Popup>
```

C#

```
// Elevate the rectangle by 32px
PopupRectangle.Translation += new Vector3(0, 0, 32);
```



Disabling default ThemeShadow on custom Flyout controls

Controls based on Flyout, DatePickerFlyout, MenuFlyout or TimePickerFlyout automatically use ThemeShadow to cast a shadow.

If the default shadow doesn't look correct on your control's content then you can disable it by setting the IsDefaultShadowEnabled property to `false` on the associated FlyoutPresenter:

XAML

```
<Flyout>
    <Flyout.FlyoutPresenterStyle>
        <Style TargetType="FlyoutPresenter">
            <Setter Property="IsDefaultShadowEnabled" Value="False" />
        </Style>
    </Flyout.FlyoutPresenterStyle>
</Flyout>
```

ThemeShadow in other elements

> **Note**
>
> Starting with Windows 11, if the app targets the Windows SDK version 22000 or later, the `Receivers` collection is ignored. However there will be no errors and the shadow continues to function.

In general we encourage you to think carefully about your use of shadow and limit its use to cases where it introduces meaningful visual hierarchy. However, we do provide a way to cast a shadow from any UI element in case you have advanced scenarios that necessitate it.

To cast a shadow from a XAML element that isn't in a Popup, you must explicitly specify the other elements that can receive the shadow in the `ThemeShadow.Receivers` collection. Receivers cannot be an ancestor of the caster in the visual tree.

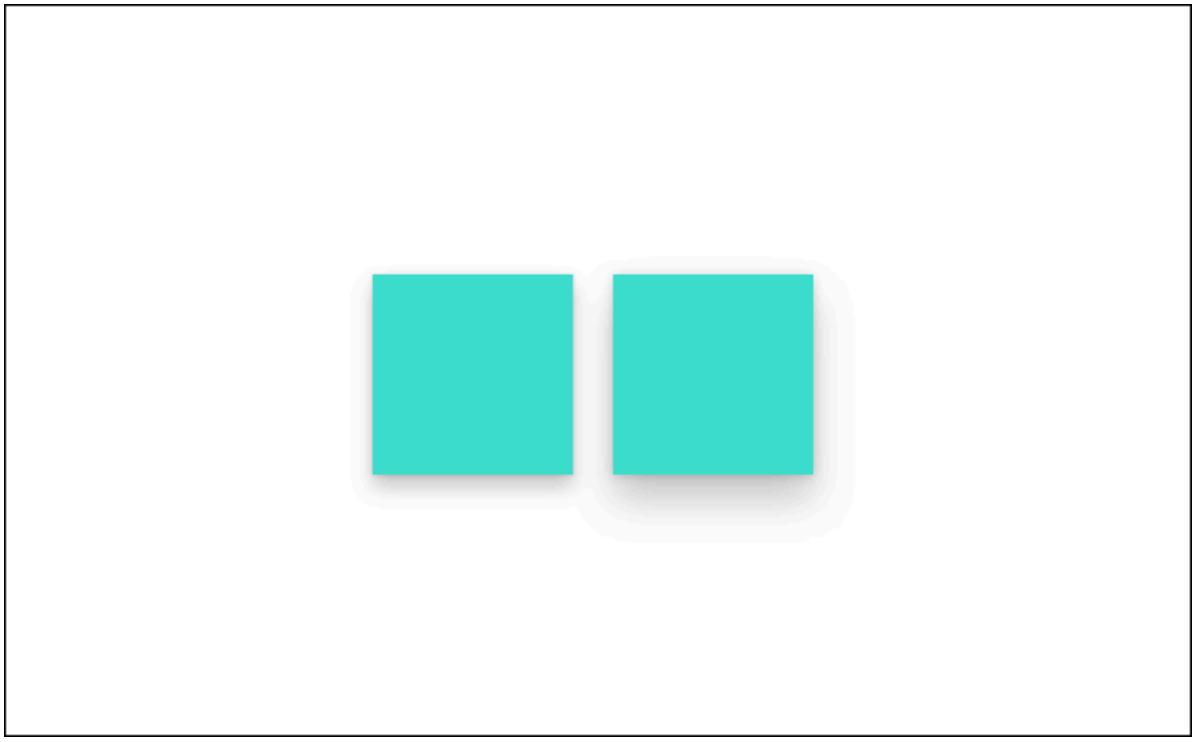
This example shows two Rectangles that cast shadows onto a Grid behind them:

XAML

```
<Grid>
    <Grid.Resources>
        <ThemeShadow x:Name="SharedShadow" />
    </Grid.Resources>

    <Grid x:Name="BackgroundGrid" Background="{ThemeResource
ApplicationPageBackgroundThemeBrush}" />

    <Rectangle x:Name="Rectangle1" Height="100" Width="100" Fill="Turquoise"
Shadow="{StaticResource SharedShadow}" />

    <Rectangle x:Name="Rectangle2" Height="100" Width="100" Fill="Turquoise"
Shadow="{StaticResource SharedShadow}" />
</Grid>
```

C#

```
/// Add BackgroundGrid as a shadow receiver and elevate the casting buttons
above it
SharedShadow.Receivers.Add(BackgroundGrid);

Rectangle1.Translation += new Vector3(0, 0, 16);
Rectangle2.Translation += new Vector3(120, 0, 32);
```



Drop shadow

DropShadow does not provide built in shadow values and you need to specify them yourself. For example implementations, see the DropShadow class.

> **Tip**
>
> Starting with Windows 11, if the app targets the Windows SDK version 22000 or later, ThemeShadow will behave like a drop shadow. If you are using DropShadow, you might consider using ThemeShadow instead.

Which shadow should I use?

Expand table

Property	ThemeShadow	DropShadow
Min SDK	SDK 18362	SDK 14393
Adaptability	Yes	No
Customization	No	Yes
Light source	None	None
Supported in 3D environments	Yes (*While it works in a 3D environment, the shadows are emulated.*)	No

- Keep in mind that the purpose of shadow is to provide meaningful hierarchy, not as a simple visual treatment.
- Generally, we recommend using ThemeShadow, which provides consistent shadow values.
- For concerns about performance, limit the number of shadows, use other visual treatment, or use DropShadow.
- If you have more advanced scenarios to achieve visual hierarchy, consider using other visual treatment (for example, color). If shadow is needed, then use DropShadow.

Iconography in Windows

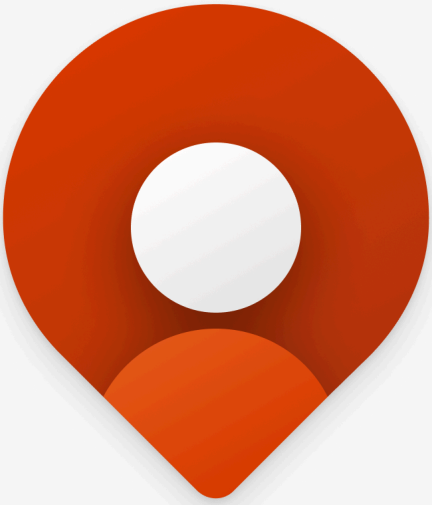
Article • 07/29/2024

Iconography is a set of visual images and symbols that help users understand and move through your app. Icons are used throughout the user interface as visual metaphors that represent a concept, action, or status.

Windows 11 uses three types of icons: application, system, and file type. This article focuses on the first two.

> **Tip**
>
> This article describes how the Fluent Design language is applied to Windows apps. For more information, see Fluent Design - Iconography.

Application icons



Application icons represent your app in the Windows shell. They're primarily used to open your application, but they also represent your app wherever it appears in the Windows shell.

App icons should represent your app's core functionality through a metaphor. For more information about designing and constructing your app's icon, see **App icons**.

System icons

Use system icons inside your app UI for items like command bars, navigation, or status indicators. Windows 11 introduces a new system icon font, Segoe Fluent Icons. This new font complements geometry in Windows 11.

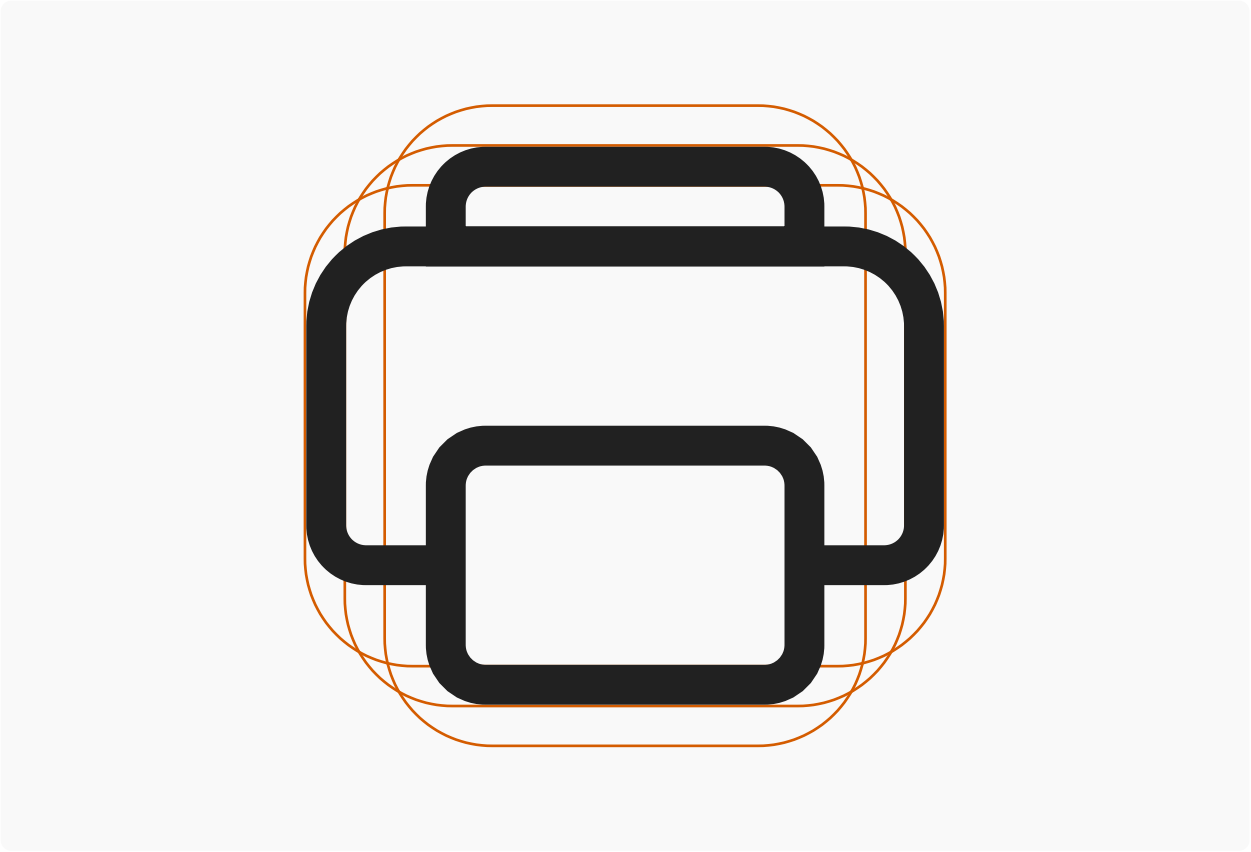
All glyphs in Segoe Fluent Icons are drawn in a monoline style. That means they're created through a single stroke of 1 epx.

Glyphs in Segoe Fluent Icons follow three aesthetic principles:

- **Minimal**: Glyphs contain only the details that are necessary to communicate the concept.
- **Harmonious**: Glyphs are based on simple and geometric forms.
- **Evolved**: Glyphs use modern metaphors that are easily understood.

For more information about using icons in your app UI, see **Icons in Windows apps**.

Icon size



Font metrics for Segoe Fluent Icons match how designers and developers are accustomed to working with SVG and bitmap icons.

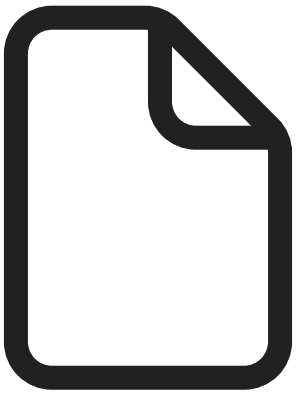
Each font glyph is designed so that the footprint of the icon area is a square em. An icon with a 16-epx font size is the equivalent of a 16x16-epx icon, to make sizing and positioning more predictable.

Modifiers

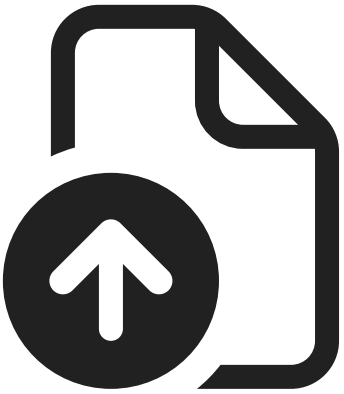
You can visually construct system icon glyphs by combining a base icon with a modifier icon.

Base icons are the main element of a visual metaphor. Base elements should occupy the entire icon footprint.

Modifier icons modify the meaning of the base icon. Modifier elements should be placed in one of the bottom quadrants of the icon footprint.



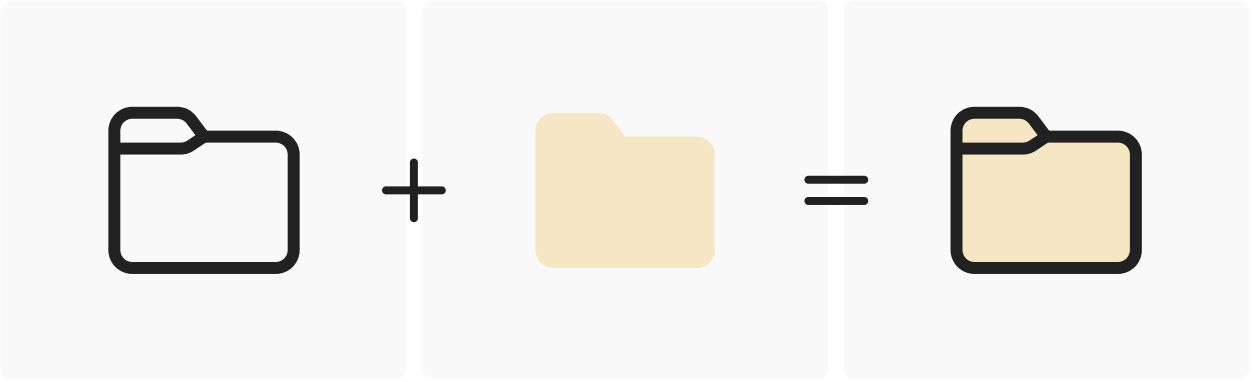
Base icon only
On its own, the paper sheet icon communicates the concept of a *file*.



Base icon + modifier icon
Adding an up arrow to the file icon changes the meaning of the icon to represent an *uploaded file*.

Layering

Icon layering is a technique that you use to overlap two glyphs. We recommend using icon layering to create a different state of the same icon (for example, an active or selected state).



Localization

Understand the cultural connotations of symbols. Although iconography doesn't require localization in most cases, certain icons might be acceptable in one culture but not in another. Validate your iconography choices with the context in which you'll use them.

Examples

Open the WinUI 3 Gallery app and see Iconography principles in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

App icons

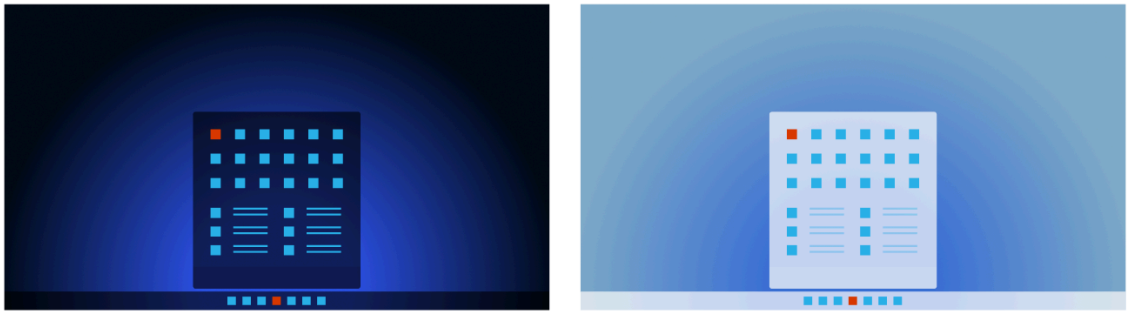
Article • 07/29/2024

App icons are the visual indicators we use to help users find and launch an application on Windows.

It's the first impression of your app and we want to help make sure it's a great one. This article covers the best practices for Windows app icon design and implementation to provide the best experience in the Windows ecosystem.

> **Tip**
>
> This article describes icons that represent your app in Windows and the Windows Store. To learn about icons that you use inside your app UI, see **Icons in Windows apps**.

Where do app icons appear?



App icons appear in a variety of places in Windows:

- Start menu
- Start menu's 'All apps' list
- Taskbar
- Splash screen
- App title bar
- Search results
- Notifications
- "Open with" list
- Task manager
- JumpLists

- Settings
- Share dialog

Principles for great app icons

Keep it simple

Create clean, straightforward, timeless designs — Icons are quick visual cues that represent your app. Simple, easy to understand designs help users recognize your app more quickly.

Keep it universal

Make it accessible, inviting, and easy to understand — Avoid complex or overly abstract forms. An intricate shape may be appropriate in some cultures but not in others. Inclusive icons are better for everyone. We focus on human unifiers – our motivations, relationships, and capabilities. We design to extend human abilities and create a sense of belonging.

Do less, better

When every detail is thoughtfully crafted, delight happens — The best apps offer beautiful design and reliable experiences. Your icon should illustrate your app in a singular element using simple forms.

Design and create your app's icon

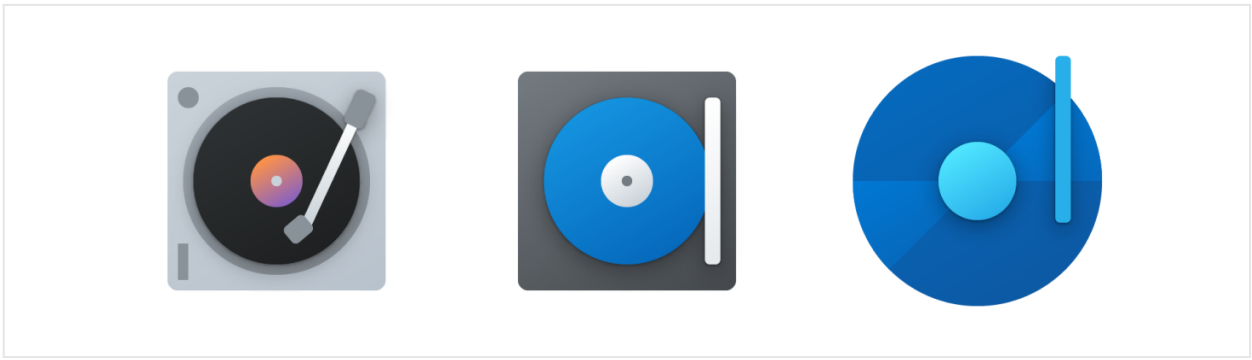
- Design guidelines for Windows app icons — Learn how to use metaphor, shape, color, and shadow to design a beautiful and functional icon for your app.
- Construct the icon for your Windows app — Create icons that look good at any size, on any screen.
- Construction guidelines for Windows 10 icons — If your app runs on Windows 10, your app's icons will be displayed in Live Tiles. Learn more about additional considerations for icons on Windows 10.

Design guidelines for Windows app icons

Article • 03/13/2023

Follow these guidelines to create a great app icon for your app that feels at home on Windows.

Design guidance: Metaphor

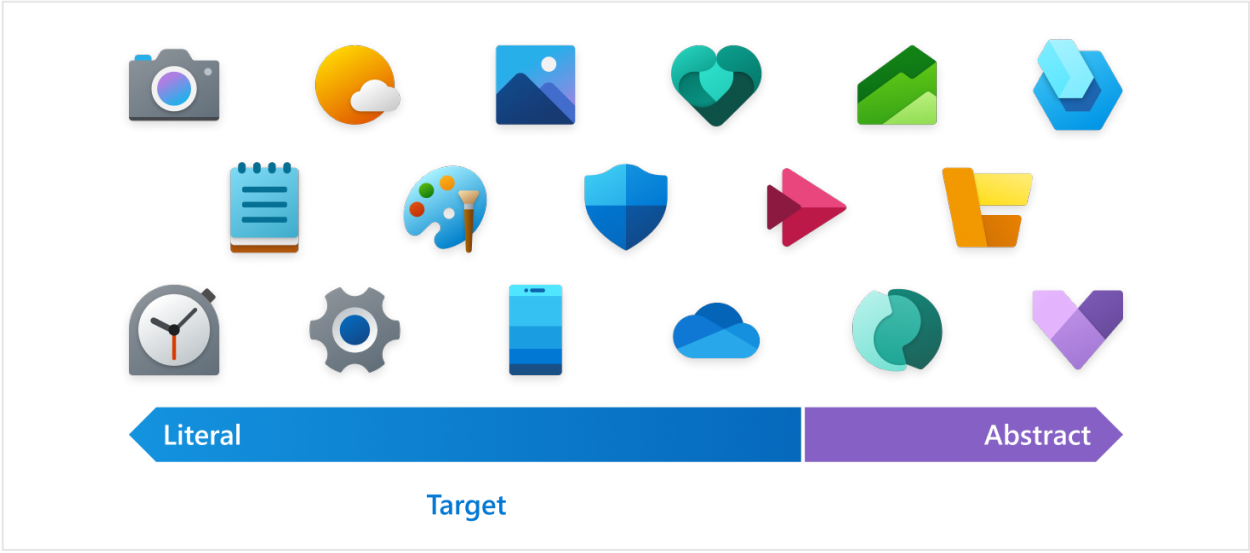


An icon should be a metaphor for its app: a visual representation of the value proposition, functions, and features of the product.

Representation

Your icon should illustrate the concept of your app in a singular element using simple forms.

When creating your icon, use clear metaphors and leverage concepts that are largely understood - such as an envelope for mail or magnifying glass for search. The key concept should be your icon's focal point; don't dilute your icon by adding decorative elements that don’t support the metaphor. To enhance communication clarity, use no more than two metaphors in a single icon. If a single metaphor can be used, that’s even better.



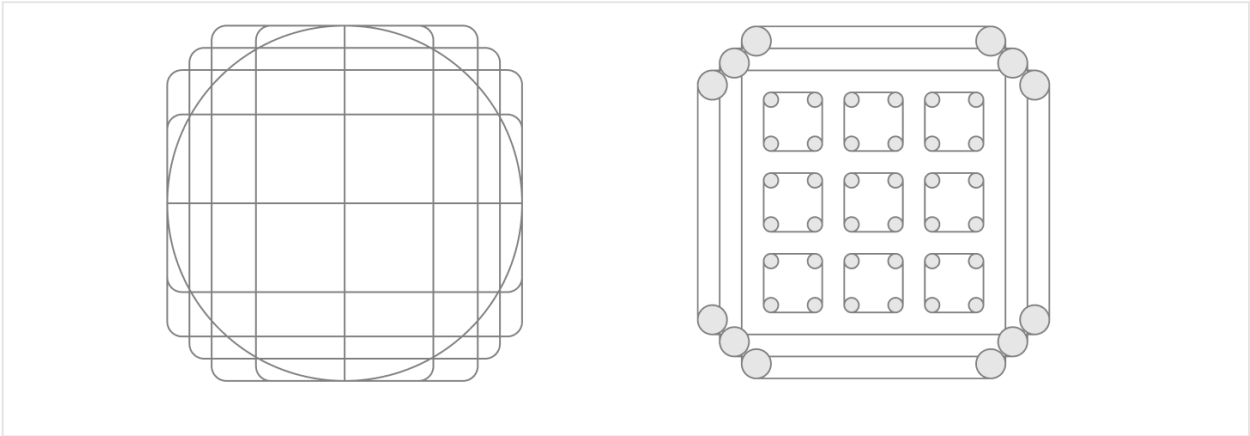
Literal metaphors are best for articulating the purpose and promise in a clear way. A good test for an effective icon is when users can tell what it represents without a label.

Only use an abstract metaphor in instances where it's impossible to find a literal, self-evident metaphor to represent the core functionality of a product.

Icons should not include typography as part of the design. Letters and words on your icon should be avoided and only used when it’s essential. The app’s name will appear in association with the icon throughout the operating system.

Design guidance: Shape

The grid and rounded corners



Microsoft aligns its icons to a 48x48 grid initially to ensure a balanced icon that takes advantage of the space available, while still maintaining a distinctive shape and silhouette. Aligning your icon's distinctive features to the grid will balance well with the other icons around it.

Approachability is a Microsoft personality principle. One way we communicate this trait is by using soft or rounded corners. Shapes used in your app's product icons should be built to align with the icon grid. The corners of these shapes should match the rounded corners in the icon grid. When rounded corners are applied to an exterior curve, use a 2px radius at 48x48. When rounded corners are applied to an interior curve, use a 1px radius instead.

Silhouette

A visually balanced silhouette allows good icon scalability and also avoids extremes of thick and thin shapes. Use the grid to design a silhouette that’s distinctive, yet legible at small sizes. Use as few shapes with as few corners as possible to distinguish your product while still feeling at home on Windows.

Detail

When adding detail, care should be taken to maintain legibility at small sizes. It is recommended to only add additional literal detail to the most prominent layer of an icon.

Design guidance: Color and gradients

Pick colors carefully and avoid relying on color alone to convey meaning. Use shape and metaphor with color to communicate. To avoid complexity when scaling an icon across a range of sizes, treatments to colors should be minimized. Color gradients, overlays of varying opacity, and tints of color should be kept to a minimum.

Gradients should be subtle for the most part. Try to limit your gradient ramps to only one or two steps in both the horizontal and vertical directions.

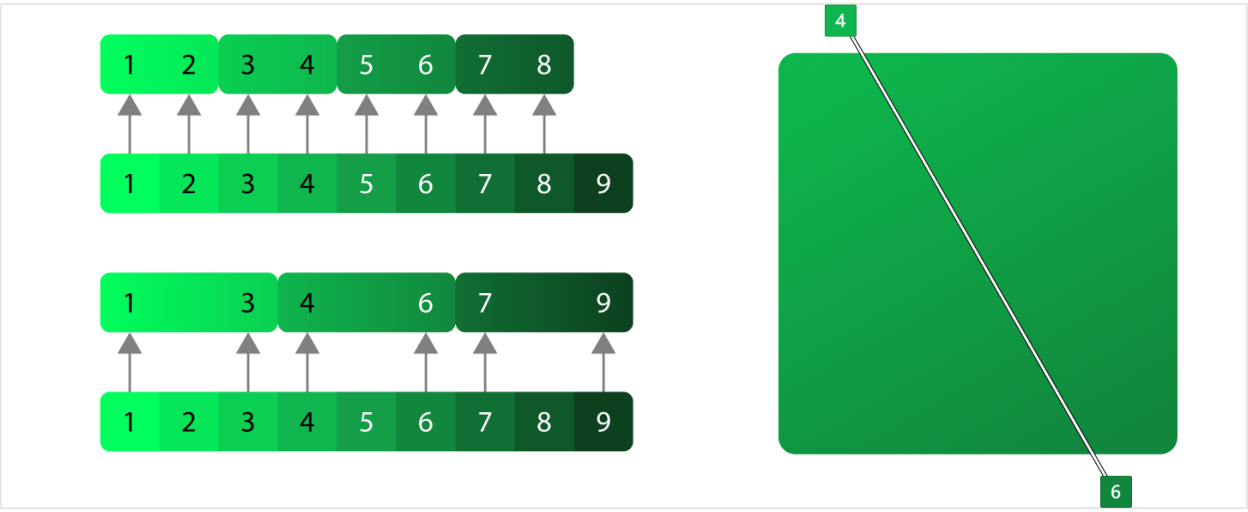
The default angle for gradients is 120 degrees. Start and end points can be adjusted accordingly. The important thing is that it’s a smooth transition. Avoid very tight transitions that would feel like reflections or dimension.

Monochrome palette

Create a monochrome palette using the following steps:

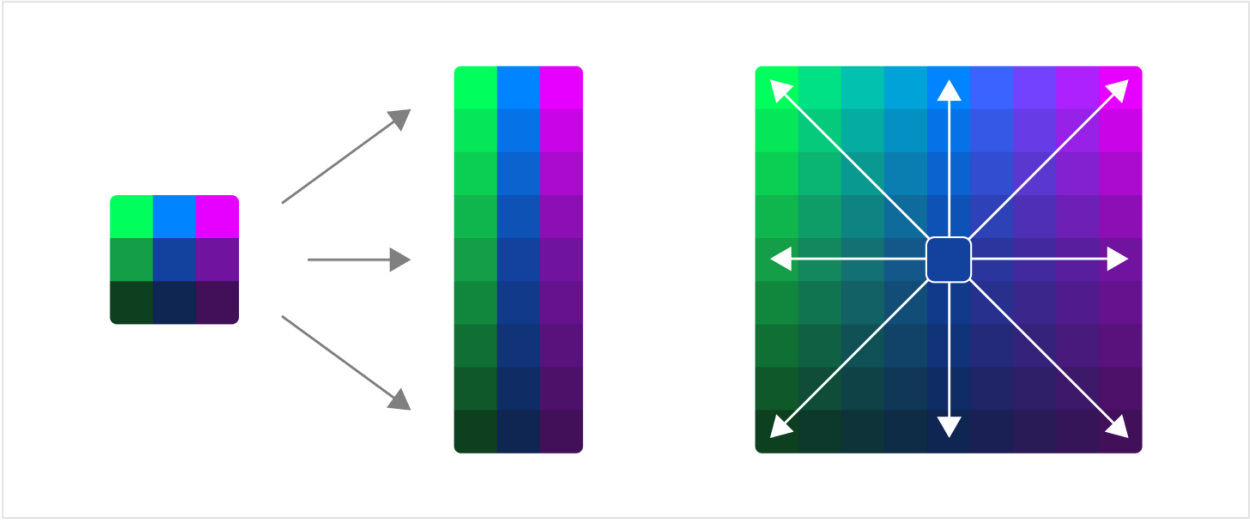
1. Create three colors from the same hue. In most cases you will have to adjust the light color to be brighter and the dark color to be less saturated, but of course you should use your best judgement.
2. Create three steps in between each base color. This will be your primary lane. Most of the icon should be comprised of these colors.
3. For a wider palette, create tints to white and shades to black using the same method as step 2. These tints and shades should be used only when you need a little more contrast.
4. The tints of the dark colors and shades of the light colors are usually useless and drab. They can be removed.

Monochrome gradients



Monochrome gradients are usually used to give a subtle hint toward an ambient light angle coming from the top left. They should not be treated as a direct light source though. The idea is to give the shapes a little movement without being too dramatic.

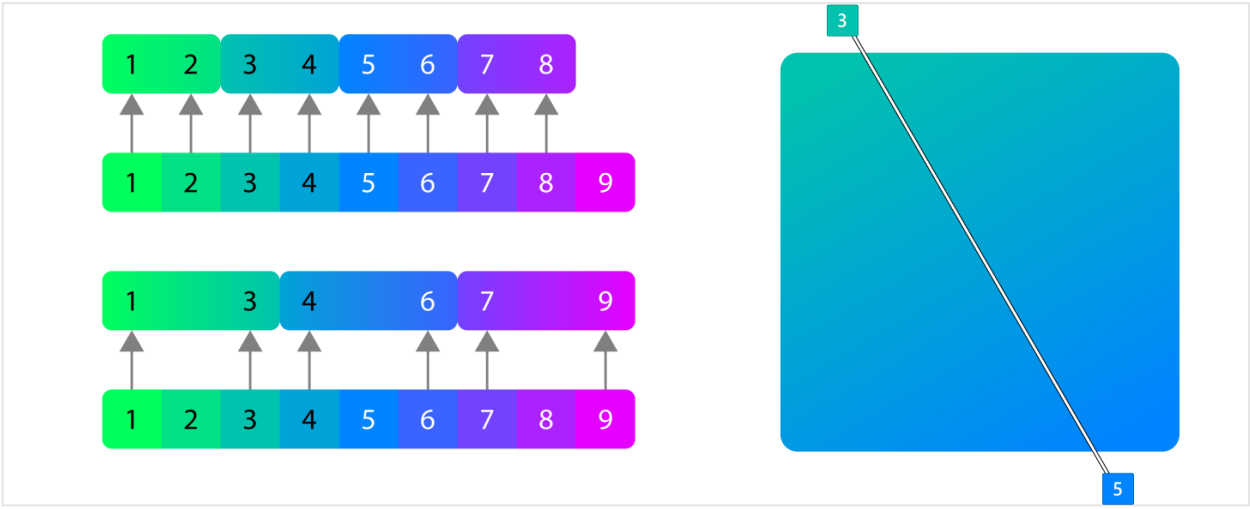
Analogous palette



Creating an analogous palette is exactly like creating a monochrome palette, but with more colors. The key to this type of palette is not to overdo it. Be thoughtful with your color transitions.

1. Create three color sets instead of one.
2. Make vertical ramps out of all three color sets.
3. Instead of creating tints and shades using white and black, use your second and third colors instead.

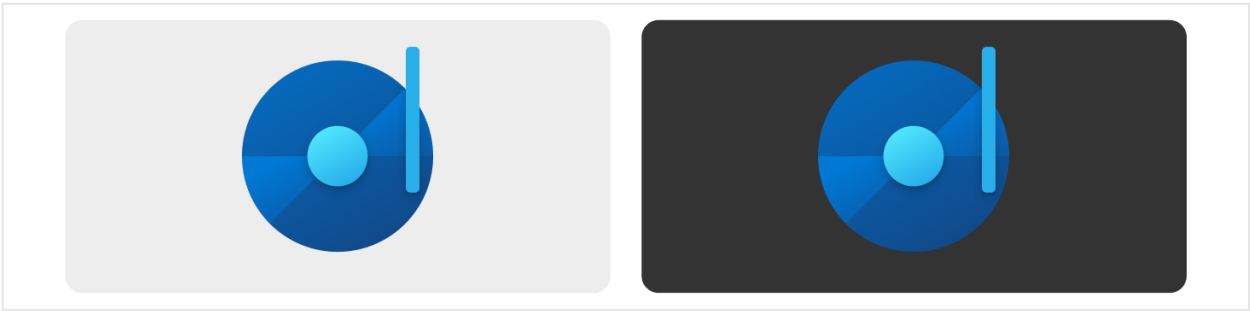
Analogous gradients



Analogous gradients should be at the same angle as the monochrome, but don't always have to be. Typically lighter hues should be on top left to avoid looking overly dramatic but also to be as consistent as possible with the monochrome.

Design guidance: Contrast, shadow, and perspective

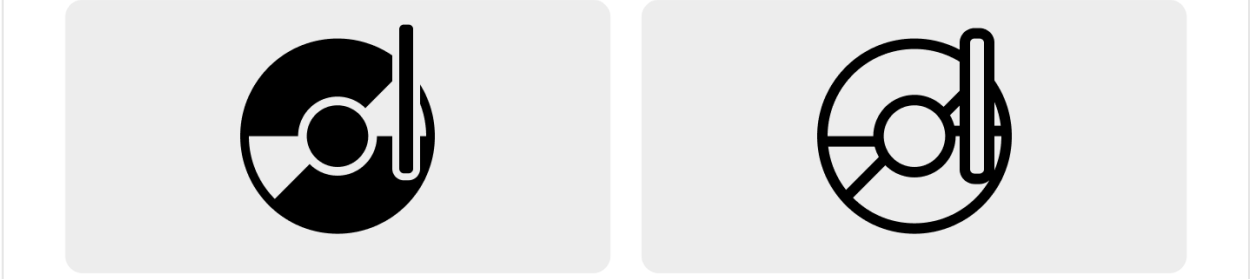
Color contrast



Accessibility is a high priority for Microsoft. App icons are primarily displayed on either light and dark backgrounds but displayed over desktop background images and tints or shades of the system accent color. It is difficult to make an icon 100% accessible on every background, but there are several things you can do to ensure your icon is as accessible as possible.

- Use color values in all 3 ranges, dark, medium, light.
- Make sure at least half of your icon passes a 3.0:1 contrast ratio on light and dark theme.
- Some hue values are more difficult than others. Yellow will never pass an accessible contrast ratio on light theme until it's brown. Reds are more difficult on dark theme.
- Though not required, you have the option to provide separate light and dark theme assets for Taskbar, Start and other theme-sensitive areas of Windows.

High contrast

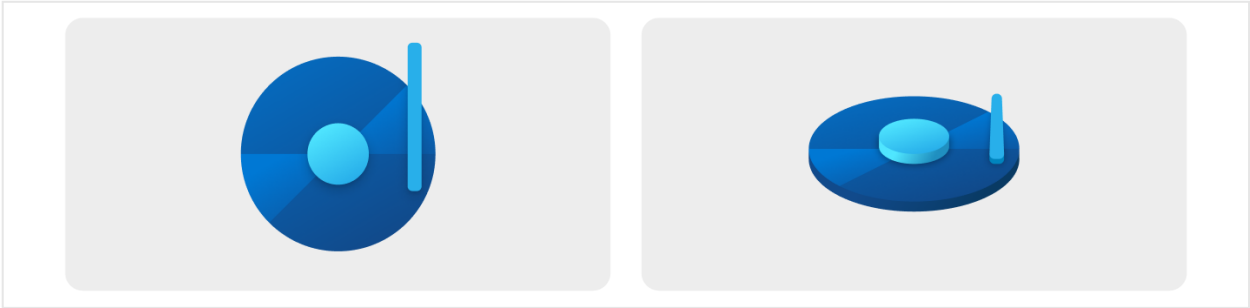


> **Tip**
>
> Windows 11 no longer requires high contrast assets for app icons.

High contrast icons are black and white and should be a direct representation of your app icon. Often the high contrast icon can be created from the color version using a

solid fill and line. Avoid gradients in high contrast icons. Sometimes monoline icons are required for in-app experiences should be designed according to these guidelines.

Layering and shadow



Top/Front facing view.

Isometric view to illustrate z-depth. For illustrative purposes only; not a suggested design option.

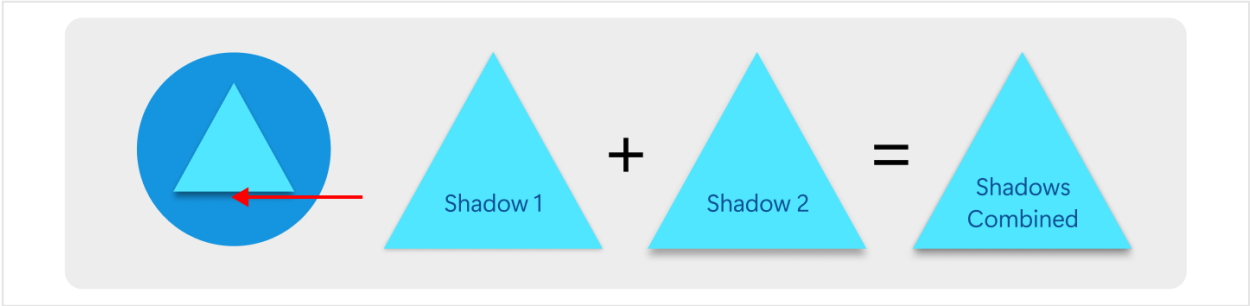
Icons are composed of flat objects sitting on top of the layers below it.

- Use as few layers as possible, and minimize extreme contrasts of scale between shapes.
- Use drop shadows within icons to create definition between object layers and visually connect components to each other within the icon design.
- In general, shadows cast from light onto dark shapes have the best result.
- Inner shadows should only cast a shadow on the graphic symbol, not on to the surrounding background.
- There are two types of inner shadow both of which have two shadows each

Shadow construction

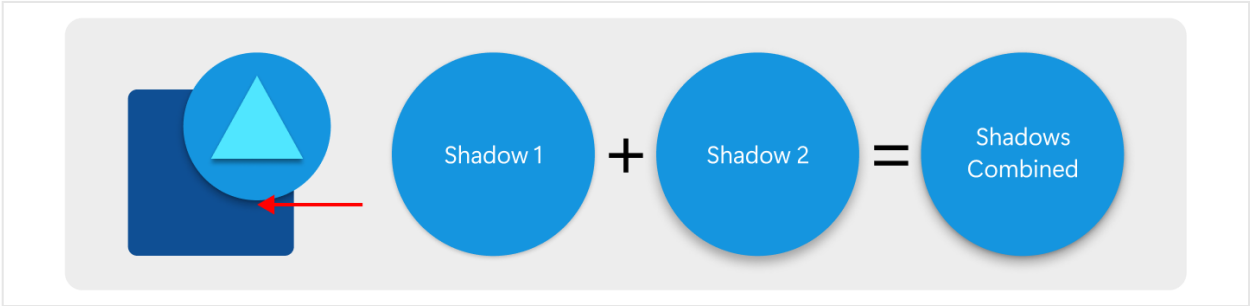
All of these values are to be rendered at 48x48 px and scaled up or down from there. If this is not adhered to, shadows will be inconsistent across the icon system. There are two types of object shadows both of which have two shadows each. Objects within the same metaphor have a shadow with slightly less blur.

Same metaphor



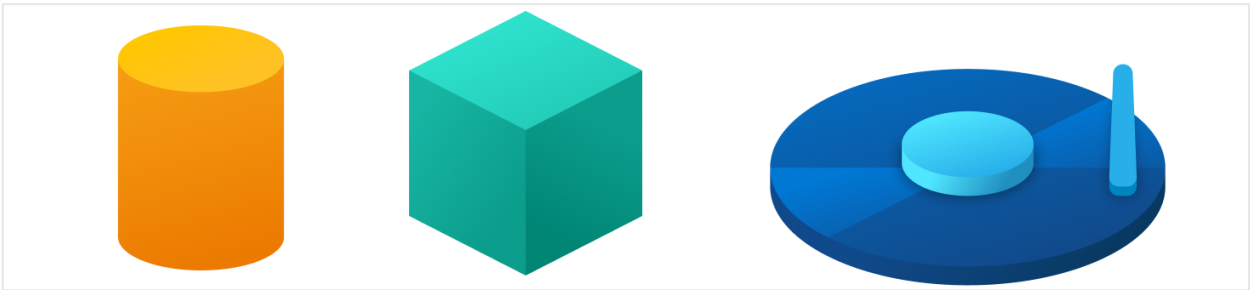
This shadow is used when you have content within a single metaphor that needs some depth. It's not always necessary to do this, but single object metaphors need some depth to feel like part of the system. the blue on shadow 2 is the only difference.

Separate metaphor



This shadow is used when you have two objects that overlap each other but are not necessarily part of the same metaphor. The shadow should be masked into the shape below it.

Perspective



The icons on the left are fairly simple; perspective is not recommended, but may be appropriate here. The icon on the right is too complex for perspective, so using it for this icon is not recommended.

Icons should be drawn with a straight-on perspective to present the metaphor in a simple easy to understand way. Exceptions are cases where the metaphor doesn't read well without viewing another side of it. For example a cylinder viewed straight on is a rectangle so the top could be added to show that it has volume. The other exception is

when an app is related to 3d where it makes sense to show dimension. In both cases the previous guidelines about flat objects still applies. Layers should always be flat and perpendicular to the viewing angle.

Construct your Windows app's icon

Article • 06/02/2022

Once you've designed your app's icon, you need to create the icon files themselves. Because Windows supports multiple themes, display resolutions, and scale factors, you should provide multiple versions of your icon to make sure it looks great on every device, at any size.

Icon sizes (WPF, UWP, WinUI)

Windows will display your app icon at a variety of sizes depending on where your icon is being displayed and the user's display scale settings. The following table lists all the possible sizes that Windows may use to display your icon.

Icon sizes (Win32)

Windows ICO files have been around for a long time. There are standard sizes that are used which is a subset of the full set above.

Icon scaling

When Windows displays your app's icon, it will look for an exact size match first. If there is no exact match it will look for the next size above and scale down. Including more icon sizes with your app means Windows will more often have a pixel-perfect match, and reduce the amount of scaling applied to scaled icons.

Expand table

Windows 11 scale factor	100%	125%	150%	200%	250%	300%	400%
Context menu, title bar, system tray	16px	20px	24px	32px	40px	48px	64px
Taskbar, search results, Start all apps list	24px	30px	36px	48px	60px	72px	96px
Start pins	32px	40px	48px	64px	80px	96px	256px

> **Note**
>
> Apps should have, at the bare minimum: 16x16, 24x24, 32x32, 48x48, and 256x256. This covers the most common icon sizes, and by providing a 256px icon, ensures Windows should only ever scale your icon down, never up.

Transparent backgrounds

Icons look best with a transparent background. If your app's branding requires your icon be plated on a background, that's okay too. However, you'll have to re-implement some theming functionality that transparent icons get for free. For example, you might provide a version of your app's icon plated on a two different backgrounds, one better suited to a light theme and the other to a dark theme.

Complete list of icons and variations

Windows utilizes different icon assets in different UI contexts. The usage has changed a little between Windows 10 & Windows 11.

The lists below define the specific filenames Windows expects to contain each corresponding icon.

App icon

In Windows 10 and 11, the AppList icon is your app's primary icon. It will be used in several places, including the Taskbar, Start pins, the all app list, and the search results list. Windows 11 selects an appropriate icon for the all apps list based on the current scale factor, but Windows 10 uses specific, explicitly defined icons if you provide them.

Separate files for all three theme variations (default, light theme, dark theme) are required, even if the icon is the same. If you do not provide these files, your icon will appear on a system icon plate to ensure a minimum contrast ratio.

App List Target Size (Required)

- AppList.targetsize-16.png
- AppList.targetsize-20.png
- AppList.targetsize-24.png
- AppList.targetsize-30.png
- AppList.targetsize-32.png
- AppList.targetsize-36.png
- AppList.targetsize-40.png
- AppList.targetsize-48.png
- AppList.targetsize-60.png
- AppList.targetsize-64.png
- AppList.targetsize-72.png
- AppList.targetsize-80.png
- AppList.targetsize-96.png
- AppList.targetsize-256.png

Dark theme (Required)

- AppList.targetsize-16_altform-unplated.png
- AppList.targetsize-20_altform-unplated.png
- AppList.targetsize-24_altform-unplated.png
- AppList.targetsize-30_altform-unplated.png
- AppList.targetsize-32_altform-unplated.png
- AppList.targetsize-36_altform-unplated.png
- AppList.targetsize-40_altform-unplated.png
- AppList.targetsize-48_altform-unplated.png
- AppList.targetsize-60_altform-unplated.png
- AppList.targetsize-64_altform-unplated.png
- AppList.targetsize-72_altform-unplated.png
- AppList.targetsize-80_altform-unplated.png
- AppList.targetsize-96_altform-unplated.png
- AppList.targetsize-256_altform-unplated.png

Light theme (Required)

- AppList.targetsize-16_altform-lightunplated.png
- AppList.targetsize-20_altform-lightunplated.png
- AppList.targetsize-24_altform-lightunplated.png
- AppList.targetsize-30_altform-lightunplated.png
- AppList.targetsize-32_altform-lightunplated.png
- AppList.targetsize-36_altform-lightunplated.png

- AppList.targetsize-40_altform-lightunplated.png
- AppList.targetsize-48_altform-lightunplated.png
- AppList.targetsize-60_altform-lightunplated.png
- AppList.targetsize-64_altform-lightunplated.png
- AppList.targetsize-72_altform-lightunplated.png
- AppList.targetsize-80_altform-lightunplated.png
- AppList.targetsize-96_altform-lightunplated.png
- AppList.targetsize-256_altform-lightunplated.png

App List Scale (Windows 10) (Optional)

- AppList.scale-100.png
- AppList.scale-125.png
- AppList.scale-150.png
- AppList.scale-200.png
- AppList.scale-400.png

Light theme (Windows 10) (Optional)

- AppList.scale-100_altform-colorful_theme-light.png
- AppList.scale-125_altform-colorful_theme-light.png
- AppList.scale-150_altform-colorful_theme-light.png
- AppList.scale-200_altform-colorful_theme-light.png
- AppList.scale-400_altform-colorful_theme-light.png

> ⓘ **Note**
>
> If you do not include the targetsize-*-altform-unplated assets above your icon will scale to a smaller size and will get an undesirable backplate behind the icon on Taskbar and Start.

Tiles

Windows 10 supports four tile sizes: small, medium, wide, and large.

Default / dark theme (partially required)

- SmallTile.scale-100.png
- SmallTile.scale-125.png
- SmallTile.scale-150.png
- SmallTile.scale-200.png
- SmallTile.scale-400.png

- MedTile.scale-100.png
- MedTile.scale-125.png
- MedTile.scale-150.png
- MedTile.scale-200.png
- MedTile.scale-400.png

- WideTile.scale-100.png
- WideTile.scale-125.png
- WideTile.scale-150.png
- WideTile.scale-200.png
- WideTile.scale-400.png

- LargeTile.scale-100.png
- LargeTile.scale-125.png
- LargeTile.scale-150.png
- LargeTile.scale-200.png
- LargeTile.scale-400.png

Light theme (optional)

- SmallTile.scale-100_altform-colorful_theme-light.png
- SmallTile.scale-125_altform-colorful_theme-light.png
- SmallTile.scale-150_altform-colorful_theme-light.png
- SmallTile.scale-200_altform-colorful_theme-light.png
- SmallTile.scale-400_altform-colorful_theme-light.png

- MedTile.scale-100_altform-colorful_theme-light.png
- MedTile.scale-125_altform-colorful_theme-light.png
- MedTile.scale-150_altform-colorful_theme-light.png
- MedTile.scale-200_altform-colorful_theme-light.png
- MedTile.scale-400_altform-colorful_theme-light.png

- WideTile.scale-100_altform-colorful_theme-light.png
- WideTile.scale-125_altform-colorful_theme-light.png
- WideTile.scale-150_altform-colorful_theme-light.png
- WideTile.scale-200_altform-colorful_theme-light.png
- WideTile.scale-400_altform-colorful_theme-light.png

- LargeTile.scale-100_altform-colorful_theme-light.png
- LargeTile.scale-125_altform-colorful_theme-light.png
- LargeTile.scale-150_altform-colorful_theme-light.png
- LargeTile.scale-200_altform-colorful_theme-light.png
- LargeTile.scale-400_altform-colorful_theme-light.png

> **Note**
>
> Windows 11 does not use the tile assets, but currently at minimum the Medium tile assets at 100% are required to publish to the Microsoft Store. If your app is Windows 10 & 11 compatible it is suggested that you include as many tile assets as possible.

Splash screen

Splash screens can also be Light and Dark theme aware like the App icon assets.

Default

- SplashScreen.scale-100.png
- SplashScreen.scale-125.png
- SplashScreen.scale-150.png
- SplashScreen.scale-200.png
- SplashScreen.scale-400.png

Dark theme (Optional)

- SplashScreen.scale-100_altform-colorful_theme-dark.png
- SplashScreen.scale-125_altform-colorful_theme-dark.png
- SplashScreen.scale-150_altform-colorful_theme-dark.png
- SplashScreen.scale-200_altform-colorful_theme-dark.png
- SplashScreen.scale-400_altform-colorful_theme-dark.png

Light theme (Optional)

- SplashScreen.scale-100_altform-colorful_theme-light.png
- SplashScreen.scale-125_altform-colorful_theme-light.png
- SplashScreen.scale-150_altform-colorful_theme-light.png
- SplashScreen.scale-200_altform-colorful_theme-light.png
- SplashScreen.scale-400_altform-colorful_theme-light.png

Badge logo

Badge icons are used on the Windows 10 lock screen and for most apps are not required.

Windows 10 (optional)

- BadgeLogo.scale-100.png
- BadgeLogo.scale-125.png
- BadgeLogo.scale-150.png
- BadgeLogo.scale-200.png
- BadgeLogo.scale-400.png

Package logo (Microsoft Store logo)

These assets are required to publish to the Microsoft Store.

Default / Dark theme (required)

- StoreLogo.scale-100.png
- StoreLogo.scale-125.png
- StoreLogo.scale-150.png
- StoreLogo.scale-200.png
- StoreLogo.scale-400.png

Light theme (optional)

- StoreLogo.scale-100_altform-colorful_theme-light.png
- StoreLogo.scale-125_altform-colorful_theme-light.png
- StoreLogo.scale-150_altform-colorful_theme-light.png
- StoreLogo.scale-200_altform-colorful_theme-light.png
- StoreLogo.scale-400_altform-colorful_theme-light.png

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

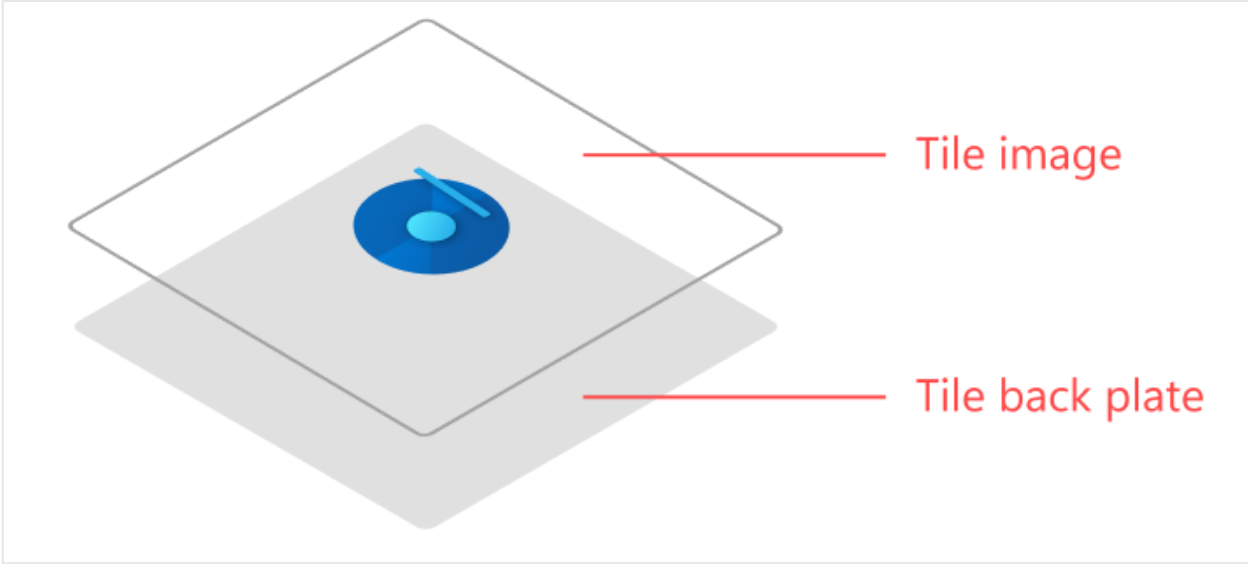
Construction guidelines for Windows 10 icons

Article • 06/02/2022

Apps running on versions of Windows with Live Tile support have additional options for how their app's icon is displayed within its tile. Apps that choose to provide tile images should provide images for all tile sizes at all common scale factors.

Windows 10 October 2020 Update introduced theme-aware Start Menu tiles, which select the tile background color based on the user's theme rather than using the color specified in the VisualElement section of the application manifest. Apps can still choose a color that suits their tile background color for versions of Windows 10 prior to October 2020.

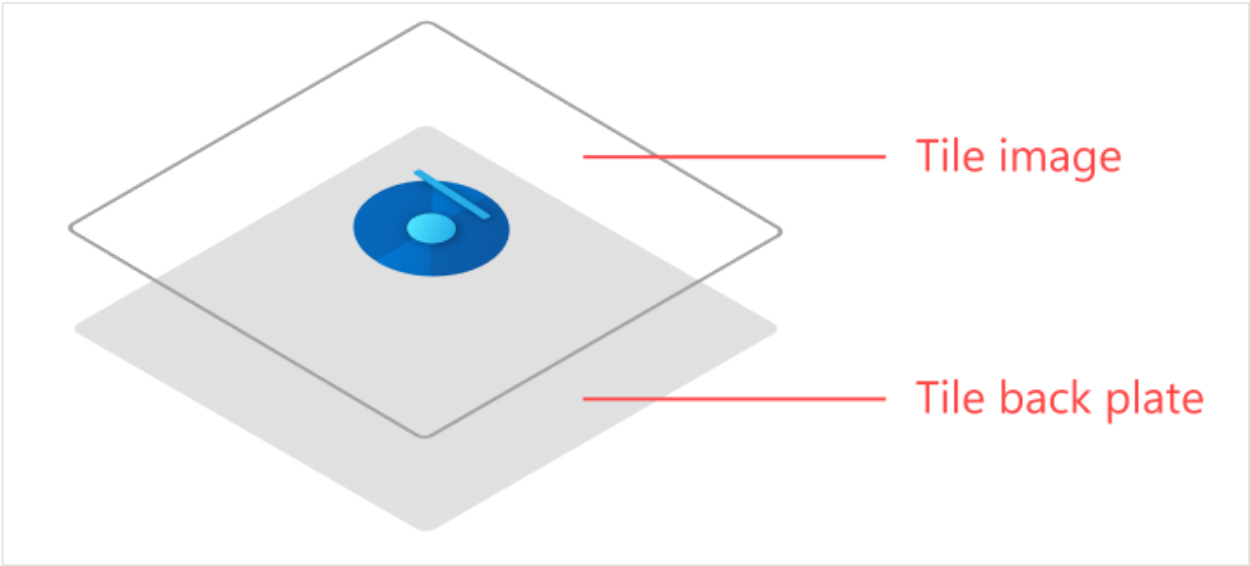
Creating an icon-based tile



The easiest way to look great on Live Tiles is to just display your app's icon against a transparent background. This is the "standard" way to do tiles. To achieve this look, you can either create tile images with transparent backgrounds, or just let Windows display your app's icon on a regular tile.

Creating a full bleed tile

When required, apps can create full-bleed Live Tile images to fully customize their tile. Typically, this functionality is used by games. Non-game apps should generally not use full-bleed tiles, because it will make your tile stand out awkwardly compared to "standard" icon-based tiles. Full bleed tiles can be used to achieve several looks.



Apps using a full bleed tile can make their icon any size they want. To do this, simply make your icon the size you would like it to display when the tile is shown. App icons should never take up the complete tile. Be sure to use at least 16% margins on each side.

Some apps, usually games, might want to display a full-bleed image instead of an icon. In this case, do not use margins - the image should take up all available space.

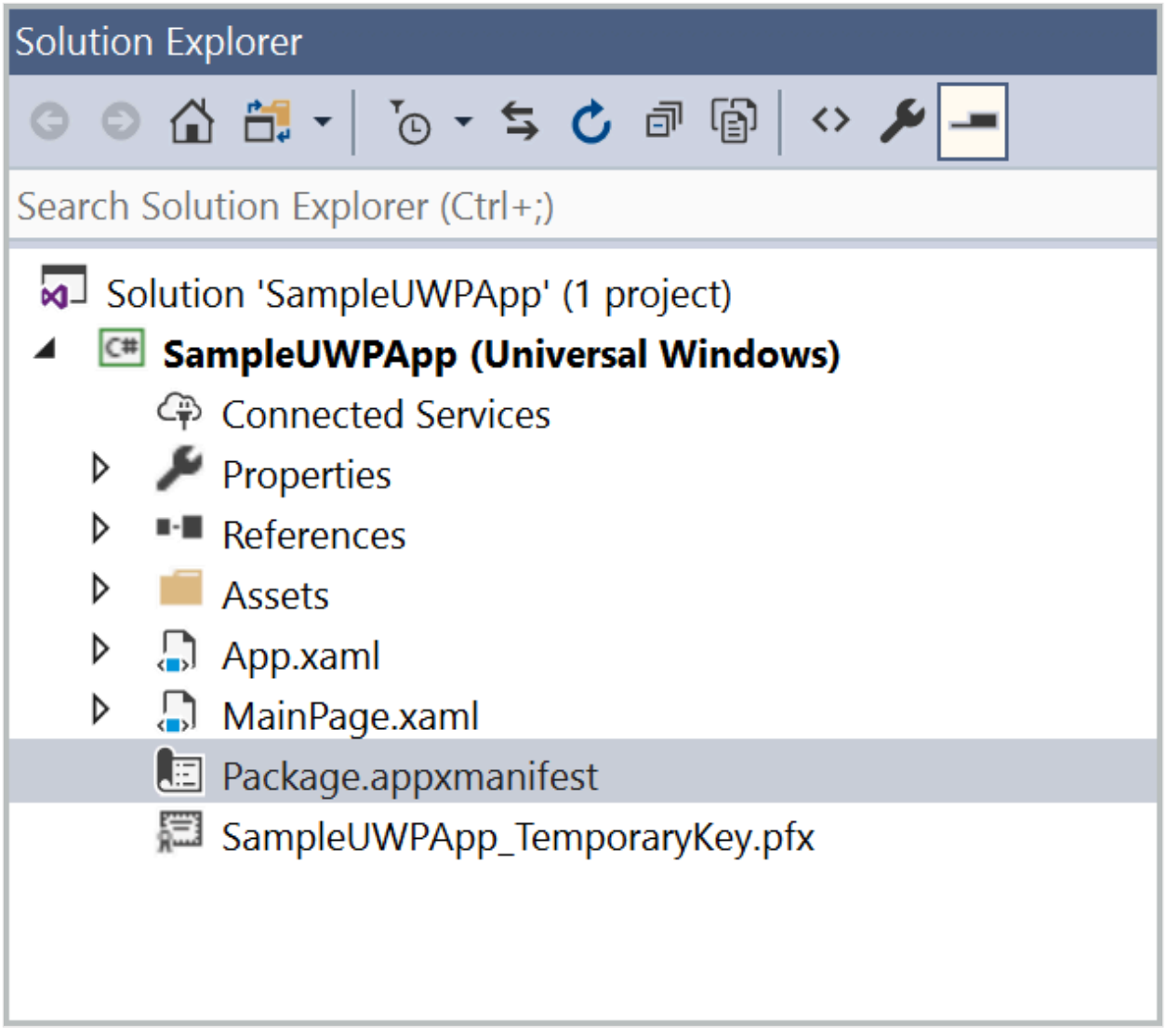
Create icons using Visual Studio's asset generation tool

Article • 05/24/2022

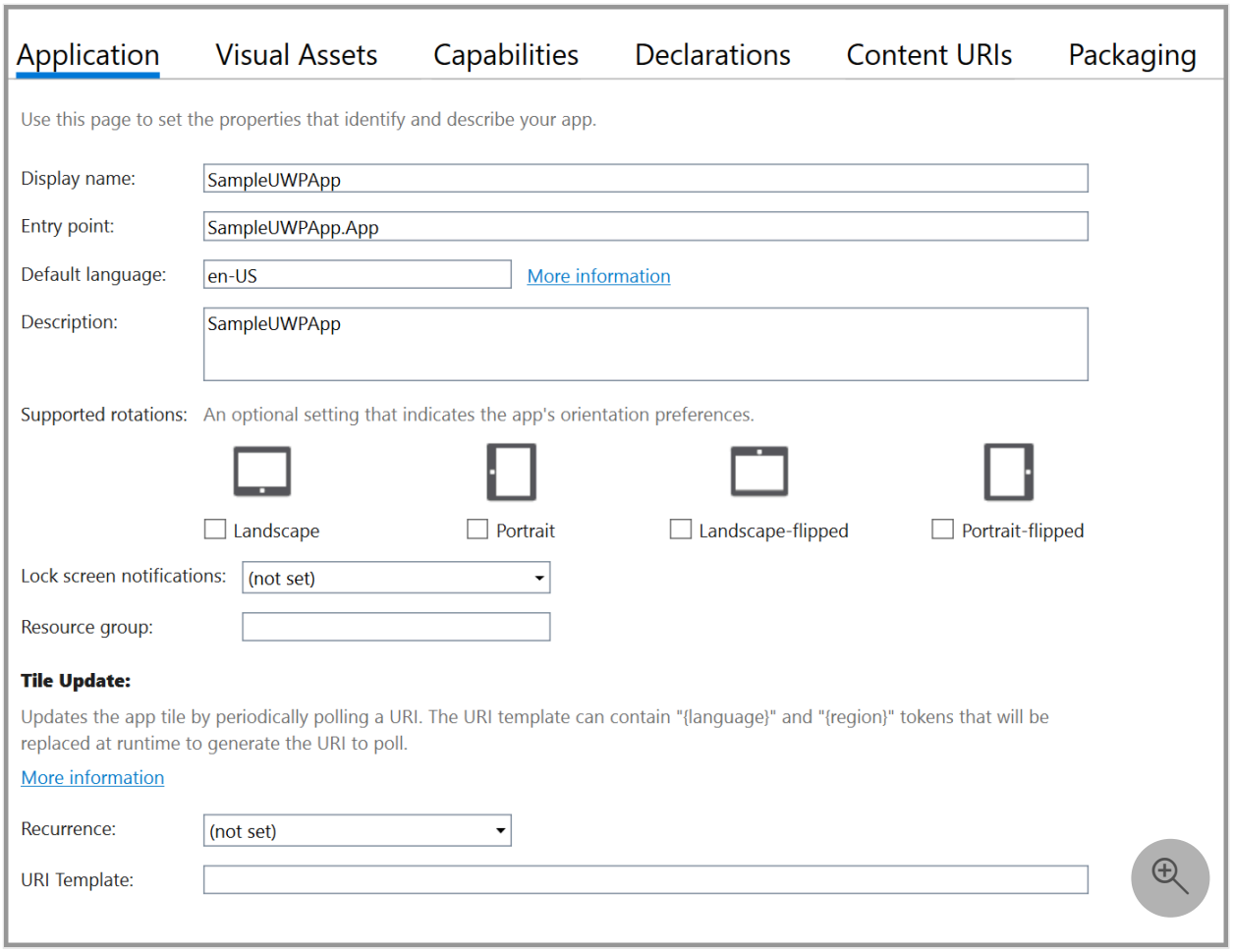
While handcrafting each icon file will create the best, most consistent user experience, teams running short on resources can take advantage of Visual Studio's Manifest Designer. This tool can create an entire set of app icons and tile images from a single image. This is useful to create an initial set of icons, but will not achieve the same result as handcrafting each icon file, as Visual Studio will have to scale your image to create the required image sizes.

Launching the Manifest Designer

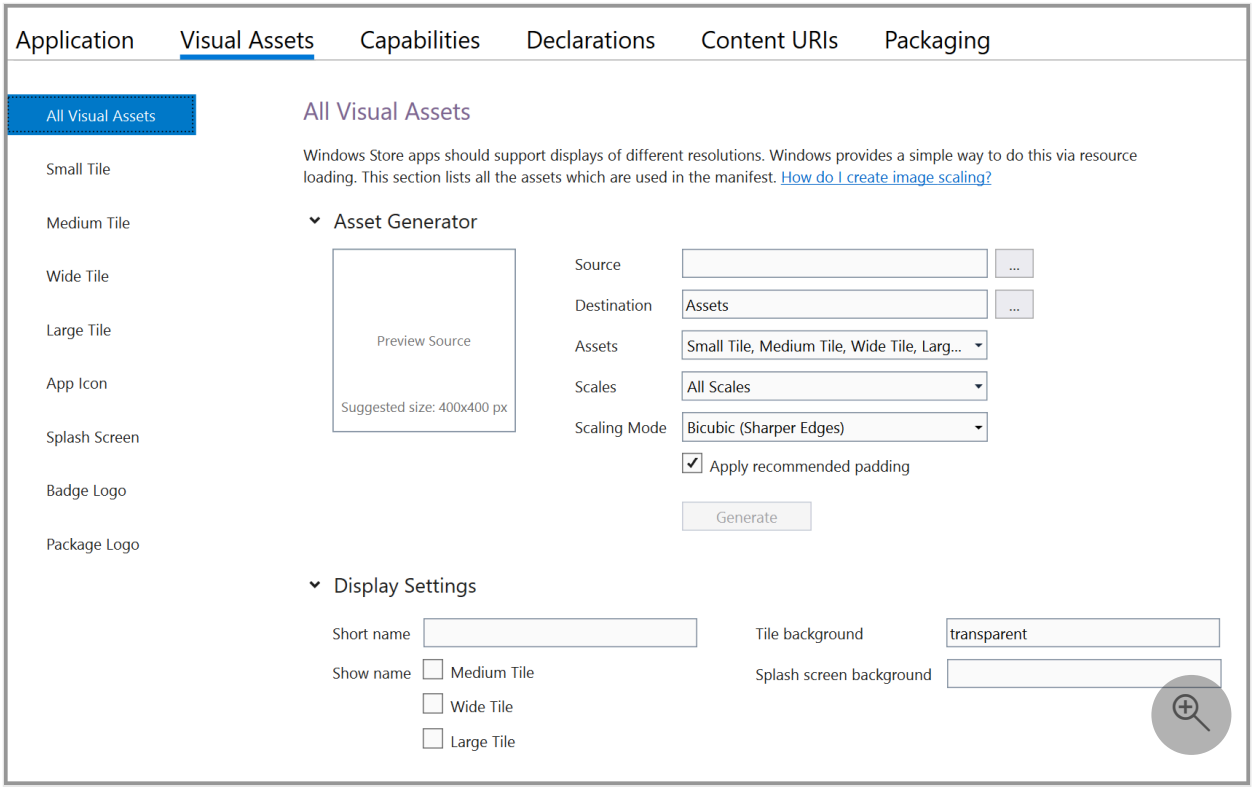
1. Use Visual Studio to open a WinUI or UWP project.
2. In the **Solution Explorer**, double-click the Package.appxmanifest file.



3. Visual Studio displays the Manifest Designer.



4. Click the **Visual Assets** tab.



Generating icons with the Manifest Designer

1. Click the `...` next to the Source field and select the image you want to use. For best results, use a vector-based image, Adobe Illustrator file, or PDF. If you're using a bitmap image, make sure it's at least 400 by 400 pixels so that you get sharp results.

2. In the Display Settings section, configure these options:

    - **Short name**: Specify a short name for your app.
    - **Show name**: Indicate whether you want to display the short name on medium, wide, or large tiles.
    - **Tile background**: Specify the hex value or a color name for the tile background color. For example, #464646. The default value is transparent. **NOTE:** This setting will be ignored on versions of Windows that support theme-aware Live Tiles.
    - **Splash screen background** (Optional): Specify the hex value or color name for the splash screen background.

3. Click **Generate**.

> **Note**
>
> Visual Studio's Manifest Designer doesn't generate a badge logo by default. That's because your badge logo is unique and probably shouldn't match your other app icons. For more info, see Badge notifications for Windows apps.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Segoe MDL2 Assets icons

Article • 07/18/2024

This article provides developer guidelines for using the Segoe MDL2 Assets icons and lists the font glyphs along with their unicode values and descriptive names.

Important APIs:

- FontIcon class

About Segoe MDL2 Assets

> **ⓘ Important**
>
> With the release of Windows 10, the `Segoe MDL2 Assets` font replaced the Windows 8/8.1 `Segoe UI Symbol` icon font.
>
> With the release of Windows 11, the **`Segoe Fluent Icons`** font replaced `Segoe MDL2 Assets` as the recommended symbol icon font. `Segoe UI Symbol` and `Segoe MDL2 Assets` are still available, but we recommend updating your app to use the Segoe Fluent Icons font.

Most of the icons included in the `Segoe MDL2 Assets` font are mapped to the Private Use Area of Unicode (PUA). The PUA allows font developers to assign private Unicode values to glyphs that don't map to existing code points. This is useful when creating a symbol font, but it creates an interoperability problem. If the font is not available, the glyphs won't show up. Use these glyphs only when you can explicitly specify the `Segoe MDL2 Assets` font. If you are working with tiles, you can't use these glyphs because you can't specify the tile font and PUA glyphs are not available via font-fallback.

Unlike with `Segoe UI Symbol`, the icons in the `Segoe MDL2 Assets` font are not intended for use in-line with text. This means that some older "tricks" like the progressive disclosure arrows no longer apply. Likewise, since all of the new icons are sized and positioned the same, they do not have to be made with zero width; we have just made sure they work as a set. Ideally, you can overlay two icons that were designed as a set and they will fall into place. We may do this to allow colorization in the code. For example, U+EA3A and U+EA3B were created for the Start tile Badge status. Because these are already centered the circle fill can be colored for different states.

Layering and mirroring

All glyphs in `Segoe MDL2 Assets` have the same fixed width with a consistent height and left origin point, so layering and colorization effects can be achieved by drawing glyphs directly on top of each other. This example show a black outline drawn on top of the zero-width red heart.

Many of the icons also have mirrored forms available for use in languages that use right-to-left text directionality such as Arabic, Dari, Persian, and Hebrew.

Using the icons

To use a glyph from the `Segoe MDL2 Assets` font, then use a **FontIcon**.

XAML

```
<FontIcon FontFamily="Segoe MDL2 Assets" Glyph="&#xE700;"/>
```

How do I get this font?

- On Windows: There's nothing you need to do, the font comes with Windows.
- On a Mac, you need to download and install the font: Get the Segoe UI and MDL2 icon fonts

Icon list

Please keep in mind that the `Segoe MDL2 Assets` font includes many more icons than we can show here. Many of the icons are intended for specialized purposes and are not typically used anywhere else.

> **Note**
>
> Glyphs with prefixes ranging from **E0-** to **E5-** (e.g. E001, E5B1) are currently marked as legacy and we recommend that they not be used.

The following tables display all Segoe MDL2 Assets icons and their respective unicode values and descriptive names. Select a range from the following list to view glyphs

according to the PUA range they belong to.

- PUA E700-E900
- PUA EA00-EC00
- PUA ED00-EF00
- PUA F000-F200
- PUA F300-F500
- PUA F600-F800

PUA E700-E900

The following table of glyphs displays unicode points prefixed from E7- to E9-.

Back to top

Expand table

Glyph	Unicode point	Description
	E700	GlobalNavigationButton
	E701	Wifi
	E702	Bluetooth
	E703	Connect
	E704	InternetSharing
	E705	VPN
	E706	Brightness
	E707	MapPin
	E708	QuietHours
	E709	Airplane

	E70A	Tablet
	E70B	QuickNote
	E70C	RememberedDevice
	E70D	ChevronDown
	E70E	ChevronUp
	E70F	Edit
	E710	Add
	E711	Cancel
	E712	More
	E713	Setting
	E714	Video
	E715	Mail
	E716	People
	E717	Phone
	E718	Pin
	E719	Shop
	E71A	Stop
	E71B	Link
	E71C	Filter

	E71D	AllApps
	E71E	Zoom
	E71F	ZoomOut
	E720	Microphone
	E721	Search
	E722	Camera
	E723	Attach
	E724	Send
	E725	SendFill
	E726	WalkSolid
	E727	InPrivate
	E728	FavoriteList
	E729	PageSolid
	E72A	Forward
	E72B	Back
	E72C	Refresh
	E72D	Share
	E72E	Lock
	E730	ReportHacked

	E731	EMI
	E734	FavoriteStar
	E735	FavoriteStarFill
	E736	ReadingMode
	E737	Favicon
	E738	Remove
	E739	Checkbox
	E73A	CheckboxComposite
	E73B	CheckboxFill
	E73C	CheckboxIndeterminate
	E73D	CheckboxCompositeReversed
	E73E	CheckMark
	E73F	BackToWindow
	E740	FullScreen
	E741	ResizeTouchLarger
	E742	ResizeTouchSmaller
	E743	ResizeMouseSmall
	E744	ResizeMouseMedium
	E745	ResizeMouseWide

	E746	ResizeMouseTall
	E747	ResizeMouseLarge
	E748	SwitchUser
	E749	Print
	E74A	Up
	E74B	Down
	E74C	OEM
	E74D	Delete
	E74E	Save
	E74F	Mute
	E750	BackSpaceQWERTY
	E751	ReturnKey
	E752	UpArrowShiftKey
	E753	Cloud
	E754	Flashlight
	E755	RotationLock
	E756	CommandPrompt
	E759	SIPMove
	E75A	SIPUndock

	E75B	SIPRedock
	E75C	EraseTool
	E75D	UnderscoreSpace
	E75E	GripperTool
	E75F	Dialpad
	E760	PageLeft
	E761	PageRight
	E762	MultiSelect
	E763	KeyboardLeftHanded
	E764	KeyboardRightHanded
	E765	KeyboardClassic
	E766	KeyboardSplit
	E767	Volume
	E768	Play
	E769	Pause
	E76B	ChevronLeft
	E76C	ChevronRight
	E76D	InkingTool
	E76E	Emoji2

	E76F	GripperBarHorizontal
	E770	System
	E771	Personalize
	E772	Devices
	E773	SearchAndApps
	E774	Globe
	E775	TimeLanguage
	E776	EaseOfAccess
	E777	UpdateRestore
	E778	HangUp
	E779	ContactInfo
	E77A	Unpin
	E77B	Contact
	E77C	Memo
	E77E	IncomingCall
	E77F	Paste
	E780	PhoneBook
	E781	LEDLight
	E783	Error

	E784	GripperBarVertical
	E785	Unlock
	E786	Slideshow
	E787	Calendar
	E788	GripperResize
	E789	Megaphone
	E78A	Trim
	E78B	NewWindow
	E78C	SaveLocal
	E790	Color
	E791	DataSense
	E792	SaveAs
	E793	Light
	E799	AspectRatio
	E7A5	DataSenseBar
	E7A6	Redo
	E7A7	Undo
	E7A8	Crop
	E7AC	OpenWith

	E7AD	Rotate
	E7B3	RedEye
	E7B5	SetlockScreen
	E7B7	MapPin2
	E7B8	Package
	E7BA	Warning
	E7BC	ReadingList
	E7BE	Education
	E7BF	ShoppingCart
	E7C0	Train
	E7C1	Flag
	E7C3	Page
	E7C4	TaskView
	E7C5	BrowsePhotos
	E7C6	HalfStarLeft
	E7C7	HalfStarRight
	E7C8	Record
	E7C9	TouchPointer
	E7DE	LangJPN

	E7E3	Ferry
	E7E6	Highlight
	E7E7	ActionCenterNotification
	E7E8	PowerButton
	E7EA	ResizeTouchNarrower
	E7EB	ResizeTouchShorter
	E7EC	DrivingMode
	E7ED	RingerSilent
	E7EE	OtherUser
	E7EF	Admin
	E7F0	CC
	E7F1	SDCard
	E7F2	CallForwarding
	E7F3	SettingsDisplaySound
	E7F4	TVMonitor
	E7F5	Speakers
	E7F6	Headphone
	E7F7	DeviceLaptopPic
	E7F8	DeviceLaptopNoPic

	E7F9	DeviceMonitorRightPic
	E7FA	DeviceMonitorLeftPic
	E7FB	DeviceMonitorNoPic
	E7FC	Game
	E7FD	HorizontalTabKey
	E802	StreetsideSplitMinimize
	E803	StreetsideSplitExpand
	E804	Car
	E805	Walk
	E806	Bus
	E809	TiltUp
	E80A	TiltDown
	E80B	CallControl
	E80C	RotateMapRight
	E80D	RotateMapLeft
	E80F	Home
	E811	ParkingLocation
	E812	MapCompassTop
	E813	MapCompassBottom

	E814	IncidentTriangle
	E815	Touch
	E816	MapDirections
	E819	StartPoint
	E81A	StopPoint
	E81B	EndPoint
	E81C	History
	E81D	Location
	E81E	MapLayers
	E81F	Accident
	E821	Work
	E822	Construction
	E823	Recent
	E825	Bank
	E826	DownloadMap
	E829	InkingToolFill2
	E82A	HighlightFill2
	E82B	EraseToolFill
	E82C	EraseToolFill2

	E82D	Dictionary
	E82E	DictionaryAdd
	E82F	ToolTip
	E830	ChromeBack
	E835	ProvisioningPackage
	E836	AddRemoteDevice
	E838	FolderOpen
	E839	Ethernet
	E83A	ShareBroadband
	E83B	DirectAccess
	E83C	DialUp
	E83D	DefenderApp
	E83E	BatteryCharging9
	E83F	Battery10
	E840	Pinned
	E841	PinFill
	E842	PinnedFill
	E843	PeriodKey
	E844	PuncKey

	E845	RevToggleKey
	E846	RightArrowKeyTime1
	E847	RightArrowKeyTime2
	E848	LeftQuote
	E849	RightQuote
	E84A	DownShiftKey
	E84B	UpShiftKey
	E84C	PuncKey0
	E84D	PuncKeyLeftBottom
	E84E	RightArrowKeyTime3
	E84F	RightArrowKeyTime4
	E850	Battery0
	E851	Battery1
	E852	Battery2
	E853	Battery3
	E854	Battery4
	E855	Battery5
	E856	Battery6
	E857	Battery7

	E858	Battery8
	E859	Battery9
	E85A	BatteryCharging0
	E85B	BatteryCharging1
	E85C	BatteryCharging2
	E85D	BatteryCharging3
	E85E	BatteryCharging4
	E85F	BatteryCharging5
	E860	BatteryCharging6
	E861	BatteryCharging7
	E862	BatteryCharging8
	E863	BatterySaver0
	E864	BatterySaver1
	E865	BatterySaver2
	E866	BatterySaver3
	E867	BatterySaver4
	E868	BatterySaver5
	E869	BatterySaver6
	E86A	BatterySaver7

	E86B	BatterySaver8
	E86C	SignalBars1
	E86D	SignalBars2
	E86E	SignalBars3
	E86F	SignalBars4
	E870	SignalBars5
	E871	SignalNotConnected
	E872	Wifi1
	E873	Wifi2
	E874	Wifi3
	E875	MobSIMLock
	E876	MobSIMMissing
	E877	Vibrate
	E878	RoamingInternational
	E879	RoamingDomestic
	E87A	CallForwardInternational
	E87B	CallForwardRoaming
	E87C	JpnRomanji
	E87D	JpnRomanjiLock

个英	E87E	JpnRomanjiShift
个	E87F	JpnRomanjiShiftLock
↖	E880	StatusDataTransfer
🔒↖	E881	StatusDataTransferVPN
2	E882	StatusDualSIM2
2🔒	E883	StatusDualSIM2VPN
1	E884	StatusDualSIM1
1🔒	E885	StatusDualSIM1VPN
📶	E886	StatusSGLTE
📞	E887	StatusSGLTECell
📶🔒	E888	StatusSGLTEDataVPN
🔒	E889	StatusVPN
	E88A	WifiHotspot
한	E88B	LanguageKor
繁	E88C	LanguageCht
简	E88D	LanguageChs
	E88E	USB
	E88F	InkingToolFill
	E890	View

	E891	HighlightFill
	E892	Previous
	E893	Next
	E894	Clear
	E895	Sync
	E896	Download
	E897	Help
	E898	Upload
	E899	Emoji
	E89A	TwoPage
	E89B	LeaveChat
	E89C	MailForward
	E89E	RotateCamera
	E89F	ClosePane
	E8A0	OpenPane
	E8A1	PreviewLink
	E8A2	AttachCamera
	E8A3	ZoomIn
	E8A4	Bookmarks

	E8A5	Document
	E8A6	ProtectedDocument
	E8A7	OpenInNewWindow
	E8A8	MailFill
	E8A9	ViewAll
	E8AA	VideoChat
	E8AB	Switch
	E8AC	Rename
	E8AD	Go
	E8AE	SurfaceHub
	E8AF	Remote
	E8B0	Click
	E8B1	Shuffle
	E8B2	Movies
	E8B3	SelectAll
	E8B4	Orientation
	E8B5	Import
	E8B6	ImportAll
	E8B7	Folder

	E8B8	Webcam
	E8B9	Picture
	E8BA	Caption
	E8BB	ChromeClose
	E8BC	ShowResults
	E8BD	Message
	E8BE	Leaf
	E8BF	CalendarDay
	E8C0	CalendarWeek
	E8C1	Characters
	E8C2	MailReplyAll
	E8C3	Read
	E8C4	ShowBcc
	E8C5	HideBcc
	E8C6	Cut
	E8C7	PaymentCard
	E8C8	Copy
	E8C9	Important
	E8CA	MailReply

	E8CB	Sort
	E8CC	MobileTablet
	E8CD	DisconnectDrive
	E8CE	MapDrive
	E8CF	ContactPresence
	E8D0	Priority
	E8D1	GotoToday
	E8D2	Font
	E8D3	FontColor
	E8D4	Contact2
	E8D5	FolderFill
	E8D6	Audio
	E8D7	Permissions
	E8D8	DisableUpdates
	E8D9	Unfavorite
	E8DA	OpenLocal
	E8DB	Italic
	E8DC	Underline
	E8DD	Bold

	E8DE	MoveToFolder
	E8DF	LikeDislike
	E8E0	Dislike
	E8E1	Like
	E8E2	AlignRight
	E8E3	AlignCenter
	E8E4	AlignLeft
	E8E5	OpenFile
	E8E6	ClearSelection
	E8E7	FontDecrease
	E8E8	FontIncrease
	E8E9	FontSize
	E8EA	CellPhone
	E8EB	Reshare
	E8EC	Tag
	E8ED	RepeatOne
	E8EE	RepeatAll
	E8EF	Calculator
	E8F0	Directions

	E8F1	Library
	E8F2	ChatBubbles
	E8F3	PostUpdate
	E8F4	NewFolder
	E8F5	CalendarReply
	E8F6	UnsyncFolder
	E8F7	SyncFolder
	E8F8	BlockContact
	E8F9	SwitchApps
	E8FA	AddFriend
	E8FB	Accept
	E8FC	GoToStart
	E8FD	BulletedList
	E8FE	Scan
	E8FF	Preview
	E902	Group
	E904	ZeroBars
	E905	OneBar
	E906	TwoBars

	E907	ThreeBars
	E908	FourBars
	E909	World
	E90A	Comment
	E90B	MusicInfo
	E90C	DockLeft
	E90D	DockRight
	E90E	DockBottom
	E90F	Repair
	E910	Accounts
	E911	DullSound
	E912	Manage
	E913	Street
	E914	Printer3D
	E915	RadioBullet
	E916	Stopwatch
	E91B	Photo
	E91C	ActionCenter
	E91F	FullCircleMask

	E921	ChromeMinimize
	E922	ChromeMaximize
	E923	ChromeRestore
	E924	Annotation
	E925	BackSpaceQWERTYSm
	E926	BackSpaceQWERTYMd
	E927	Swipe
	E928	Fingerprint
	E929	Handwriting
	E92C	ChromeBackToWindow
	E92D	ChromeFullScreen
	E92E	KeyboardStandard
	E92F	KeyboardDismiss
	E930	Completed
	E931	ChromeAnnotate
	E932	Label
	E933	IBeam
	E934	IBeamOutline
	E935	FlickDown

	E936	FlickUp
	E937	FlickLeft
	E938	FlickRight
	E939	FeedbackApp
	E93C	MusicAlbum
	E93E	Streaming
	E943	Code
	E944	ReturnToWindow
	E945	LightningBolt
	E946	Info
	E947	CalculatorMultiply
	E948	CalculatorAddition
	E949	CalculatorSubtract
	E94A	CalculatorDivide
	E94B	CalculatorSquareroot
	E94C	CalculatorPercentage
	E94D	CalculatorNegate
	E94E	CalculatorEqualTo
	E94F	CalculatorBackspace

	E950	Component
	E951	DMC
	E952	Dock
	E953	MultimediaDMS
	E954	MultimediaDVR
	E955	MultimediaPMP
	E956	PrintfaxPrinterFile
	E957	Sensor
	E958	StorageOptical
	E95A	Communications
	E95B	Headset
	E95D	Projector
	E95E	Health
	E95F	Wire
	E960	Webcam2
	E961	Input
	E962	Mouse
	E963	Smartcard
	E964	SmartcardVirtual

	E965	MediaStorageTower
	E966	ReturnKeySm
	E967	GameConsole
	E968	Network
	E969	StorageNetworkWireless
	E96A	StorageTape
	E96D	ChevronUpSmall
	E96E	ChevronDownSmall
	E96F	ChevronLeftSmall
	E970	ChevronRightSmall
	E971	ChevronUpMed
	E972	ChevronDownMed
	E973	ChevronLeftMed
	E974	ChevronRightMed
	E975	Devices2
	E976	ExpandTile
	E977	PC1
	E978	PresenceChicklet
	E979	PresenceChickletVideo

	E97A	Reply
	E97B	SetTile
	E97C	Type
	E97D	Korean
	E97E	HalfAlpha
	E97F	FullAlpha
	E980	Key12On
	E981	ChineseChangjie
	E982	QWERTYOn
	E983	QWERTYOff
	E984	ChineseQuick
	E985	Japanese
	E986	FullHiragana
	E987	FullKatakana
	E988	HalfKatakana
	E989	ChineseBoPoMoFo
	E98A	ChinesePinyin
	E98F	ConstructionCone
	E990	XboxOneConsole

	E992	Volume0
	E993	Volume1
	E994	Volume2
	E995	Volume3
	E996	BatteryUnknown
	E998	WifiAttentionOverlay
	E99A	Robot
	E9A1	TapAndSend
	E9A6	FitPage
	E9A8	PasswordKeyShow
	E9A9	PasswordKeyHide
	E9AA	BidiLtr
	E9AB	BidiRtl
	E9AC	ForwardSm
	E9AD	CommaKey
	E9AE	DashKey
	E9AF	DullSoundKey
	E9B0	HalfDullSound
	E9B1	RightDoubleQuote

	E9B2	LeftDoubleQuote
	E9B3	PuncKeyRightBottom
	E9B4	PuncKey1
	E9B5	PuncKey2
	E9B6	PuncKey3
	E9B7	PuncKey4
	E9B8	PuncKey5
	E9B9	PuncKey6
	E9BA	PuncKey9
	E9BB	PuncKey7
	E9BC	PuncKey8
	E9CA	Frigid
	E9CE	Unknown
	E9D2	AreaChart
	E9D5	CheckList
	E9D9	Diagnostic
	E9E9	Equalizer
	E9F3	Process
	E9F5	Processing

	E9F9	ReportDocument

PUA EA00-EC00

The following table of glyphs displays unicode points prefixed from EA- to EC-.

Back to top

Expand table

Glyph	Unicode point	Description
	EA0C	VideoSolid
	EA0D	MixedMediaBadge
	EA14	DisconnectDisplay
	EA18	Shield
	EA1F	Info2
*	EA21	ActionCenterAsterisk
	EA24	Beta
	EA35	SaveCopy
	EA37	List
	EA38	Asterisk
	EA39	ErrorBadge
	EA3A	CircleRing
	EA3B	CircleFill

	EA3C	MergeCall
	EA3D	PrivateCall
	EA3F	Record2
	EA40	AllAppsMirrored
	EA41	BookmarksMirrored
	EA42	BulletedListMirrored
	EA43	CallForwardInternationalMirrored
	EA44	CallForwardRoamingMirrored
	EA47	ChromeBackMirrored
	EA48	ClearSelectionMirrored
	EA49	ClosePaneMirrored
	EA4A	ContactInfoMirrored
	EA4B	DockRightMirrored
	EA4C	DockLeftMirrored
	EA4E	ExpandTileMirrored
	EA4F	GoMirrored
	EA50	GripperResizeMirrored
	EA51	HelpMirrored
	EA52	ImportMirrored

	EA53	ImportAllMirrored
	EA54	LeaveChatMirrored
	EA55	ListMirrored
	EA56	MailForwardMirrored
	EA57	MailReplyMirrored
	EA58	MailReplyAllMirrored
	EA5B	OpenPaneMirrored
	EA5C	OpenWithMirrored
	EA5E	ParkingLocationMirrored
	EA5F	ResizeMouseMediumMirrored
	EA60	ResizeMouseSmallMirrored
	EA61	ResizeMouseTallMirrored
	EA62	ResizeTouchNarrowerMirrored
	EA63	SendMirrored
	EA64	SendFillMirrored
	EA65	ShowResultsMirrored
	EA69	Media
	EA6A	SyncError
	EA6C	Devices3

	EA79	SlowMotionOn
	EA80	Lightbulb
	EA81	StatusCircle
	EA82	StatusTriangle
	EA83	StatusError
	EA84	StatusWarning
	EA86	Puzzle
	EA89	CalendarSolid
	EA8A	HomeSolid
	EA8B	ParkingLocationSolid
	EA8C	ContactSolid
	EA8D	ConstructionSolid
	EA8E	AccidentSolid
	EA8F	Ringer
	EA90	PDF
	EA91	ThoughtBubble
	EA92	HeartBroken
	EA93	BatteryCharging10
	EA94	BatterySaver9

	EA95	BatterySaver10
	EA97	CallForwardingMirrored
	EA98	MultiSelectMirrored
	EA99	Broom
	EAC2	ForwardCall
	EADF	Trackers
	EAFC	Market
	EB05	PieSingle
	EB0F	StockDown
	EB11	StockUp
	EB3C	Design
	EB41	Website
	EB42	Drop
	EB44	Radar
	EB47	BusSolid
	EB48	FerrySolid
	EB49	StartPointSolid
	EB4A	StopPointSolid
	EB4B	EndPointSolid

	EB4C	AirplaneSolid
	EB4D	TrainSolid
	EB4E	WorkSolid
	EB4F	ReminderFill
	EB50	Reminder
	EB51	Heart
	EB52	HeartFill
	EB55	EthernetError
	EB56	EthernetWarning
	EB57	StatusConnecting1
	EB58	StatusConnecting2
	EB59	StatusUnsecure
	EB5A	WifiError0
	EB5B	WifiError1
	EB5C	WifiError2
	EB5D	WifiError3
	EB5E	WifiError4
	EB5F	WifiWarning0
	EB60	WifiWarning1

	EB61	WifiWarning2
	EB62	WifiWarning3
	EB63	WifiWarning4
	EB66	Devices4
	EB67	NUIIris
	EB68	NUIFace
	EB7E	EditMirrored
	EB82	NUIFPStartSlideHand
	EB83	NUIFPStartSlideAction
	EB84	NUIFPContinueSlideHand
	EB85	NUIFPContinueSlideAction
	EB86	NUIFPRollRightHand
	EB87	NUIFPRollRightHandAction
	EB88	NUIFPRollLeftHand
	EB89	NUIFPRollLeftAction
	EB8A	NUIFPPressHand
	EB8B	NUIFPPressAction
	EB8C	NUIFPPressRepeatHand
	EB8D	NUIFPPressRepeatAction

	EB90	StatusErrorFull
	EB91	TaskViewExpanded
	EB95	Certificate
	EB96	BackSpaceQWERTYLg
	EB97	ReturnKeyLg
	EB9D	FastForward
	EB9E	Rewind
	EB9F	Photo2
	EBA0	MobBattery0
	EBA1	MobBattery1
	EBA2	MobBattery2
	EBA3	MobBattery3
	EBA4	MobBattery4
	EBA5	MobBattery5
	EBA6	MobBattery6
	EBA7	MobBattery7
	EBA8	MobBattery8
	EBA9	MobBattery9
	EBAA	MobBattery10

	EBAB	MobBatteryCharging0
	EBAC	MobBatteryCharging1
	EBAD	MobBatteryCharging2
	EBAE	MobBatteryCharging3
	EBAF	MobBatteryCharging4
	EBB0	MobBatteryCharging5
	EBB1	MobBatteryCharging6
	EBB2	MobBatteryCharging7
	EBB3	MobBatteryCharging8
	EBB4	MobBatteryCharging9
	EBB5	MobBatteryCharging10
	EBB6	MobBatterySaver0
	EBB7	MobBatterySaver1
	EBB8	MobBatterySaver2
	EBB9	MobBatterySaver3
	EBBA	MobBatterySaver4
	EBBB	MobBatterySaver5
	EBBC	MobBatterySaver6
	EBBD	MobBatterySaver7

	EBBE	MobBatterySaver8
	EBBF	MobBatterySaver9
	EBC0	MobBatterySaver10
	EBC3	DictionaryCloud
	EBC4	ResetDrive
	EBC5	VolumeBars
	EBC6	Project
	EBD2	AdjustHologram
	EBD4	WifiCallBars
	EBD5	WifiCall0
	EBD6	WifiCall1
	EBD7	WifiCall2
	EBD8	WifiCall3
	EBD9	WifiCall4
	EBDA	Family
	EBDB	LockFeedback
	EBDE	DeviceDiscovery
	EBE6	WindDirection
	EBE7	RightArrowKeyTime0

	EBE8	Bug
	EBFC	TabletMode
	EBFD	StatusCircleLeft
	EBFE	StatusTriangleLeft
	EBFF	StatusErrorLeft
	EC00	StatusWarningLeft
	EC02	MobBatteryUnknown
	EC05	NetworkTower
	EC06	CityNext
	EC07	CityNext2
	EC08	Courthouse
	EC09	Groceries
	EC0A	Sustainable
	EC0B	BuildingEnergy
	EC11	ToggleFilled
	EC12	ToggleBorder
	EC13	SliderThumb
	EC14	ToggleThumb
	EC15	MiracastLogoSmall

	EC16	MiracastLogoLarge
	EC19	PLAP
	EC1B	Badge
	EC1E	SignalRoaming
	EC20	MobileLocked
	EC24	InsiderHubApp
	EC25	PersonalFolder
	EC26	HomeGroup
	EC27	MyNetwork
	EC31	KeyboardFull
	EC32	Cafe
	EC37	MobSignal1
	EC38	MobSignal2
	EC39	MobSignal3
	EC3A	MobSignal4
	EC3B	MobSignal5
	EC3C	MobWifi1
	EC3D	MobWifi2
	EC3E	MobWifi3

	EC3F	MobWifi4
	EC40	MobAirplane
	EC41	MobBluetooth
	EC42	MobActionCenter
	EC43	MobLocation
	EC44	MobWifiHotspot
	EC45	LanguageJpn
	EC46	MobQuietHours
	EC47	MobDrivingMode
	EC48	SpeedOff
	EC49	SpeedMedium
	EC4A	SpeedHigh
	EC4E	ThisPC
	EC4F	MusicNote
	EC50	FileExplorer
	EC51	FileExplorerApp
	EC52	LeftArrowKeyTime0
	EC54	MicOff
	EC55	MicSleep

	EC56	MicError
	EC57	PlaybackRate1x
	EC58	PlaybackRateOther
	EC59	CashDrawer
	EC5A	BarcodeScanner
	EC5B	ReceiptPrinter
	EC5C	MagStripeReader
	EC61	CompletedSolid
	EC64	CcompanionApp
	EC6C	Favicon2
	EC6D	SwipeRevealArt
	EC71	MicOn
	EC72	MicClipping
	EC74	TabletSelected
	EC75	MobileSelected
	EC76	LaptopSelected
	EC77	TVMonitorSelected
	EC7A	DeveloperTools
	EC7E	MobCallForwarding

	EC7F	MobCallForwardingMirrored
	EC80	BodyCam
	EC81	PoliceCar
	EC87	Draw
	EC88	DrawSolid
	EC8A	LowerBrightness
	EC8F	ScrollUpDown
	EC92	DateTime
	ECA5	Tiles
	ECA7	PartyLeader
	ECAA	AppIconDefault
	ECAD	Calories
	ECB9	BandBattery0
	ECBA	BandBattery1
	ECBB	BandBattery2
	ECBC	BandBattery3
	ECBD	BandBattery4
	ECBE	BandBattery5
	ECBF	BandBattery6

	ECC4	AddSurfaceHub
	ECC5	DevUpdate
	ECC6	Unit
	ECC8	AddTo
	ECC9	RemoveFrom
	ECCA	RadioBtnOff
	ECCB	RadioBtnOn
	ECCC	RadioBullet2
	ECCD	ExploreContent
	ECE4	Blocked2
	ECE7	ScrollMode
	ECE8	ZoomMode
	ECE9	PanMode
	ECF0	WiredUSB
	ECF1	WirelessUSB
	ECF3	USBSafeConnect

PUA ED00-EF00

The following table of glyphs displays unicode points prefixed from ED- to EF-.

Back to top

Expand table

Glyph	Unicode point	Description
	ED0C	ActionCenterNotificationMirrored
	ED0D	ActionCenterMirrored
	ED0E	SubscriptionAdd
	ED10	ResetDevice
	ED11	SubscriptionAddMirrored
	ED14	QRCode
	ED15	Feedback
	ED1E	Subtitles
	ED1F	SubtitlesAudio
	ED25	OpenFolderHorizontal
	ED28	CalendarMirrored
	ED2A	MobeSIM
	ED2B	MobeSIMNoProfile
	ED2C	MobeSIMLocked
	ED2D	MobeSIMBusy
	ED2E	SignalError
	ED2F	StreamingEnterprise
	ED30	Headphone0

	ED31	Headphone1
	ED32	Headphone2
	ED33	Headphone3
	ED35	Apps
	ED39	KeyboardBrightness
	ED3A	KeyboardLowerBrightness
	ED3C	SkipBack10
	ED3D	SkipForward30
	ED41	TreeFolderFolder
	ED42	TreeFolderFolderFill
	ED43	TreeFolderFolderOpen
	ED44	TreeFolderFolderOpenFill
	ED47	MultimediaDMP
	ED4C	KeyboardOneHanded
	ED4D	Narrator
	ED53	EmojiTabPeople
	ED54	EmojiTabSmilesAnimals
	ED55	EmojiTabCelebrationObjects
	ED56	EmojiTabFoodPlants

	ED57	EmojiTabTransitPlaces
	ED58	EmojiTabSymbols
	ED59	EmojiTabTextSmiles
	ED5A	EmojiTabFavorites
	ED5B	EmojiSwatch
	ED5C	ConnectApp
	ED5D	CompanionDeviceFramework
	ED5E	Ruler
	ED5F	FingerInking
	ED60	StrokeErase
	ED61	PointErase
	ED62	ClearAllInk
	ED63	Pencil
	ED64	Marker
	ED65	InkingCaret
	ED66	InkingColorOutline
	ED67	InkingColorFill
	EDA2	HardDrive
	EDA3	NetworkAdapter

	EDA4	Touchscreen
	EDA5	NetworkPrinter
	EDA6	CloudPrinter
	EDA7	KeyboardShortcut
	EDA8	BrushSize
	EDA9	NarratorForward
	EDAA	NarratorForwardMirrored
	EDAB	SyncBadge12
	EDAC	RingerBadge12
	EDAD	AsteriskBadge12
	EDAE	ErrorBadge12
	EDAF	CircleRingBadge12
	EDB0	CircleFillBadge12
	EDB1	ImportantBadge12
	EDB3	MailBadge12
	EDB4	PauseBadge12
	EDB5	PlayBadge12
	EDC6	PenWorkspace
	EDD6	CaretRight8

	EDD9	CaretLeftSolid8
	EDDA	CaretRightSolid8
	EDDB	CaretUpSolid8
	EDDC	CaretDownSolid8
	EDE0	Strikethrough
	EDE1	Export
	EDE2	ExportMirrored
	EDE3	ButtonMenu
	EDE4	CloudSearch
	EDE5	PinyinIMELogo
	EDFB	CalligraphyPen
	EE35	ReplyMirrored
	EE3F	LockscreenDesktop
	EE40	TaskViewSettings
	EE47	MiniExpand2Mirrored
	EE49	MiniContract2Mirrored
	EE4A	Play36
	EE56	PenPalette
	EE57	GuestUser

	EE63	SettingsBattery
	EE64	TaskbarPhone
	EE65	LockScreenGlance
	EE6F	GenericScan
	EE71	ImageExport
	EE77	WifiEthernet
	EE79	ActionCenterQuiet
	EE7A	ActionCenterQuietNotification
	EE92	TrackersMirrored
	EE93	DateTimeMirrored
	EE94	Wheel
	EEA3	VirtualMachineGroup
	EECA	ButtonView2
	EF15	PenWorkspaceMirrored
	EF16	PenPaletteMirrored
	EF17	StrokeEraseMirrored
	EF18	PointEraseMirrored
	EF19	ClearAllInkMirrored
	EF1F	BackgroundToggle

	EF20	Marquee
	EF2C	ChromeCloseContrast
	EF2D	ChromeMinimizeContrast
	EF2E	ChromeMaximizeContrast
	EF2F	ChromeRestoreContrast
	EF31	TrafficLight
	EF3B	Replay
	EF3C	Eyedropper
	EF3D	LineDisplay
	EF3E	PINPad
	EF3F	SignatureCapture
	EF40	ChipCardCreditCardReader
	EF58	PlayerSettings
	EF6B	LandscapeOrientation
	EF90	Flow
	EFA5	Touchpad
	EFA9	Speech

PUA F000-F200

The following table of glyphs displays unicode points prefixed from F0- to F2-.

Back to top

Expand table

Glyph	Unicode point	Description
	F000	KnowledgeArticle
	F003	Relationship
	F080	DefaultAPN
	F081	UserAPN
	F085	DoublePinyin
	F08C	BlueLight
	F093	ButtonA
	F094	ButtonB
	F095	ButtonY
	F096	ButtonX
	F0AD	ArrowUp8
	F0AE	ArrowDown8
	F0AF	ArrowRight8
	F0B0	ArrowLeft8
	F0B2	QuarentinedItems
	F0B3	QuarentinedItemsMirrored

	F0B4	Protractor
	F0B5	ChecklistMirrored
	F0B6	StatusCircle7
	F0B7	StatusCheckmark7
	F0B8	StatusErrorCircle7
	F0B9	Connected
	F0C6	PencilFill
	F0C7	CalligraphyFill
	F0CA	QuarterStarLeft
	F0CB	QuarterStarRight
	F0CC	ThreeQuarterStarLeft
	F0CD	ThreeQuarterStarRight
	F0CE	QuietHoursBadge12
	F0D2	BackMirrored
	F0D3	ForwardMirrored
	F0D5	ChromeBackContrast
	F0D6	ChromeBackContrastMirrored
	F0D7	ChromeBackToWindowContrast
	F0D8	ChromeFullScreenContrast

	F0E2	GridView
	F0E3	ClipboardList
	F0E4	ClipboardListMirrored
	F0E5	OutlineQuarterStarLeft
	F0E6	OutlineQuarterStarRight
	F0E7	OutlineHalfStarLeft
	F0E8	OutlineHalfStarRight
	F0E9	OutlineThreeQuarterStarLeft
	F0EA	OutlineThreeQuarterStarRight
	F0EB	SpatialVolume0
	F0EC	SpatialVolume1
	F0ED	SpatialVolume2
	F0EE	SpatialVolume3
	F0EF	ApplicationGuard
	F0F7	OutlineStarLeftHalf
	F0F8	OutlineStarRightHalf
	F0F9	ChromeAnnotateContrast
	F0FB	DefenderBadge12
	F103	DetachablePC

	F108	LeftStick
	F109	RightStick
	F10A	TriggerLeft
	F10B	TriggerRight
	F10C	BumperLeft
	F10D	BumperRight
	F10E	Dpad
	F110	EnglishPunctuation
	F111	ChinesePunctuation
	F119	HMD
	F11B	CtrlSpatialRight
	F126	PaginationDotOutline10
	F127	PaginationDotSolid10
	F128	StrokeErase2
	F129	SmallErase
	F12A	LargeErase
	F12B	FolderHorizontal
	F12E	MicrophoneListening
	F12F	StatusExclamationCircle7

	F131	Video360
	F133	GiftboxOpen
	F136	StatusCircleOuter
	F137	StatusCircleInner
	F138	StatusCircleRing
	F139	StatusTriangleOuter
	F13A	StatusTriangleInner
	F13B	StatusTriangleExclamation
	F13C	StatusCircleExclamation
	F13D	StatusCircleErrorX
	F13E	StatusCircleCheckmark
	F13F	StatusCircleInfo
	F140	StatusCircleBlock
	F141	StatusCircleBlock2
	F142	StatusCircleQuestionMark
	F143	StatusCircleSync
	F146	Dial1
	F147	Dial2
	F148	Dial3

4	F149	Dial4
5	F14A	Dial5
6	F14B	Dial6
7	F14C	Dial7
8	F14D	Dial8
9	F14E	Dial9
10	F14F	Dial10
11	F150	Dial11
12	F151	Dial12
13	F152	Dial13
14	F153	Dial14
15	F154	Dial15
16	F155	Dial16
	F156	DialShape1
	F157	DialShape2
	F158	DialShape3
	F159	DialShape4
	F161	TollSolid
	F163	TrafficCongestionSolid

	F164	ExploreContentSingle
	F165	CollapseContent
	F166	CollapseContentSingle
	F167	InfoSolid
	F168	GroupList
	F169	CaretBottomRightSolidCenter8
	F16A	ProgressRingDots
	F16B	Checkbox14
	F16C	CheckboxComposite14
	F16D	CheckboxIndeterminateCombo14
	F16E	CheckboxIndeterminateCombo
	F175	StatusPause7
	F17F	CharacterAppearance
	F180	Lexicon
	F182	ScreenTime
	F191	HeadlessDevice
	F193	NetworkSharing
	F19D	EyeGaze
	F19E	ToggleLeft

	F19F	ToggleRight
	F1AD	WindowsInsider
	F1CB	ChromeSwitch
	F1CC	ChromeSwitchContast
	F1D8	StatusCheckmark
	F1D9	StatusCheckmarkLeft
	F20C	KeyboardLeftAligned
	F20D	KeyboardRightAligned
	F210	KeyboardSettings
	F211	NetworkPhysical
	F22C	IOT
	F22E	UnknownMirrored
	F246	ViewDashboard
	F259	ExploitProtectionSettings
	F260	KeyboardNarrow
	F261	Keyboard12Key
	F26B	KeyboardDock
	F26C	KeyboardUndock
	F26D	KeyboardLeftDock

	F26E	KeyboardRightDock
	F270	Ear
	F271	PointerHand
	F272	Bullseye
	F2B7	LocaleLanguage

PUA F300-F500

The following table of glyphs displays unicode points prefixed from F3- to F5-.

Back to top

Expand table

Glyph	Unicode point	Description
	F32A	PassiveAuthentication
	F354	ColorSolid
	F384	NetworkOffline
	F385	NetworkConnected
	F386	NetworkConnectedCheckmark
	F3B1	SignOut
	F3CC	StatusInfo
	F3CD	StatusInfoLeft
	F3E2	NearbySharing

	F3E7	CtrlSpatialLeft
	F404	InteractiveDashboard
	F406	ClippingTool
	F407	RectangularClipping
	F408	FreeFormClipping
	F413	CopyTo
	F439	DynamicLock
	F45E	PenTips
	F45F	PenTipsMirrored
	F460	HWPJoin
	F461	HWPInsert
	F462	HWPStrikeThrough
	F463	HWPScratchOut
	F464	HWPSplit
	F465	HWPNewLine
	F466	HWPOverwrite
	F473	MobWifiWarning1
	F474	MobWifiWarning2
	F475	MobWifiWarning3

	F476	MobWifiWarning4
	F49A	Globe2
	F4A5	SpecialEffectSize
	F4A9	GIF
	F4AA	Sticker2
	F4BE	SurfaceHubSelected
	F4BF	HoloLensSelected
	F4C0	Earbud
	F4C3	MixVolumes
	F540	Safe
	F552	LaptopSecure
	F56D	PrintDefault
	F56E	PageMirrored
	F56F	LandscapeOrientationMirrored
	F570	ColorOff
	F571	PrintAllPages
	F572	PrintCustomRange
	F573	PageMarginPortraitNarrow
	F574	PageMarginPortraitNormal

	F575	PageMarginPortraitModerate
	F576	PageMarginPortraitWide
	F577	PageMarginLandscapeNarrow
	F578	PageMarginLandscapeNormal
	F579	PageMarginLandscapeModerate
	F57A	PageMarginLandscapeWide
	F57B	CollateLandscape
	F57C	CollatePortrait
	F57D	CollatePortraitSeparated
	F57E	DuplexLandscapeOneSided
	F57F	DuplexLandscapeOneSidedMirrored
	F580	DuplexLandscapeTwoSidedLongEdge
	F581	DuplexLandscapeTwoSidedLongEdgeMirrored
	F582	DuplexLandscapeTwoSidedShortEdge
	F583	DuplexLandscapeTwoSidedShortEdgeMirrored
	F584	DuplexPortraitOneSided
	F585	DuplexPortraitOneSidedMirrored
	F586	DuplexPortraitTwoSidedLongEdge
	F587	DuplexPortraitTwoSidedLongEdgeMirrored

	F588	DuplexPortraitTwoSidedShortEdge
	F589	DuplexPortraitTwoSidedShortEdgeMirrored
	F58A	PPSOneLandscape
	F58B	PPSTwoLandscape
	F58C	PPSTwoPortrait
	F58D	PPSFourLandscape
	F58E	PPSFourPortrait
	F58F	HolePunchOff
	F590	HolePunchPortraitLeft
	F591	HolePunchPortraitRight
	F592	HolePunchPortraitTop
	F593	HolePunchPortraitBottom
	F594	HolePunchLandscapeLeft
	F595	HolePunchLandscapeRight
	F596	HolePunchLandscapeTop
	F597	HolePunchLandscapeBottom
	F598	StaplingOff
	F599	StaplingPortraitTopLeft
	F59A	StaplingPortraitTopRight

	F59B	StaplingPortraitBottomRight
	F59C	StaplingPortraitTwoLeft
	F59D	StaplingPortraitTwoRight
	F59E	StaplingPortraitTwoTop
	F59F	StaplingPortraitTwoBottom
	F5A0	StaplingPortraitBookBinding
	F5A1	StaplingLandscapeTopLeft
	F5A2	StaplingLandscapeTopRight
	F5A3	StaplingLandscapeBottomLeft
	F5A4	StaplingLandscapeBottomRight
	F5A5	StaplingLandscapeTwoLeft
	F5A6	StaplingLandscapeTwoRight
	F5A7	StaplingLandscapeTwoTop
	F5A8	StaplingLandscapeTwoBottom
	F5A9	StaplingLandscapeBookBinding
	F5AA	StatusDataTransferRoaming
	F5AB	MobSIMError
	F5AC	CollateLandscapeSeparated
	F5AD	PPSOnePortrait

	F5AE	StaplingPortraitBottomLeft
	F5B0	PlaySolid
	F5E7	RepeatOff
	F5ED	Set
	F5EE	SetSolid
	F5EF	FuzzyReading
	F5F2	VerticalBattery0
	F5F3	VerticalBattery1
	F5F4	VerticalBattery2
	F5F5	VerticalBattery3
	F5F6	VerticalBattery4
	F5F7	VerticalBattery5
	F5F8	VerticalBattery6
	F5F9	VerticalBattery7
	F5FA	VerticalBattery8
	F5FB	VerticalBattery9
	F5FC	VerticalBattery10
	F5FD	VerticalBatteryCharging0
	F5FE	VerticalBatteryCharging1

	F5FF	VerticalBatteryCharging2

PUA F600-F800

The following table of glyphs displays unicode points prefixed from F6- to F8-.

Back to top

Expand table

Glyph	Unicode point	Description
	F600	VerticalBatteryCharging3
	F601	VerticalBatteryCharging4
	F602	VerticalBatteryCharging5
	F603	VerticalBatteryCharging6
	F604	VerticalBatteryCharging7
	F605	VerticalBatteryCharging8
	F606	VerticalBatteryCharging9
	F607	VerticalBatteryCharging10
	F608	VerticalBatteryUnknown
	F614	DoublePortrait
	F615	DoubleLandscape
	F616	SinglePortrait
	F617	SingleLandscape

	F618	SIMError
	F619	SIMMissing
	F61A	SIMLock
	F61B	eSIM
	F61C	eSIMNoProfile
	F61D	eSIMLocked
	F61E	eSIMBusy
	F61F	NoiseCancelation
	F620	NoiseCancelationOff
	F623	MusicSharing
	F624	MusicSharingOff
	F63C	CircleShapeSolid
	F657	WifiCallBars
	F658	WifiCall0
	F659	WifiCall1
	F65A	WifiCall2
	F65B	WifiCall3
	F65C	WifiCall4
	F69E	CHTLanguageBar

	F6A9	ComposeMode
	F6B8	ExpressiveInputEntry
	F6BA	EmojiTabMoreSymbols
	F6FA	WebSearch
	F712	Kiosk
	F714	RTTLogo
	F715	VoiceCall
	F716	GoToMessage
	F71A	ReturnToCall
	F71C	StartPresenting
	F71D	StopPresenting
	F71E	ProductivityMode
	F738	SetHistoryStatus
	F739	SetHistoryStatus2
	F73D	Keyboardsettings20
	F73E	OneHandedRight20
	F73F	OneHandedLeft20
	F740	Split20
	F741	Full20

	F742	Handwriting20
	F743	CheveronLeft20
	F744	CheveronLeft32
	F745	CheveronRight20
	F746	CheveronRight32
	F781	MicOff2
	F785	DeliveryOptimization
	F78A	CancelMedium
	F78B	SearchMedium
	F78C	AcceptMedium
	F78D	RevealPasswordMedium
	F7AD	DeleteWord
	F7AE	DeleteWordFill
	F7AF	DeleteLines
	F7B0	DeleteLinesFill
	F7B1	InstertWords
	F7B2	InstertWordsFill
	F7B3	JoinWords
	F7B4	JoinWordsFill

	F7B5	OverwriteWords
	F7B6	OverwriteWordsFill
	F7B7	AddNewLine
	F7B8	AddNewLineFill
	F7B9	OverwriteWordsKorean
	F7BA	OverwriteWordsFillKorean
	F7BB	EducationIcon
	F7ED	WindowSnipping
	F7EE	VideoCapture
	F809	StatusSecured
	F83B	NarratorApp
	F83D	PowerButtonUpdate
	F83E	RestartUpdate
	F83F	UpdateStatusDot
	F847	Eject
	F87B	Spelling
	F87C	SpellingKorean
	F87D	SpellingSerbian
	F87E	SpellingChinese

	F89A	FolderSelect
	F8A5	SmartScreen
	F8A6	ExploitProtection
	F8AA	AddBold
	F8AB	SubtractBold
	F8AC	BackSolidBold
	F8AD	ForwardSolidBold
	F8AE	PauseBold
	F8AF	ClickSolid
	F8B0	SettingsSolid
	F8B1	MicrophoneSolidBold
	F8B2	SpeechSolidBold
	F8B3	ClickedOutLoudSolidBold

Related articles

- Guidelines for icons
- Symbol enumeration
- FontIcon class

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

Windows developer feedback

Windows developer is an open source project. Select a link to

can also create and review issues and pull requests. For more information, see our contributor guide.

provide feedback:

- Open a documentation issue
- Provide product feedback

Segoe Fluent Icons font

Article • 12/09/2024

This article provides developer guidelines for using the Segoe Fluent Icons font and lists each icon along with its Unicode value and descriptive name.

Important APIs:

- FontIcon class

About Segoe Fluent Icons

> **Tip**
>
> With the release of Windows 11, the **`Segoe Fluent Icons`** font replaced `Segoe MDL2 Assets` as the recommended symbol icon font. `Segoe MDL2 Assets` is still available, but we recommend updating your app to use the `Segoe Fluent Icons` font.

Most of the icons included in the `Segoe Fluent Icons` font are mapped to the Private Use Area of Unicode (PUA). The PUA range is a non-standardized range of Unicode that allows font developers to define their own characters. This is useful when creating a symbol font, but it creates an interoperability problem when `Segoe Fluent Icons` is not available.

Icons in the `Segoe Fluent Icons` font are not intended for use in-line with text. This means that some older "tricks" like the progressive disclosure arrows no longer apply. Likewise, since all of the new icons are sized and positioned the same, they do not have to be made with zero width; we have made sure they work as a set.

Layering and mirroring

All glyphs in `Segoe Fluent Icons` have the same fixed width with a consistent height and left origin point, so layering and colorization effects can be achieved by drawing glyphs directly on top of each other. This example show a black outline drawn on top of the zero-width red heart.

XAML

```
<Grid>
    <FontIcon FontFamily="Segoe Fluent Icons" Glyph="&#xEB52;"
              Foreground="#C72335" />
    <FontIcon FontFamily="Segoe Fluent Icons" Glyph="&#xEB51;" />
</Grid>
```

Many of the icons also have mirrored forms available for use in languages that use right-to-left text directionality such as Arabic, Dari, Persian, and Hebrew.

Using the icons

If you are developing an app in XAML, you can use specified glyphs from `Segoe Fluent Icons` with a SymbolIcon and the Symbol enumeration.

XAML

```
<SymbolIcon Symbol="GlobalNavigationButton"/>
```

If you would like to use a glyph from the `Segoe Fluent Icons` font that is not included in the Symbol enum, set it as the Glyph property of a **FontIcon** control.

XAML

```
<FontIcon FontFamily="Segoe Fluent Icons" Glyph="&#xE700;"/>
```

You can also use the static resource `SymbolThemeFontFamily` to access `Segoe Fluent Icons`, instead of specifying the font by name:

XAML

```
<FontIcon FontFamily="{StaticResource SymbolThemeFontFamily}"
Glyph="&#xE700;"/>
```

> **Note**
>
> For optimal appearance, use these specific sizes: 16, 20, 24, 32, 40, 48, and 64. Deviating from these font sizes could lead to less crisp or blurry outcomes.

How do I get this font?

- On Windows 11: There's nothing you need to do, the font comes with Windows.
- On Windows 10: `Segoe Fluent Icons` is not included by default on Windows 10. You can download it here.
- On a Mac or other device: You can download `Segoe Fluent Icons` and other fonts here. You can download the font for use in design and development, but you may not ship it to another platform.

Examples

Open the WinUI 3 Gallery app and see Iconography principles in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Icon list

Please keep in mind that the `Segoe Fluent Icons` font includes many more icons than we can show here. Many of the icons are intended for specialized purposes and are not typically used anywhere else.

> ⓘ **Note**
>
> Glyphs with prefixes ranging from **E0-** to **E5-** (e.g. E001, E5B1) are currently marked as legacy and are therefore deprecated.

The following tables display all `Segoe Fluent Icons` glyphs and their respective unicode values and descriptive names. Select a range from the following list to view glyphs according to the PUA range they belong to.

- PUA E700-E900
- PUA EA00-EC00
- PUA ED00-EF00
- PUA F000-F200
- PUA F300-F500
- PUA F600-F800

PUA E700-E900

The following table of glyphs displays unicode points prefixed from E7- to E9-.

Back to top

Expand table

Glyph	Unicode point	Description
	e700	GlobalNavButton
	e701	Wifi
	e702	Bluetooth
	e703	Connect
	e704	InternetSharing
	e705	VPN
	e706	Brightness
	e707	MapPin
	e708	QuietHours
	e709	Airplane
	e70a	Tablet
	e70b	QuickNote
	e70c	RememberedDevice
	e70d	ChevronDown
	e70e	ChevronUp

	e70f	Edit
	e710	Add
	e711	Cancel
	e712	More
	e713	Settings
	e714	Video
	e715	Mail
	e716	People
	e717	Phone
	e718	Pin
	e719	Shop
	e71a	Stop
	e71b	Link
	e71c	Filter
	e71d	AllApps
	e71e	Zoom
	e71f	ZoomOut
	e720	Microphone
	e721	Search

	e722	Camera
	e723	Attach
	e724	Send
	e725	SendFill
	e726	WalkSolid
	e727	InPrivate
	e728	FavoriteList
	e729	PageSolid
	e72a	Forward
	e72b	Back
	e72c	Refresh
	e72d	Share
	e72e	Lock
	e72f	BlockedSite
	e730	ReportHacked
	e731	EMI
	e733	Blocked
	e734	FavoriteStar
	e735	FavoriteStarFill

	e736	ReadingMode
	e737	Favicon
	e738	Remove
	e739	Checkbox
	e73a	CheckboxComposite
	e73b	CheckboxFill
	e73c	CheckboxIndeterminate
	e73d	CheckboxCompositeReversed
	e73e	CheckMark
	e73f	BackToWindow
	e740	FullScreen
	e741	ResizeTouchLarger
	e742	ResizeTouchSmaller
	e743	ResizeMouseSmall
	e744	ResizeMouseMedium
	e745	ResizeMouseWide
	e746	ResizeMouseTall
	e747	ResizeMouseLarge
	e748	SwitchUser

	e749	Print
	e74a	Up
	e74b	Down
	e74c	OEM
	e74d	Delete
	e74e	Save
	e74f	Mute
	e750	BackSpaceQWERTY
	e751	ReturnKey
	e752	UpArrowShiftKey
	e753	Cloud
	e754	Flashlight
	e755	RotationLock
	e756	CommandPrompt
	e759	SIPMove
	e75a	SIPUndock
	e75b	SIPRedock
	e75c	EraseTool
	e75d	UnderscoreSpace

	e75e	GripperTool
	e75f	Dialpad
	e760	PageLeft
	e761	PageRight
	e762	MultiSelect
	e763	KeyboardLeftHanded
	e764	KeyboardRightHanded
	e765	KeyboardClassic
	e766	KeyboardSplit
	e767	Volume
	e768	Play
	e769	Pause
	e76b	ChevronLeft
	e76c	ChevronRight
	e76d	InkingTool
	e76e	Emoji2
	e76f	GripperBarHorizontal
	e770	System
	e771	Personalize

	e772	Devices
	e773	SearchAndApps
	e774	Globe
	e775	TimeLanguage
	e776	EaseOfAccess
	e777	UpdateRestore
	e778	HangUp
	e779	ContactInfo
	e77a	Unpin
	e77b	Contact
	e77c	Memo
	e77e	IncomingCall
	e77f	Paste
	e780	PhoneBook
	e781	LEDLight
	e783	Error
	e784	GripperBarVertical
	e785	Unlock
	e786	Slideshow

	e787	Calendar
	e788	GripperResize
	e789	Megaphone
	e78a	Trim
	e78b	NewWindow
	e78c	SaveLocal
	e790	Color
	e791	DataSense
	e792	SaveAs
	e793	Light
	e794	Effects
	e799	AspectRatio
	e7a1	Contrast
	e7a5	DataSenseBar
	e7a6	Redo
	e7a7	Undo
	e7a8	Crop
	e7aa	PhotoCollection
	e7ac	OpenWith

	e7ad	Rotate
	e7b3	RedEye
	e7b5	SetlockScreen
	e7b7	MapPin2
	e7b8	Package
	e7ba	Warning
	e7bc	ReadingList
	e7be	Education
	e7bf	ShoppingCart
	e7c0	Train
	e7c1	Flag
	e7c2	Move
	e7c3	Page
	e7c4	TaskView
	e7c5	BrowsePhotos
	e7c6	HalfStarLeft
	e7c7	HalfStarRight
	e7c8	Record
	e7c9	TouchPointer

	e7de	LangJPN
	e7e3	Ferry
	e7e6	Highlight
	e7e7	ActionCenterNotification
	e7e8	PowerButton
	e7ea	ResizeTouchNarrower
	e7eb	ResizeTouchShorter
	e7ec	DrivingMode
	e7ed	RingerSilent
	e7ee	OtherUser
	e7ef	Admin
	e7f0	CC
	e7f1	SDCard
	e7f2	CallForwarding
	e7f3	SettingsDisplaySound
	e7f4	TVMonitor
	e7f5	Speakers
	e7f6	Headphone
	e7f7	DeviceLaptopPic

	e7f8	DeviceLaptopNoPic
	e7f9	DeviceMonitorRightPic
	e7fa	DeviceMonitorLeftPic
	e7fb	DeviceMonitorNoPic
	e7fc	Game
	e7fd	HorizontalTabKey
	e802	StreetsideSplitMinimize
	e803	StreetsideSplitExpand
	e804	Car
	e805	Walk
	e806	Bus
	e809	TiltUp
	e80a	TiltDown
	e80b	CallControl
	e80c	RotateMapRight
	e80d	RotateMapLeft
	e80f	Home
	e811	ParkingLocation
	e812	MapCompassTop

	e813	MapCompassBottom
	e814	IncidentTriangle
	e815	Touch
	e816	MapDirections
	e819	StartPoint
	e81a	StopPoint
	e81b	EndPoint
	e81c	History
	e81d	Location
	e81e	MapLayers
	e81f	Accident
	e821	Work
	e822	Construction
	e823	Recent
	e825	Bank
	e826	DownloadMap
	e829	InkingToolFill2
	e82a	HighlightFill2
	e82b	EraseToolFill

	e82c	EraseToolFill2
	e82d	Dictionary
	e82e	DictionaryAdd
	e82f	ToolTip
	e830	ChromeBack
	e835	ProvisioningPackage
	e836	AddRemoteDevice
	e838	FolderOpen
	e839	Ethernet
	e83a	ShareBroadband
	e83b	DirectAccess
	e83c	DialUp
	e83d	DefenderApp
	e83e	BatteryCharging9
	e83f	Battery10
	e840	Pinned
	e841	PinFill
	e842	PinnedFill
	e843	PeriodKey

、。?!	e844	PuncKey
	e845	RevToggleKey
	e846	RightArrowKeyTime1
	e847	RightArrowKeyTime2
’	e848	LeftQuote
‘	e849	RightQuote
abc	e84a	DownShiftKey
ABC	e84b	UpShiftKey
， ” 。	e84c	PuncKey0
「」（）	e84d	PuncKeyLeftBottom
	e84e	RightArrowKeyTime3
	e84f	RightArrowKeyTime4
	e850	Battery0
	e851	Battery1
	e852	Battery2
	e853	Battery3
	e854	Battery4
	e855	Battery5
	e856	Battery6

	e857	Battery7
	e858	Battery8
	e859	Battery9
	e85a	BatteryCharging0
	e85b	BatteryCharging1
	e85c	BatteryCharging2
	e85d	BatteryCharging3
	e85e	BatteryCharging4
	e85f	BatteryCharging5
	e860	BatteryCharging6
	e861	BatteryCharging7
	e862	BatteryCharging8
	e863	BatterySaver0
	e864	BatterySaver1
	e865	BatterySaver2
	e866	BatterySaver3
	e867	BatterySaver4
	e868	BatterySaver5
	e869	BatterySaver6

	e86a	BatterySaver7
	e86b	BatterySaver8
	e86c	SignalBars1
	e86d	SignalBars2
	e86e	SignalBars3
	e86f	SignalBars4
	e870	SignalBars5
	e871	SignalNotConnected
	e872	Wifi1
	e873	Wifi2
	e874	Wifi3
	e875	MobSIMLock
	e876	MobSIMMissing
	e877	Vibrate
	e878	RoamingInternational
	e879	RoamingDomestic
	e87a	CallForwardInternational
	e87b	CallForwardRoaming
	e87c	JpnRomanji

	e87d	JpnRomanjiLock
	e87e	JpnRomanjiShift
	e87f	JpnRomanjiShiftLock
	e880	StatusDataTransfer
	e881	StatusDataTransferVPN
	e882	StatusDualSIM2
	e883	StatusDualSIM2VPN
	e884	StatusDualSIM1
	e885	StatusDualSIM1VPN
	e886	StatusSGLTE
	e887	StatusSGLTECell
	e888	StatusSGLTEDataVPN
	e889	StatusVPN
	e88a	WifiHotspot
	e88b	LanguageKor
	e88c	LanguageCht
	e88d	LanguageChs
	e88e	USB
	e88f	InkingToolFill

	e890	View
	e891	HighlightFill
	e892	Previous
	e893	Next
	e894	Clear
	e895	Sync
	e896	Download
	e897	Help
	e898	Upload
	e899	Emoji
	e89a	TwoPage
	e89b	LeaveChat
	e89c	MailForward
	e89e	RotateCamera
	e89f	ClosePane
	e8a0	OpenPane
	e8a1	PreviewLink
	e8a2	AttachCamera
	e8a3	ZoomIn

	e8a4	Bookmarks
	e8a5	Document
	e8a6	ProtectedDocument
	e8a7	OpenInNewWindow
	e8a8	MailFill
	e8a9	ViewAll
	e8aa	VideoChat
	e8ab	Switch
	e8ac	Rename
	e8ad	Go
	e8ae	SurfaceHub
	e8af	Remote
	e8b0	Click
	e8b1	Shuffle
	e8b2	Movies
	e8b3	SelectAll
	e8b4	Orientation
	e8b5	Import
	e8b6	ImportAll

	e8b7	Folder
	e8b8	Webcam
	e8b9	Picture
	e8ba	Caption
	e8bb	ChromeClose
	e8bc	ShowResults
	e8bd	Message
	e8be	Leaf
	e8bf	CalendarDay
	e8c0	CalendarWeek
	e8c1	Characters
	e8c2	MailReplyAll
	e8c3	Read
	e8c4	ShowBcc
	e8c5	HideBcc
	e8c6	Cut
	e8c7	PaymentCard
	e8c8	Copy
	e8c9	Important

	e8ca	MailReply
	e8cb	Sort
	e8cc	MobileTablet
	e8cd	DisconnectDrive
	e8ce	MapDrive
	e8cf	ContactPresence
	e8d0	Priority
	e8d1	GotoToday
	e8d2	Font
	e8d3	FontColor
	e8d4	Contact2
	e8d5	FolderFill
	e8d6	Audio
	e8d7	Permissions
	e8d8	DisableUpdates
	e8d9	Unfavorite
	e8da	OpenLocal
	e8db	Italic
	e8dc	Underline

	e8dd	Bold
	e8de	MoveToFolder
	e8df	LikeDislike
	e8e0	Dislike
	e8e1	Like
	e8e2	AlignRight
	e8e3	AlignCenter
	e8e4	AlignLeft
	e8e5	OpenFile
	e8e6	ClearSelection
	e8e7	FontDecrease
	e8e8	FontIncrease
	e8e9	FontSize
	e8ea	CellPhone
	e8eb	Reshare
	e8ec	Tag
	e8ed	RepeatOne
	e8ee	RepeatAll
	e8ef	Calculator

	e8f0	Directions
	e8f1	Library
	e8f2	ChatBubbles
	e8f3	PostUpdate
	e8f4	NewFolder
	e8f5	CalendarReply
	e8f6	UnsyncFolder
	e8f7	SyncFolder
	e8f8	BlockContact
	e8f9	SwitchApps
	e8fa	AddFriend
	e8fb	Accept
	e8fc	GoToStart
	e8fd	BulletedList
	e8fe	Scan
	e8ff	Preview
	e902	Group
	e904	ZeroBars
	e905	OneBar

	e906	TwoBars
	e907	ThreeBars
	e908	FourBars
	e909	World
	e90a	Comment
	e90b	MusicInfo
	e90c	DockLeft
	e90d	DockRight
	e90e	DockBottom
	e90f	Repair
	e910	Accounts
	e911	DullSound
	e912	Manage
	e913	Street
	e914	Printer3D
	e915	RadioBullet
	e916	Stopwatch
	e91b	Photo
	e91c	ActionCenter

	e91f	FullCircleMask
	e921	ChromeMinimize
	e922	ChromeMaximize
	e923	ChromeRestore
	e924	Annotation
	e925	BackSpaceQWERTYSm
	e926	BackSpaceQWERTYMd
	e927	Swipe
	e928	Fingerprint
	e929	Handwriting
	e92c	ChromeBackToWindow
	e92d	ChromeFullScreen
	e92e	KeyboardStandard
	e92f	KeyboardDismiss
	e930	Completed
	e931	ChromeAnnotate
	e932	Label
	e933	IBeam
	e934	IBeamOutline

	e935	FlickDown
	e936	FlickUp
	e937	FlickLeft
	e938	FlickRight
	e939	FeedbackApp
	e93c	MusicAlbum
	e93e	Streaming
	e943	Code
	e944	ReturnToWindow
	e945	LightningBolt
	e946	Info
	e947	CalculatorMultiply
	e948	CalculatorAddition
	e949	CalculatorSubtract
	e94a	CalculatorDivide
	e94b	CalculatorSquareroot
	e94c	CalculatorPercentage
	e94d	CalculatorNegate
	e94e	CalculatorEqualTo

	e94f	CalculatorBackspace
	e950	Component
	e951	DMC
	e952	Dock
	e953	MultimediaDMS
	e954	MultimediaDVR
	e955	MultimediaPMP
	e956	PrintfaxPrinterFile
	e957	Sensor
	e958	StorageOptical
	e95a	Communications
	e95b	Headset
	e95d	Projector
	e95e	Health
	e95f	Wire
	e960	Webcam2
	e961	Input
	e962	Mouse
	e963	Smartcard

	e964	SmartcardVirtual
	e965	MediaStorageTower
	e966	ReturnKeySm
	e967	GameConsole
	e968	Network
	e969	StorageNetworkWireless
	e96a	StorageTape
	e96d	ChevronUpSmall
	e96e	ChevronDownSmall
	e96f	ChevronLeftSmall
	e970	ChevronRightSmall
	e971	ChevronUpMed
	e972	ChevronDownMed
	e973	ChevronLeftMed
	e974	ChevronRightMed
	e975	Devices2
	e976	ExpandTile
	e977	PC1
	e978	PresenceChicklet

	e979	PresenceChickletVideo
	e97a	Reply
	e97b	SetTile
	e97c	Type
	e97d	Korean
	e97e	HalfAlpha
	e97f	FullAlpha
	e980	Key12On
	e981	ChineseChangjie
	e982	QWERTYOn
	e983	QWERTYOff
	e984	ChineseQuick
	e985	Japanese
	e986	FullHiragana
	e987	FullKatakana
	e988	HalfKatakana
	e989	ChineseBoPoMoFo
	e98a	ChinesePinyin
	e98f	ConstructionCone

	e990	XboxOneConsole
	e992	Volume0
	e993	Volume1
	e994	Volume2
	e995	Volume3
	e996	BatteryUnknown
	e998	WifiAttentionOverlay
	e99a	Robot
	e9a1	TapAndSend
	e9a4	TextBulletListSquare
	e9a6	FitPage
	e9a8	PasswordKeyShow
	e9a9	PasswordKeyHide
	e9aa	BidiLtr
	e9ab	BidiRtl
	e9ac	ForwardSm
	e9ad	CommaKey
	e9ae	DashKey
	e9af	DullSoundKey

	e9b0	HalfDullSound
	e9b1	RightDoubleQuote
	e9b2	LeftDoubleQuote
	e9b3	PuncKeyRightBottom
	e9b4	PuncKey1
	e9b5	PuncKey2
	e9b6	PuncKey3
	e9b7	PuncKey4
	e9b8	PuncKey5
	e9b9	PuncKey6
	e9ba	PuncKey9
	e9bb	PuncKey7
	e9bc	PuncKey8
	e9ca	Frigid
	e9ce	Unknown
	e9d2	AreaChart
	e9d5	CheckList
	e9d9	Diagnostic
	e9e9	Equalizer

	e9f3	Process
	e9f5	Processing
	e9f9	ReportDocument

PUA EA00-EC00

The following table of glyphs displays unicode points prefixed from EA- to EC-.

Back to top

Expand table

Glyph	Unicode point	Description
	ea0c	VideoSolid
	ea0d	MixedMediaBadge
	ea14	DisconnectDisplay
	ea18	Shield
	ea1f	Info2
	ea21	ActionCenterAsterisk
	ea24	Beta
	ea35	SaveCopy
	ea37	List
	ea38	Asterisk
	ea39	ErrorBadge

	ea3a	CircleRing
	ea3b	CircleFill
	ea3c	MergeCall
	ea3d	PrivateCall
	ea3f	Record2
	ea40	AllAppsMirrored
	ea41	BookmarksMirrored
	ea42	BulletedListMirrored
	ea43	CallForwardInternationalMirrored
	ea44	CallForwardRoamingMirrored
	ea47	ChromeBackMirrored
	ea48	ClearSelectionMirrored
	ea49	ClosePaneMirrored
	ea4a	ContactInfoMirrored
	ea4b	DockRightMirrored
	ea4c	DockLeftMirrored
	ea4e	ExpandTileMirrored
	ea4f	GoMirrored
	ea50	GripperResizeMirrored

	ea51	HelpMirrored
	ea52	ImportMirrored
	ea53	ImportAllMirrored
	ea54	LeaveChatMirrored
	ea55	ListMirrored
	ea56	MailForwardMirrored
	ea57	MailReplyMirrored
	ea58	MailReplyAllMirrored
	ea5b	OpenPaneMirrored
	ea5c	OpenWithMirrored
	ea5e	ParkingLocationMirrored
	ea5f	ResizeMouseMediumMirrored
	ea60	ResizeMouseSmallMirrored
	ea61	ResizeMouseTallMirrored
	ea62	ResizeTouchNarrowerMirrored
	ea63	SendMirrored
	ea64	SendFillMirrored
	ea65	ShowResultsMirrored
	ea69	Media

	ea6a	SyncError
	ea6c	Devices3
	ea79	SlowMotionOn
	ea80	Lightbulb
	ea81	StatusCircle
	ea82	StatusTriangle
	ea83	StatusError
	ea84	StatusWarning
	ea85	VolumeDisabled
	ea86	Puzzle
	ea89	CalendarSolid
	ea8a	HomeSolid
	ea8b	ParkingLocationSolid
	ea8c	ContactSolid
	ea8d	ConstructionSolid
	ea8e	AccidentSolid
	ea8f	Ringer
	ea90	PDF
	ea91	ThoughtBubble

	ea92	HeartBroken
	ea93	BatteryCharging10
	ea93	BatteryCharging10
	ea94	BatterySaver9
	ea95	BatterySaver10
	ea97	CallForwardingMirrored
	ea98	MultiSelectMirrored
	ea99	Broom
	eac2	ForwardCall
	eac7	DesktopLeafTwo
	ead4	Emojiplay
	ead5	EmojiBrush
	ead6	EyeTracking
	ead7	EyeTrackingText
	eadf	Trackers
	eafc	Market
	eb05	PieSingle
	eb0f	StockUp
	eb11	StockDown

	eb19	ClicktoDoOff
	eb1d	ClicktoDo
	eb3b	GenericApp
	eb3c	Design
	eb41	Website
	eb42	Drop
	eb44	Radar
	eb47	BusSolid
	eb48	FerrySolid
	eb49	StartPointSolid
	eb4a	StopPointSolid
	eb4b	EndPointSolid
	eb4c	AirplaneSolid
	eb4d	TrainSolid
	eb4e	WorkSolid
	eb4f	ReminderFill
	eb50	Reminder
	eb51	Heart
	eb52	HeartFill

	eb55	EthernetError
	eb56	EthernetWarning
	eb57	StatusConnecting1
	eb58	StatusConnecting2
	eb59	StatusUnsecure
	eb5a	WifiError0
	eb5b	WifiError1
	eb5c	WifiError2
	eb5d	WifiError3
	eb5e	WifiError4
	eb5f	WifiWarning0
	eb60	WifiWarning1
	eb61	WifiWarning2
	eb62	WifiWarning3
	eb63	WifiWarning4
	eb66	Devices4
	eb67	NUIIris
	eb68	NUIFace
	eb77	GatewayRouter

	eb7e	EditMirrored
	eb82	NUIFPStartSlideHand
	eb83	NUIFPStartSlideAction
	eb84	NUIFPContinueSlideHand
	eb85	NUIFPContinueSlideAction
	eb86	NUIFPRollRightHand
	eb87	NUIFPRollRightHandAction
	eb88	NUIFPRollLeftHand
	eb89	NUIFPRollLeftAction
	eb8a	NUIFPPressHand
	eb8b	NUIFPPressAction
	eb8c	NUIFPPressRepeatHand
	eb8d	NUIFPPressRepeatAction
	eb90	StatusErrorFull
	eb91	TaskViewExpanded
	eb95	Certificate
	eb96	BackSpaceQWERTYLg
	eb97	ReturnKeyLg
	eb9d	FastForward

	eb9e	Rewind
	eb9f	Photo2
	eba0	MobBattery0
	eba1	MobBattery1
	eba2	MobBattery2
	eba3	MobBattery3
	eba4	MobBattery4
	eba5	MobBattery5
	eba6	MobBattery6
	eba7	MobBattery7
	eba8	MobBattery8
	eba9	MobBattery9
	ebaa	MobBattery10
	ebab	MobBatteryCharging0
	ebac	MobBatteryCharging1
	ebad	MobBatteryCharging2
	ebae	MobBatteryCharging3
	ebaf	MobBatteryCharging4
	ebb0	MobBatteryCharging5

	ebb1	MobBatteryCharging6
	ebb2	MobBatteryCharging7
	ebb3	MobBatteryCharging8
	ebb4	MobBatteryCharging9
	ebb5	MobBatteryCharging10
	ebb6	MobBatterySaver0
	ebb7	MobBatterySaver1
	ebb8	MobBatterySaver2
	ebb9	MobBatterySaver3
	ebba	MobBatterySaver4
	ebbb	MobBatterySaver5
	ebbc	MobBatterySaver6
	ebbd	MobBatterySaver7
	ebbe	MobBatterySaver8
	ebbf	MobBatterySaver9
	ebc0	MobBatterySaver10
	ebc3	DictionaryCloud
	ebc4	ResetDrive
	ebc5	VolumeBars

	ebc6	Project
	ebd2	AdjustHologram
	ebd3	CloudDownload
	ebd4	MobWifiCallBars
	ebd5	MobWifiCall0
	ebd6	MobWifiCall1
	ebd7	MobWifiCall2
	ebd8	MobWifiCall3
	ebd9	MobWifiCall4
	ebda	Family
	ebdb	LockFeedback
	ebde	DeviceDiscovery
	ebe6	WindDirection
	ebe7	RightArrowKeyTime0
	ebe8	Bug
	ebfc	TabletMode
	ebfd	StatusCircleLeft
	ebfe	StatusTriangleLeft
	ebff	StatusErrorLeft

	ec00	StatusWarningLeft
	ec02	MobBatteryUnknown
	ec05	NetworkTower
	ec06	CityNext
	ec07	CityNext2
	ec08	Courthouse
	ec09	Groceries
	ec0a	Sustainable
	ec0b	BuildingEnergy
	ec11	ToggleFilled
	ec12	ToggleBorder
	ec13	SliderThumb
	ec14	ToggleThumb
	ec15	MiracastLogoSmall
	ec16	MiracastLogoLarge
	ec19	PLAP
	ec1b	Badge
	ec1e	SignalRoaming
	ec20	MobileLocked

	ec24	InsiderHubApp
	ec25	PersonalFolder
	ec26	HomeGroup
	ec27	MyNetwork
	ec31	KeyboardFull
	ec32	Cafe
	ec34	FormatText
	ec37	MobSignal1
	ec38	MobSignal2
	ec39	MobSignal3
	ec3a	MobSignal4
	ec3b	MobSignal5
	ec3c	MobWifi1
	ec3d	MobWifi2
	ec3e	MobWifi3
	ec3f	MobWifi4
	ec40	MobAirplane
	ec41	MobBluetooth
	ec42	MobActionCenter

	ec43	MobLocation
	ec44	MobWifiHotspot
	ec45	LanguageJpn
	ec46	MobQuietHours
	ec47	MobDrivingMode
	ec48	SpeedOff
	ec49	SpeedMedium
	ec4a	SpeedHigh
	ec4e	ThisPC
	ec4f	MusicNote
	ec50	FileExplorer
	ec51	FileExplorerApp
	ec52	LeftArrowKeyTime0
	ec54	MicOff
	ec55	MicSleep
	ec56	MicError
	ec57	PlaybackRate1x
	ec58	PlaybackRateOther
	ec59	CashDrawer

	ec5a	BarcodeScanner
	ec5b	ReceiptPrinter
	ec5c	MagStripeReader
	ec61	CompletedSolid
	ec64	CompanionApp
	ec6c	Favicon2
	ec6d	SwipeRevealArt
	ec71	MicOn
	ec72	MicClipping
	ec74	TabletSelected
	ec75	MobileSelected
	ec76	LaptopSelected
	ec77	TVMonitorSelected
	ec7a	DeveloperTools
	ec7e	MobCallForwarding
	ec7f	MobCallForwardingMirrored
	ec80	BodyCam
	ec81	PoliceCar
	ec83	UpdateStatusDot2

	ec87	Draw
	ec88	DrawSolid
	ec8a	LowerBrightness
	ec8f	ScrollUpDown
	ec91	Uninstall
	ec92	DateTime
	ec94	HoloLens
	ec9c	CloudNotSynced
	eca5	Tiles
	eca7	PartyLeader
	ecaa	AppIconDefault
	ecad	Calories
	ecaf	POI
	ecb9	BandBattery0
	ecba	BandBattery1
	ecbb	BandBattery2
	ecbc	BandBattery3
	ecbd	BandBattery4
	ecbe	BandBattery5

	ecbf	BandBattery6
	ecc4	AddSurfaceHub
	ecc5	DevUpdate
	ecc6	Unit
	ecc8	AddTo
	ecc9	RemoveFrom
	ecca	RadioBtnOff
	eccb	RadioBtnOn
	eccc	RadioBullet2
	eccd	ExploreContent
	ece4	Blocked2
	ece7	ScrollMode
	ece8	ZoomMode
	ece9	PanMode
	ecf0	WiredUSB
	ecf1	WirelessUSB
	ecf3	USBSafeConnect

PUA ED00-EF00

The following table of glyphs displays unicode points prefixed from ED- to EF-.

Back to top

Expand table

Glyph	Unicode point	Description
	ed0c	ActionCenterNotificationMirrored
	ed0d	ActionCenterMirrored
	ed0e	SubscriptionAdd
	ed10	ResetDevice
	ed11	SubscriptionAddMirrored
	ed14	QRCode
	ed15	Feedback
	ed1a	Hide
	ed1e	Subtitles
	ed1f	SubtitlesAudio
	ed21	RestartUpdate2
	ed25	OpenFolderHorizontal
	ed28	CalendarMirrored
	ed2a	MobeSIM
	ed2b	MobeSIMNoProfile
	ed2c	MobeSIMLocked

	ed2d	MobeSIMBusy
	ed2e	SignalError
	ed2f	StreamingEnterprise
	ed30	Headphone0
	ed31	Headphone1
	ed32	Headphone2
	ed33	Headphone3
	ed35	Apps
	ed39	KeyboardBrightness
	ed3a	KeyboardLowerBrightness
	ed3c	SkipBack10
	ed3d	SkipForward30
	ed41	TreeFolderFolder
	ed42	TreeFolderFolderFill
	ed43	TreeFolderFolderOpen
	ed44	TreeFolderFolderOpenFill
	ed47	MultimediaDMP
	ed4c	KeyboardOneHanded
	ed4d	Narrator

	ed53	EmojiTabPeople
	ed54	EmojiTabSmilesAnimals
	ed55	EmojiTabCelebrationObjects
	ed56	EmojiTabFoodPlants
	ed57	EmojiTabTransitPlaces
	ed58	EmojiTabSymbols
	ed59	EmojiTabTextSmiles
	ed5a	EmojiTabFavorites
	ed5b	EmojiSwatch
	ed5c	ConnectApp
	ed5d	CompanionDeviceFramework
	ed5e	Ruler
	ed5f	FingerInking
	ed60	StrokeErase
	ed61	PointErase
	ed62	ClearAllInk
	ed63	Pencil
	ed64	Marker
	ed65	InkingCaret

	ed66	InkingColorOutline
	ed67	InkingColorFill
	eda2	HardDrive
	eda3	NetworkAdapter
	eda4	Touchscreen
	eda5	NetworkPrinter
	eda6	CloudPrinter
	eda7	KeyboardShortcut
	eda8	BrushSize
	eda9	NarratorForward
	edaa	NarratorForwardMirrored
	edab	SyncBadge12
	edac	RingerBadge12
	edad	AsteriskBadge12
	edae	ErrorBadge12
	edaf	CircleRingBadge12
	edb0	CircleFillBadge12
	edb1	ImportantBadge12
	edb3	MailBadge12

	edb4	PauseBadge12
	edb5	PlayBadge12
	edc6	PenWorkspace
	edd5	CaretLeft8
	edd6	CaretRight8
	edd7	CaretUp8
	edd8	CaretDown8
	edd9	CaretLeftSolid8
	edda	CaretRightSolid8
	eddb	CaretUpSolid8
	eddc	CaretDownSolid8
	ede0	Strikethrough
	ede1	Export
	ede2	ExportMirrored
	ede3	ButtonMenu
	ede4	CloudSearch
	ede5	PinyinIMELogo
	edfb	CalligraphyPen
	ee35	ReplyMirrored

	ee3f	LockscreenDesktop
	ee40	TaskViewSettings
	ee41	FullHiraganaPrivateMode
	ee42	FullKatakanaPrivateMode
	ee43	HalfAlphaPrivateMode
	ee44	HalfKatakanaPrivateMode
	ee45	FullAlphaPrivateMode
	ee47	MiniExpand2Mirrored
	ee49	MiniContract2Mirrored
	ee4a	Play36
	ee56	PenPalette
	ee57	GuestUser
	ee63	SettingsBattery
	ee64	TaskbarPhone
	ee65	LockScreenGlance
	ee6f	GenericScan
	ee71	ImageExport
	ee77	WifiEthernet
	ee79	ActionCenterQuiet

	ee7a	ActionCenterQuietNotification
	ee7e	FIDOPasskey
	ee92	TrackersMirrored
	ee93	DateTimeMirrored
	ee94	Wheel
	ee95	StopSolid
	eea0	RAM
	eea1	CPU
	eea3	VirtualMachineGroup
	eeca	ButtonView2
	ef15	PenWorkspaceMirrored
	ef16	PenPaletteMirrored
	ef17	StrokeEraseMirrored
	ef18	PointEraseMirrored
	ef19	ClearAllInkMirrored
	ef1f	BackgroundToggle
	ef20	Marquee
	ef2c	ChromeCloseContrast
	ef2d	ChromeMinimizeContrast

	ef2e	ChromeMaximizeContrast
	ef2f	ChromeRestoreContrast
	ef31	TrafficLight
	ef3b	Replay
	ef3c	Eyedropper
	ef3d	LineDisplay
	ef3e	PINPad
	ef3f	SignatureCapture
	ef40	ChipCardCreditCardReader
	ef42	MarketDown
	ef58	PlayerSettings
	ef60	TextEdit
	ef6b	LandscapeOrientation
	ef90	Flow
	efa5	Touchpad
	efa9	Speech
	efda	AppIconDefaultAdd
	efff	CRMScheduleReports

PUA F000-F200

The following table of glyphs displays unicode points prefixed from F0- to F2-.

Back to top

Expand table

Glyph	Unicode point	Description
	f000	KnowledgeArticle
	f003	Relationship
	f012	ZipFolder
	f080	DefaultAPN
	f081	UserAPN
	f085	DoublePinyin
	f08c	BlueLight
	f08d	CaretSolidLeft
	f08e	CaretSolidDown
	f08f	CaretSolidRight
	f090	CaretSolidUp
	f093	ButtonA
	f094	ButtonB
	f095	ButtonY
	f096	ButtonX

	f0ad	ArrowUp8
	f0ae	ArrowDown8
	f0af	ArrowRight8
	f0b0	ArrowLeft8
	f0b2	QuarentinedItems
	f0b3	QuarentinedItemsMirrored
	f0b4	Protractor
	f0b5	ChecklistMirrored
	f0b6	StatusCircle7
	f0b7	StatusCheckmark7
	f0b8	StatusErrorCircle7
	f0b9	Connected
	f0c6	PencilFill
	f0c7	CalligraphyFill
	f0ca	QuarterStarLeft
	f0cb	QuarterStarRight
	f0cc	ThreeQuarterStarLeft
	f0cd	ThreeQuarterStarRight
	f0ce	QuietHoursBadge12

	f0d2	BackMirrored
	f0d3	ForwardMirrored
	f0d5	ChromeBackContrast
	f0d6	ChromeBackContrastMirrored
	f0d7	ChromeBackToWindowContrast
	f0d8	ChromeFullScreenContrast
	f0e2	GridView
	f0e3	ClipboardList
	f0e4	ClipboardListMirrored
	f0e5	OutlineQuarterStarLeft
	f0e6	OutlineQuarterStarRight
	f0e7	OutlineHalfStarLeft
	f0e8	OutlineHalfStarRight
	f0e9	OutlineThreeQuarterStarLeft
	f0ea	OutlineThreeQuarterStarRight
	f0eb	SpatialVolume0
	f0ec	SpatialVolume1
	f0ed	SpatialVolume2
	f0ee	SpatialVolume3

	f0ef	ApplicationGuard
	f0f7	OutlineStarLeftHalf
	f0f8	OutlineStarRightHalf
	f0f9	ChromeAnnotateContrast
	f0fb	DefenderBadge12
	f103	DetachablePC
	f108	LeftStick
	f109	RightStick
	f10a	TriggerLeft
	f10b	TriggerRight
	f10c	BumperLeft
	f10d	BumperRight
	f10e	Dpad
	f110	EnglishPunctuation
	f111	ChinesePunctuation
	f119	HMD
	f11b	CtrlSpatialRight
	f126	PaginationDotOutline10
	f127	PaginationDotSolid10

	f128	StrokeErase2
	f129	SmallErase
	f12a	LargeErase
	f12b	FolderHorizontal
	f12e	MicrophoneListening
	f12f	StatusExclamationCircle7
	f131	Video360
	f133	GiftboxOpen
	f136	StatusCircleOuter
	f137	StatusCircleInner
	f138	StatusCircleRing
	f139	StatusTriangleOuter
	f13a	StatusTriangleInner
	f13b	StatusTriangleExclamation
	f13c	StatusCircleExclamation
	f13d	StatusCircleErrorX
	f13e	StatusCircleCheckmark
	f13f	StatusCircleInfo
	f140	StatusCircleBlock

–	f141	StatusCircleBlock2
?	f142	StatusCircleQuestionMark
⮔	f143	StatusCircleSync
1	f146	Dial1
2	f147	Dial2
3	f148	Dial3
4	f149	Dial4
5	f14a	Dial5
6	f14b	Dial6
7	f14c	Dial7
8	f14d	Dial8
9	f14e	Dial9
10	f14f	Dial10
11	f150	Dial11
12	f151	Dial12
13	f152	Dial13
14	f153	Dial14
15	f154	Dial15
16	f155	Dial16

	f156	DialShape1
	f157	DialShape2
	f158	DialShape3
	f159	DialShape4
	f15f	ClosedCaptionsInternational
	f161	TollSolid
	f163	TrafficCongestionSolid
	f164	ExploreContentSingle
	f165	CollapseContent
	f166	CollapseContentSingle
	f167	InfoSolid
	f168	GroupList
	f169	CaretBottomRightSolidCenter8
	f16a	ProgressRingDots
	f16b	Checkbox14
	f16c	CheckboxComposite14
	f16d	CheckboxIndeterminateCombo14
	f16e	CheckboxIndeterminateCombo
	f175	StatusPause7

	f17f	CharacterAppearance
	f180	Lexicon
	f182	ScreenTime
	f191	HeadlessDevice
	f193	NetworkSharing
	f19d	EyeGaze
	f19e	ToggleLeft
	f19f	ToggleRight
	f1ad	WindowsInsider
	f1cb	ChromeSwitch
	f1cc	ChromeSwitchContast
	f1d8	StatusCheckmark
	f1d9	StatusCheckmarkLeft
	f20c	KeyboardLeftAligned
	f20d	KeyboardRightAligned
	f210	KeyboardSettings
	f211	NetworkPhysical
	f22c	IOT
	f22e	UnknownMirrored

	f246	ViewDashboard
	f259	ExploitProtectionSettings
	f260	KeyboardNarrow
	f261	Keyboard12Key
	f26b	KeyboardDock
	f26c	KeyboardUndock
	f26d	KeyboardLeftDock
	f26e	KeyboardRightDock
	f270	Ear
	f271	PointerHand
	f272	Bullseye
	f28b	DocumentApproval
	f2b7	LocaleLanguage

PUA F300-F500

The following table of glyphs displays unicode points prefixed from F3- to F5-.

Back to top

Expand table

Glyph	Unicode point	Description
	f112	ReadOutLoud

	f117	ProjectToDevice
	f120	TaskManagerApp
	f191	HeadlessDevice
	f196	Beaker
	f1ad	WindowsInsider
	f1b1	PowerButtonUpdate2
	f1e8	LeafTwo
	f232	GridViewSmall
	f27c	Earbudsingle
	f27f	HearingAid
	f285	MobSnooze
	f2a3	MobNotificationBell
	f2a5	MobNotificationBellFilled
	f2a8	MobSnoozeFilled
	f2c7	BulletedList2
	f2c8	BulletedList2Mirrored
	f2d9	CirclePause
	f32a	PassiveAuthentication
	f354	ColorSolid

	f384	NetworkOffline
	f385	NetworkConnected
	f386	NetworkConnectedCheckmark
	f3b1	SignOut
	f3cc	StatusInfo
	f3cd	StatusInfoLeft
	f3e2	NearbySharing
	f3e7	CtrlSpatialLeft
	f404	InteractiveDashboard
	f405	DeclineCall
	f406	ClippingTool
	f407	RectangularClipping
	f408	FreeFormClipping
	f413	CopyTo
	f427	IDBadge
	f432	BatterySaver
	f439	DynamicLock
	f45e	PenTips
	f45f	PenTipsMirrored

	f460	HWPJoin
	f461	HWPInsert
	f462	HWPStrikeThrough
	f463	HWPScratchOut
	f464	HWPSplit
	f465	HWPNewLine
	f466	HWPOverwrite
	f473	MobWifiWarning1
	f474	MobWifiWarning2
	f475	MobWifiWarning3
	f476	MobWifiWarning4
	f47f	MicLocationCombo
	f49a	Globe2
	f4a5	SpecialEffectSize
	f4a9	GIF
	f4aa	Sticker2
	f4bd	Snooze
	f4be	SurfaceHubSelected
	f4bf	HoloLensSelected

	f4c0	Earbud
	f4c3	MixVolumes
	f540	Safe
	f552	LaptopSecure
	f56d	PrintDefault
	f56e	PageMirrored
	f56f	LandscapeOrientationMirrored
	f570	ColorOff
	f571	PrintAllPages
	f572	PrintCustomRange
	f573	PageMarginPortraitNarrow
	f574	PageMarginPortraitNormal
	f575	PageMarginPortraitModerate
	f576	PageMarginPortraitWide
	f577	PageMarginLandscapeNarrow
	f578	PageMarginLandscapeNormal
	f579	PageMarginLandscapeModerate
	f57a	PageMarginLandscapeWide
	f57b	CollateLandscape

	f57c	CollatePortrait
	f57d	CollatePortraitSeparated
	f57e	DuplexLandscapeOneSided
	f57f	DuplexLandscapeOneSidedMirrored
	f580	DuplexLandscapeTwoSidedLongEdge
	f581	DuplexLandscapeTwoSidedLongEdgeMirrored
	f582	DuplexLandscapeTwoSidedShortEdge
	f583	DuplexLandscapeTwoSidedShortEdgeMirrored
	f584	DuplexPortraitOneSided
	f585	DuplexPortraitOneSidedMirrored
	f586	DuplexPortraitTwoSidedLongEdge
	f587	DuplexPortraitTwoSidedLongEdgeMirrored
	f588	DuplexPortraitTwoSidedShortEdge
	f589	DuplexPortraitTwoSidedShortEdgeMirrored
	f58a	PPSOneLandscape
	f58b	PPSTwoLandscape
	f58c	PPSTwoPortrait
	f58d	PPSFourLandscape
	f58e	PPSFourPortrait

	f58f	HolePunchOff
	f590	HolePunchPortraitLeft
	f591	HolePunchPortraitRight
	f592	HolePunchPortraitTop
	f593	HolePunchPortraitBottom
	f594	HolePunchLandscapeLeft
	f595	HolePunchLandscapeRight
	f596	HolePunchLandscapeTop
	f597	HolePunchLandscapeBottom
	f598	StaplingOff
	f599	StaplingPortraitTopLeft
	f59a	StaplingPortraitTopRight
	f59b	StaplingPortraitBottomRight
	f59c	StaplingPortraitTwoLeft
	f59d	StaplingPortraitTwoRight
	f59e	StaplingPortraitTwoTop
	f59f	StaplingPortraitTwoBottom
	f5a0	StaplingPortraitBookBinding
	f5a1	StaplingLandscapeTopLeft

	f5a2	StaplingLandscapeTopRight
	f5a3	StaplingLandscapeBottomLeft
	f5a4	StaplingLandscapeBottomRight
	f5a5	StaplingLandscapeTwoLeft
	f5a6	StaplingLandscapeTwoRight
	f5a7	StaplingLandscapeTwoTop
	f5a8	StaplingLandscapeTwoBottom
	f5a9	StaplingLandscapeBookBinding
	f5aa	StatusDataTransferRoaming
	f5ab	MobSIMError
	f5ac	CollateLandscapeSeparated
	f5ad	PPSOnePortrait
	f5ae	StaplingPortraitBottomLeft
	f5b0	PlaySolid
	f5e7	RepeatOff
	f5ed	Set
	f5ee	SetSolid
	f5ef	FuzzyReading
	f5f2	VerticalBattery0

	f5f3	VerticalBattery1
	f5f4	VerticalBattery2
	f5f5	VerticalBattery3
	f5f6	VerticalBattery4
	f5f7	VerticalBattery5
	f5f8	VerticalBattery6
	f5f9	VerticalBattery7
	f5fa	VerticalBattery8
	f5fb	VerticalBattery9
	f5fc	VerticalBattery10
	f5fd	VerticalBatteryCharging0
	f5fe	VerticalBatteryCharging1
	f5ff	VerticalBatteryCharging2

PUA F600-F800

The following table of glyphs displays unicode points prefixed from F6- to F8-.

Back to top

Expand table

Glyph	Unicode point	Description
	f600	VerticalBatteryCharging3

	f601	VerticalBatteryCharging4
	f602	VerticalBatteryCharging5
	f603	VerticalBatteryCharging6
	f604	VerticalBatteryCharging7
	f605	VerticalBatteryCharging8
	f606	VerticalBatteryCharging9
	f607	VerticalBatteryCharging10
	f608	VerticalBatteryUnknown
	f618	SIMError
	f619	SIMMissing
	f61a	SIMLock
	f61b	eSIM
	f61c	eSIMNoProfile
	f61d	eSIMLocked
	f61e	eSIMBusy
	f61f	NoiseCancelation
	f620	NoiseCancelationOff
	f623	MusicSharing
	f624	MusicSharingOff

	f63c	CircleShapeSolid
	f657	WifiCallBars
	f658	WifiCall0
	f659	WifiCall1
	f65a	WifiCall2
	f65b	WifiCall3
	f65c	WifiCall4
	f67b	Pen
	f683	TextSelect
	f684	TextNavigate
	f698	PinyinIMELogo2
	f69b	UserRemove
	f69e	CHTLanguageBar
	f6a9	ComposeMode
	f6b8	ExpressiveInputEntry
	f6ba	EmojiTabMoreSymbols
	f6c4	PhoneScreen
	f6c5	AlertUrgent
	f6c6	PhoneDesktop

	f6fa	WebSearch
	f712	Kiosk
	f714	RTTLogo
	f715	VoiceCall
	f716	GoToMessage
	f71a	ReturnToCall
	f71c	StartPresenting
	f71d	StopPresenting
	f71e	ProductivityMode
	f738	SetHistoryStatus
	f739	SetHistoryStatus2
	f73d	Keyboardsettings20
	f73e	OneHandedRight20
	f73f	OneHandedLeft20
	f740	Split20
	f741	Full20
	f742	Handwriting20
	f743	ChevronLeft20
	f744	ChevronLeft32

	f745	ChevronRight20
	f746	ChevronRight32
	f763	Event12
	f781	MicOff2
	f785	DeliveryOptimization
	f78a	CancelMedium
	f78b	SearchMedium
	f78c	AcceptMedium
	f78d	RevealPasswordMedium
	f7ad	DeleteWord
	f7ae	DeleteWordFill
	f7af	DeleteLines
	f7b0	DeleteLinesFill
	f7b1	InstertWords
	f7b2	InstertWordsFill
	f7b3	JoinWords
	f7b4	JoinWordsFill
	f7b5	OverwriteWords
	f7b6	OverwriteWordsFill

	f7b7	AddNewLine
	f7b8	AddNewLineFill
	f7b9	OverwriteWordsKorean
	f7ba	OverwriteWordsFillKorean
	f7bb	EducationIcon
	f7ed	WindowSnipping
	f7ee	VideoCapture
	f809	StatusSecured
	f83b	NarratorApp
	f83d	PowerButtonUpdate
	f83e	RestartUpdate
	f83f	UpdateStatusDot
	f847	Eject
	f87b	Spelling
	f87c	SpellingKorean
	f87d	SpellingSerbian
	f87e	SpellingChinese
	f89a	FolderSelect
	f8a5	SmartScreen

	f8a6	ExploitProtection
	f8aa	AddBold
	f8ab	SubtractBold
	f8ac	BackSolidBold
	f8ad	ForwardSolidBold
	f8ae	PauseBold
	f8af	ClickSolid
	f8b0	SettingsSolid
	f8b1	MicrophoneSolidBold
	f8b2	SpeechSolidBold
	f8b3	ClickedOutLoudSolidBold
	f8c0	VPNOverlay
	f8c1	VPNRoamingOverly
	f8c2	WifiVPN3
	f8c3	WifiVPN4
	f8c4	WifiVPN5
	f8c5	SignalBarsVPN2
	f8c6	SignalBarsVPN3
	f8c7	SignalBarsVPN4

	f8c8	SignalBarsVPN5
	f8c9	SignalBarsVPNRoaming3
	f8ca	SignalBarsVPNRoaming4
	f8cb	SignalBarsVPNRoaming5
	f8cc	EthernetVPN

Layout

Article • 11/26/2024

These articles help you create a flexible UI that looks great on different screen sizes, window sizes, resolutions, and orientations.

App silhouettes

Learn about common layout patterns.

App title bar

Learn about design recommendations and behaviors of the parts of a standard title bar.

Screen sizes and breakpoints

Learn about screen sizes across the Windows ecosystem and how to design for breakpoints.

Responsive design techniques

Learn about responsive design techniques that optimize your app's layout for breakpoints.

Layouts with XAML

Implement responsive UI in XAML with adaptive or tailored layouts.

Layout panels

Learn about each type of layout panel and how to use them to arrange UI elements.

Alignment, margin, padding

Use alignment, margin, and padding to influence layout behavior.

Spacing

Use sizing to provide a balance between information density and user comfort.

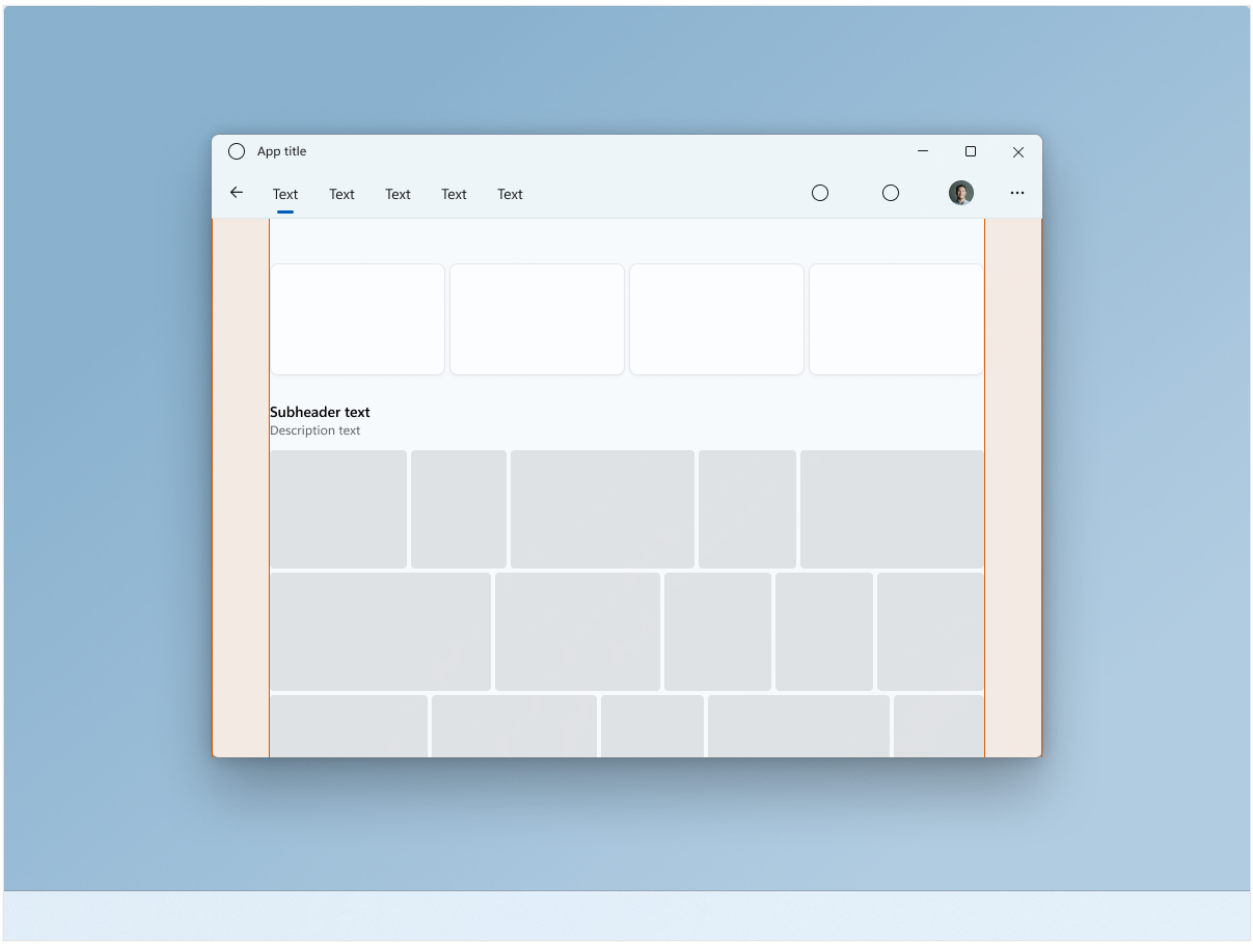
Windows app silhouettes

Article • 10/24/2022

Silhouettes indicate a common pattern of relationships between elements such as app layering, menus, navigation, commanding and content areas. This article focuses on the common silhouettes as used in several Windows in-box apps.

Also refer to Content Basics for common arrangements of content and controls.

Top navigation silhouette



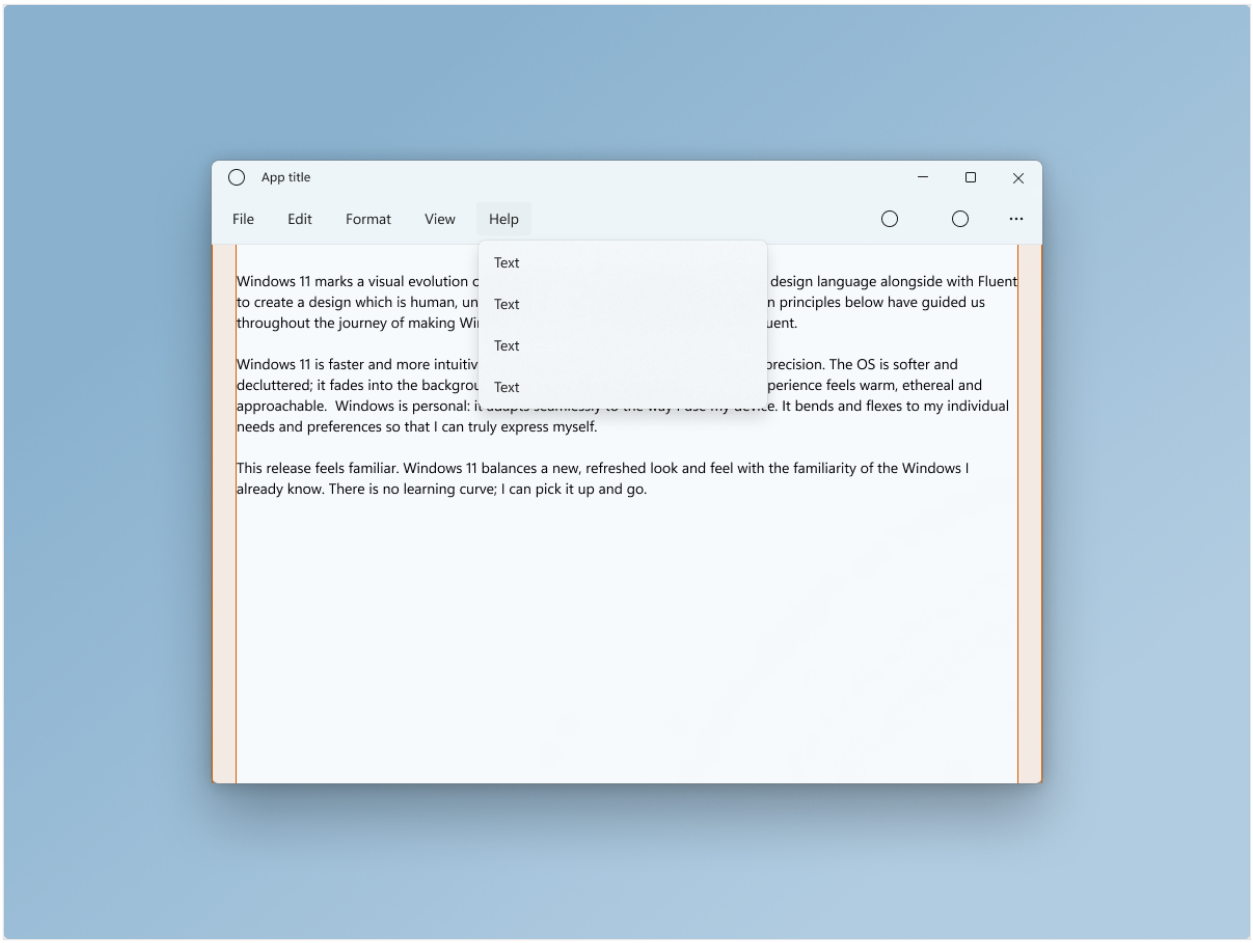
A NavigationView can be used at the top of the app's content layer, building a connection with the content below. Note the location of user identity/person/picture control when using top navigation.

Placing navigation on the same row as commands can be useful when trying to maximize the amount of vertical space for the content below.

Content margins can vary. This example uses 56epx margins, complementing large pieces of media. Use smaller margins for smaller/tighter content needs.

In Windows 11, Photos is a good example of an app that uses a top navigation silhouette.

Menu bar silhouette

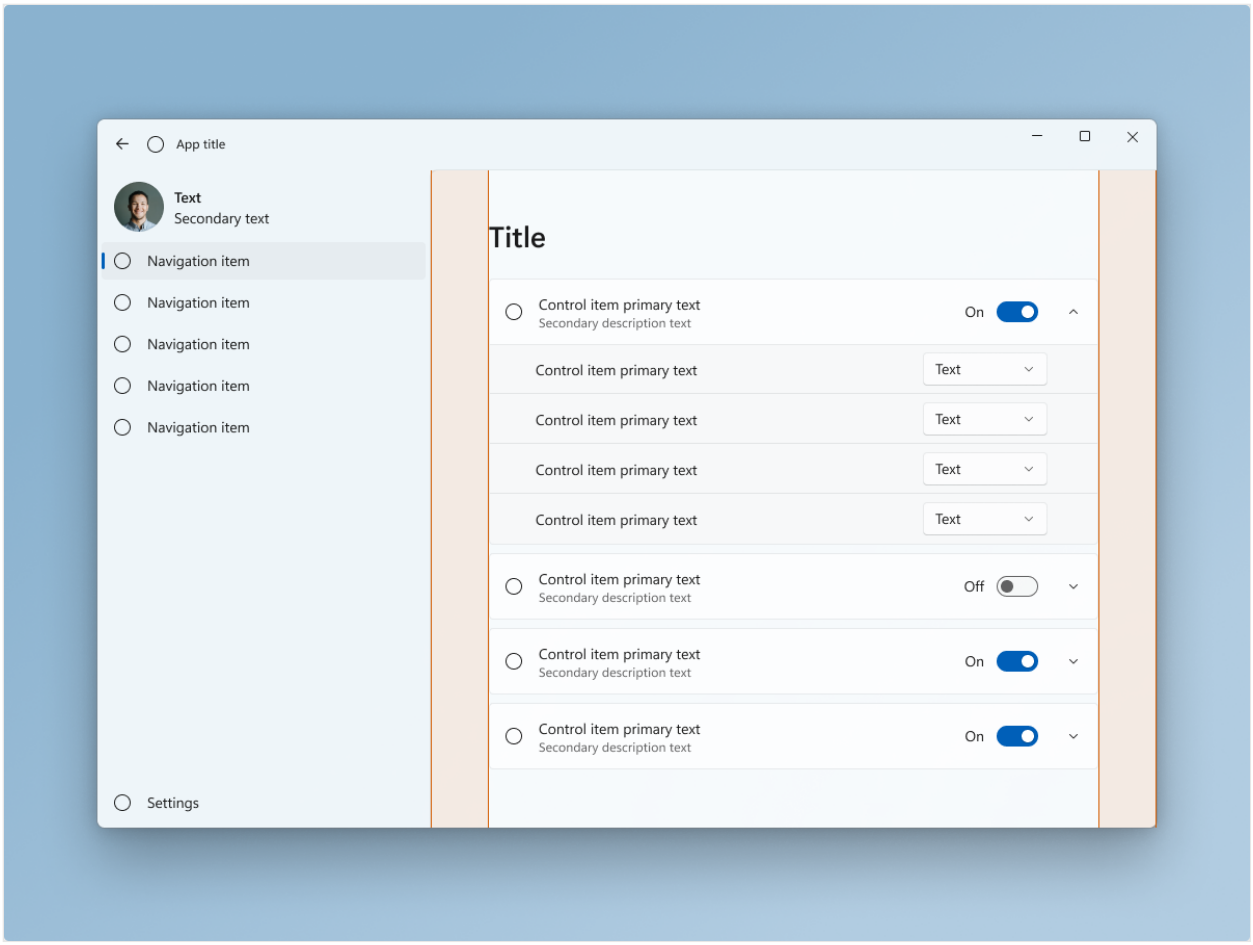


A MenuBar can be used as part of the of the base layer along with a CommandBar. This brings more focus to the content area's primary task, in this case composition and editing.

This example depicts a text editor using 12epx margins to compliment the utility of the app.

In Windows 11, Notepad is a good example of an app that uses a menu bar silhouette.

Left navigation silhouette

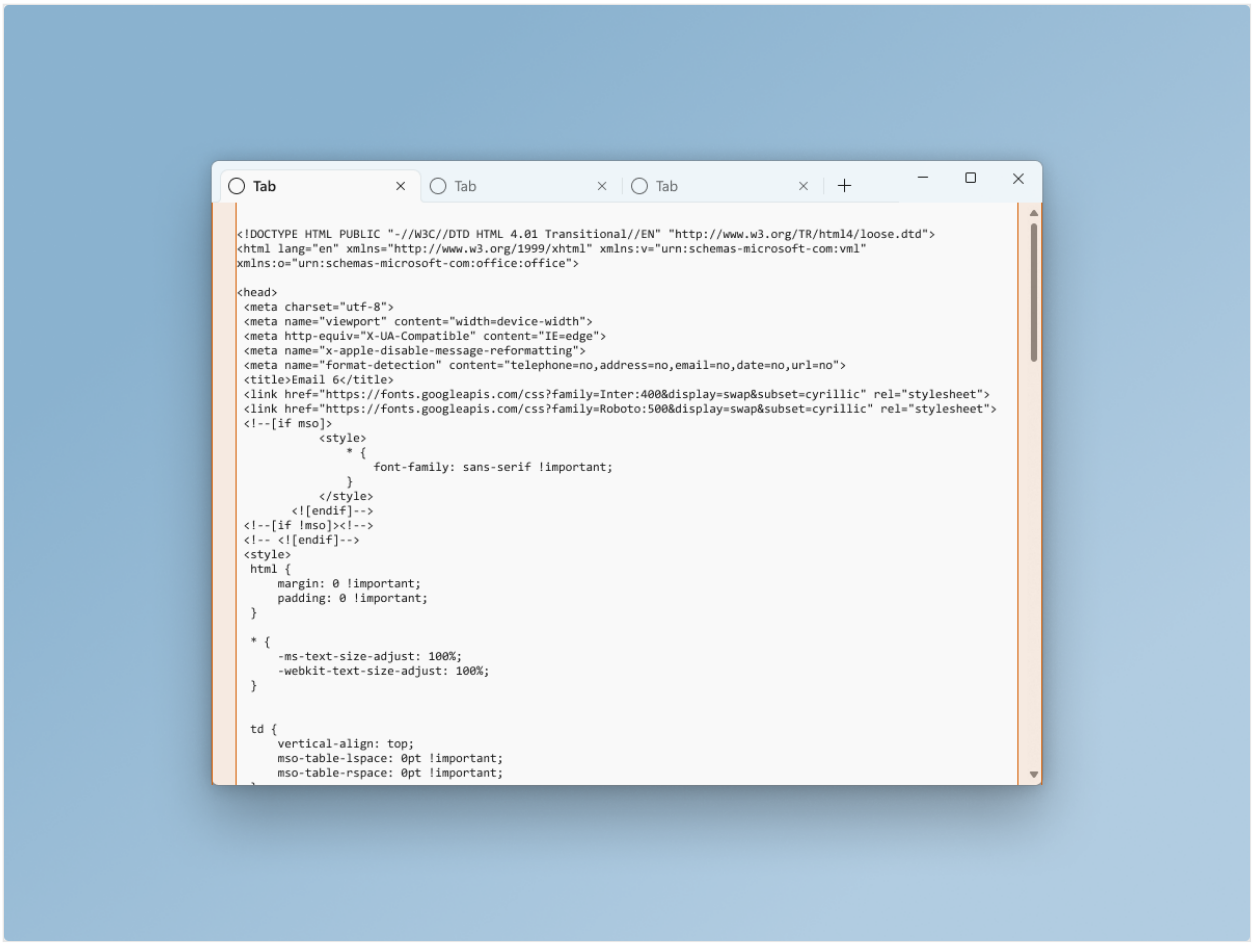


NavigationView controls automatically rests on the app's base layer. This brings more focus to the content area's primary task. Note the location of user identity/person/picture control when using left navigation.

Content margins can vary. This example uses 56epx margins to compliment the cohesion of the content within the expanders. Use smaller margins when content cohesion is less of a concern, because other design elements reinforce cohesion, content is not nested in expanders, or content should not logically be grouped together.

In Windows 11, Settings is a good example of an app that uses a left navigation silhouette.

Tab View Silhouette



A TabView can integrate with the app's base layer, and title barcontrol. This brings more focus to the content area's primary task, in this case code composition and editing.

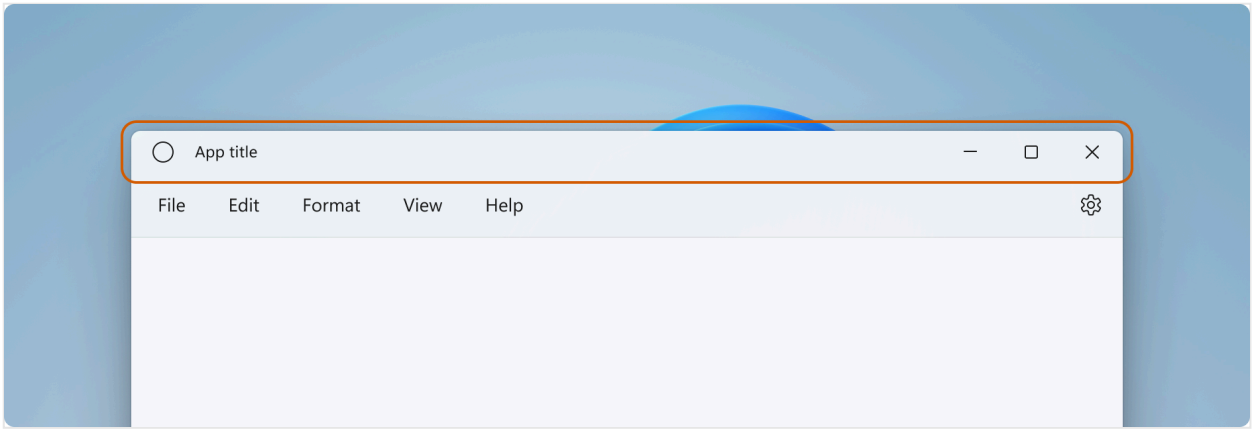
This example depicts a text editor using 12epx margins to compliment the utility of the app.

In Windows 11, Terminal is a good example of an app that uses a tab view silhouette.

Title bar design

Article • 07/31/2024

The title bar sits at the top of an app on the base layer. Its main purpose is to allow users to be able to identify the app via its title, move the app window, and minimize, maximize, or close the app.



Standard design

This section describes the design recommendations and behaviors of the parts of a standard title bar.

Bar

Design

- The standard title bar has a height of 32px.
- The title bar's default background is Mica, however we recommend that title bars blend with the rest of the window if possible.
- Title bars help users differentiate when a window is active and inactive. All title bar elements should be semi-transparent when the window is inactive.
- The title bar colors should adjust when users switch to high contrast themes, or between light and dark modes.
  - For high contrast themes, apps should use the `SystemColors` class for determining proper UI element coloring to facilitate a superior high-contrast experience.

Behavior

- The title bar plays a vital role in repositioning and resizing a window. All empty space in the title bar or space taken up by non-interactive elements like the

window title should be draggable.
- A right-click/press-and-hold on any part of the title bar that does not have an interactive element should show the system window menu.
- A double-click/tap should toggle between maximizing the window and restoring the window.

Icon

Design

- The size of the window icon is 16px by 16px.
- Place the icon 16px from the left-most border in LTR, or right-most border in RTL.
  - If the back button is present, place the window icon 16px to the right of the back button.
- The window icon should be vertically centered in the title bar. For example, When the title bar height is 32px, the top and bottom margins are 8px.

Behavior

- A single-click/tap on the icon should show system window menu.
- A double-click/tap should close the window.

Title

Design

- Place the window title 16px from the window icon or back button.
  - If neither an icon nor back button are present, place the window title 16px from the left-most border in LTR, or right-most border in RTL.
- The window title should use the Segoe UI Variable (if available) or Segoe UI font.
- The window title should use *caption* style text (see XAML type ramp).
- The window title can be truncated, and an ellipsis added if the window width is smaller than the length of the title bar elements. The icon and caption buttons (min, max and close) should always be fully visible.

Behavior

- A right-click/press-and-hold on the icon should show the system window menu.
- A double-click/tap should toggle between maximizing the window and restoring the window.
- The window title and other textual elements in the title bar should respond to text-scaling. This might require that the title bar grows in height.

Caption controls (minimize, maximize, restore, close)

If you create your own caption buttons for your app, follow these guidelines to match the system caption buttons.

Design

- Use these icons for the buttons:
  - Minimize icon: E921 ChromeMinimize
  - Maximize icon: E922 ChromeMaximize
  - Restore icon: E923 ChromeRestore
  - Close icon: E8BB ChromeClose
- The icons for the maximize and restore buttons have rounded corners.
- Caption buttons have full bleed backplates.
- Caption buttons respond to *rest*, *on hover*, *on pressed*, *active*, and *inactive* states.

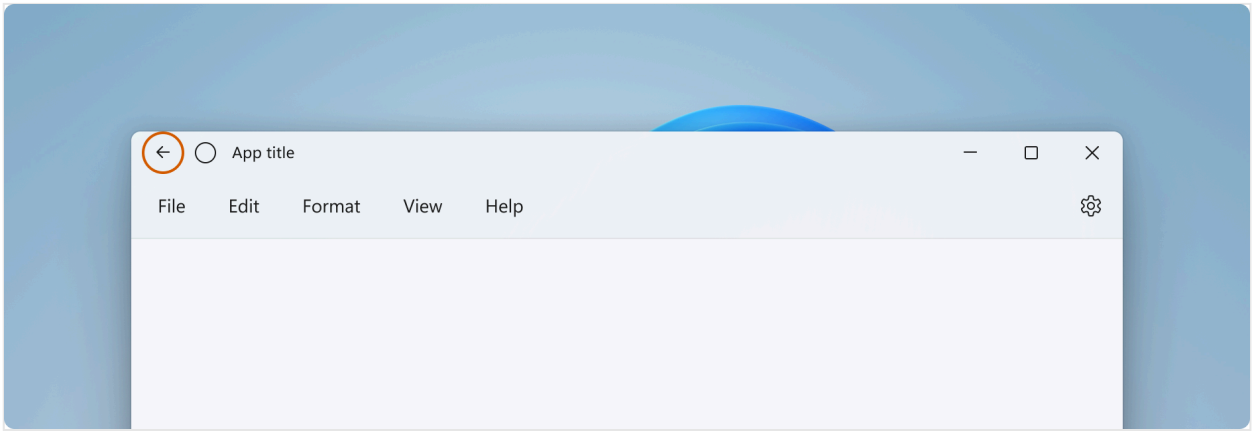
Additional design patterns

Back button

Design

The recommended back button icon is: E830 ChromeBack

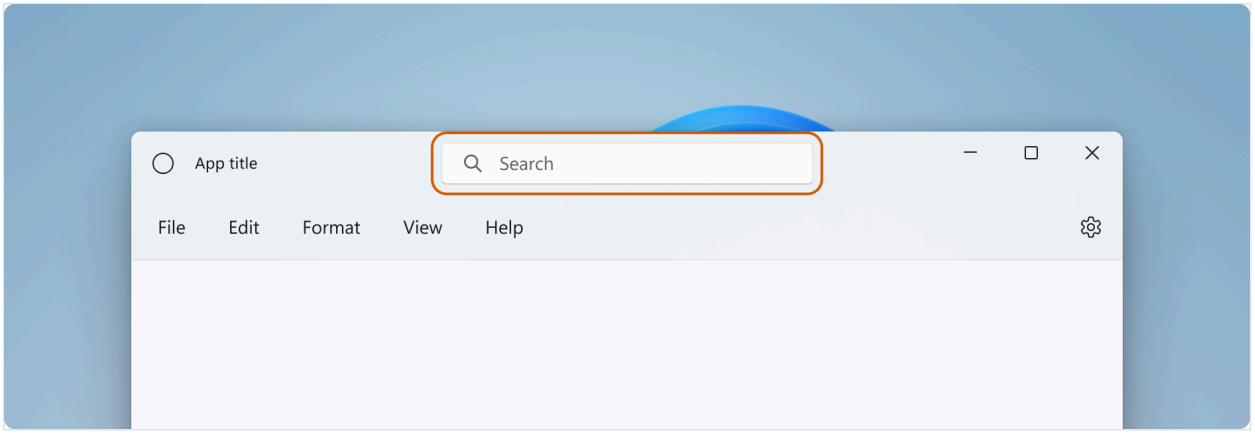
- If a back button is present, it should be placed to the left of the app title or icon/title combination (in LTR).
- The back button responds to *rest*, *on hover*, *on pressed*, *active*, and *inactive* states.
- The back button should be 16px by 16px and vertically centered in the title bar. The button should have a full bleed backplate.
- The back button should be 16px from the left-most border in LTR, or right-most border in RTL, and 16px from the next element to the left or right of it.



Search

Design

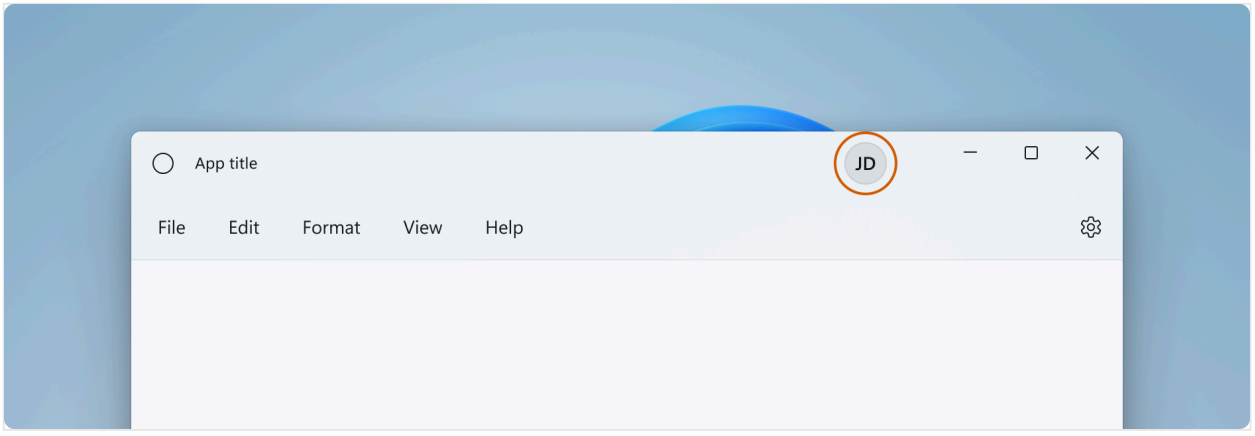
If global search functionality is present, a searchbox should be added to the title bar, centered to the window. Increase the size of the title bar to 48px when you include a searchbox.



The search box needs to be responsive to react to window size changes.

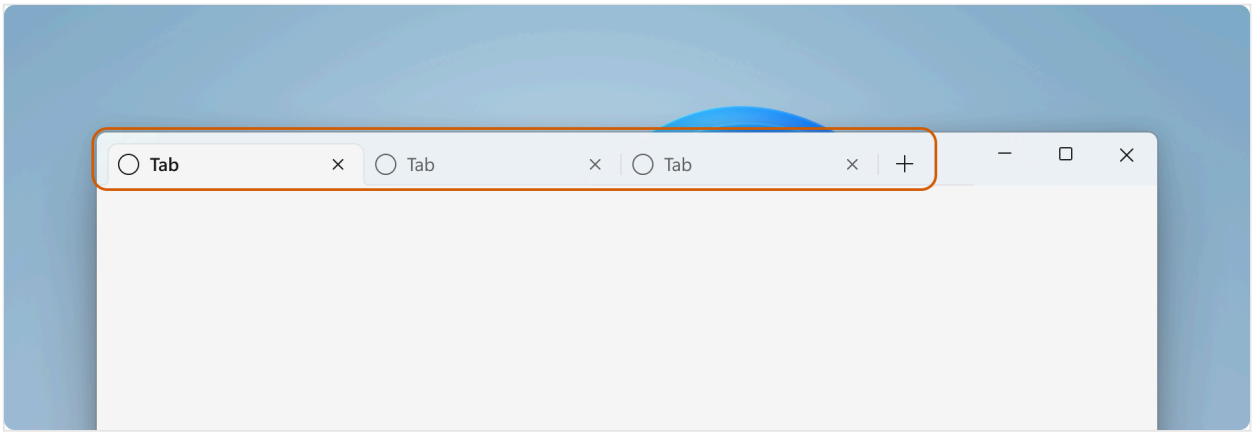
People

If account representation is present, the person-picture control should be placed to the left of the caption controls. Increase the size of the title bar to 48px when you include a person-picture.



Tabs

If you use tabs as the main element of your app, use the title bar space and keep caption controls anchored to the right.



Next steps

See the developer guidance for title bar

Control size and density

Article • 01/22/2025

Use a combination of control size and density to optimize your Windows application and provide a user experience that is most appropriate for your app's functionality and interaction requirements.

By default, XAML apps are rendered with a low-density (or `Standard`) layout. However, beginning with WinUI 2.1, a high-density (or `Compact`) layout option, for information rich UI and similar specialized scenarios, is also supported. This can be specified through a basic style resource (see examples below).

While functionality and behavior has not changed and remains consistent across the two size and density options, the default body font size has been updated to 14px for all controls to support these two density options. This font size works across regions and devices and ensures your application remains balanced and comfortable for users.

Examples

Open the WinUI 3 Gallery app and see Spacing examples

Open the WinUI 3 Gallery app and see Compact Sizing in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Fluent Standard sizing

Fluent Standard sizing was created to provide a balance between information density and user comfort. Effectively, all items on the screen align to a 40x40 effective pixels (epx) target, which lets UI elements align to a grid and scale appropriately based on system level scaling.

Standard sizing is designed to accommodate both touch and pointer input.

> **Note**
>
> For more info on effective pixels and scaling, see Screen sizes and breakpoints

For more info on system level scaling, see Alignment, margin, padding.

Fluent Compact sizing

Compact sizing enables dense, information-rich groups of controls and can help with the following:

- Browsing large amounts of content.
- Maximizing visible content on a page.
- Navigating and interacting with controls and content

Compact sizing is designed primarily to accommodate pointer input.

Examples of compact sizing

Compact sizing is implemented through a special resource dictionary that can be specified in your application at either the page level or on a specific layout. The resource dictionary is available in the WinUI Nuget package.

The following examples show how the `Compact` style can be applied for the page and an individual Grid control.

Page level

XAML

```
<Page.Resources>
    <ResourceDictionary
        Source="ms-appx:///Microsoft.UI.Xaml/DensityStyles/Compact.xaml"/>
</Page.Resources>
```

Grid level

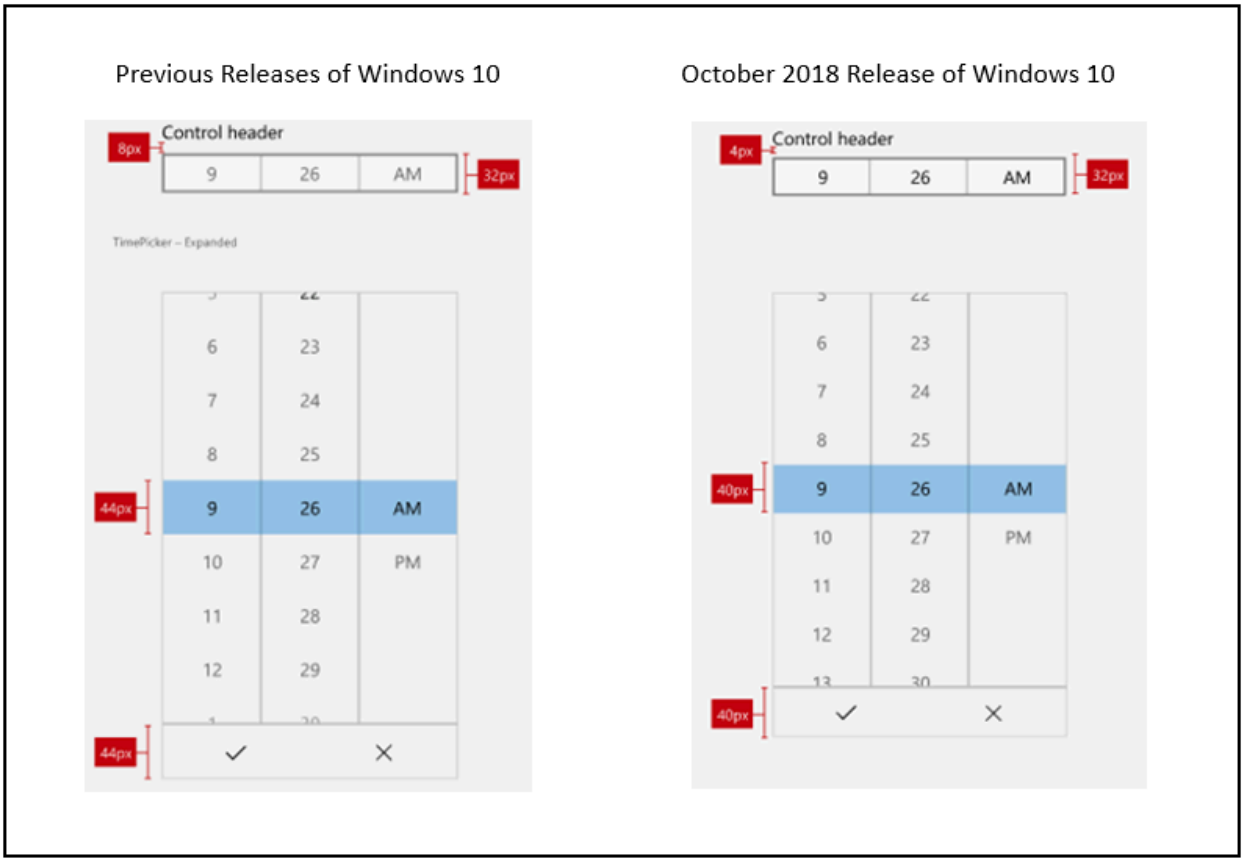
XAML

```
<Grid>
    <Grid.Resources>
        <ResourceDictionary
           Source="ms-
appx:///Microsoft.UI.Xaml/DensityStyles/Compact.xaml"/>
    </Grid.Resources>
</Grid>
```

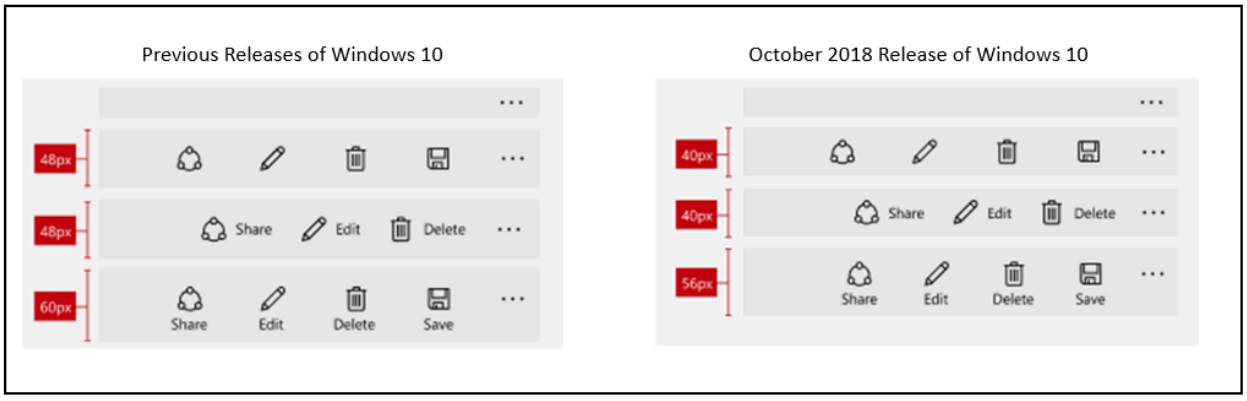
Sizing in Windows apps

In the Windows 10 October 2018 Update (version 1809 and later), the standard, default size for all Windows XAML controls was decreased to increase usability across all usage scenarios.

The following image shows some of the control layout changes that were introduced with the Windows 10 October 2018 Update. Specifically, the margin between a header and the top of a control was decreased from 8epx to 4epx, and the 44epx grid was changed to a 40epx grid.



This next image shows the changes made to control sizes for the Windows 10 October 2018 Update. Specifically, alignment to the 40epx grid.



Related articles

- Guidelines for touch targets
- ResourceDictionary and XAML resource references
- Resource Dictionary
- XAML Styles

Screen sizes and breakpoints

Article • 11/21/2024

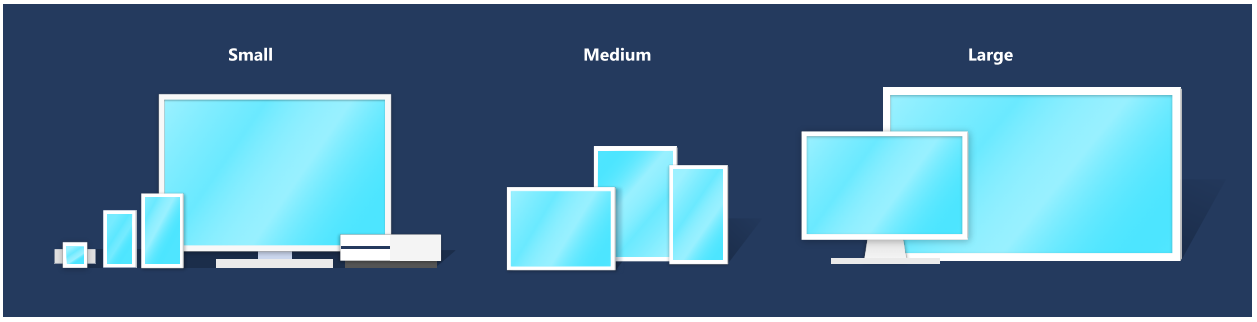
Windows apps can run on any device running Windows, which includes tablets, desktops, TVs, and more. With a huge number of device targets and screen sizes across the Windows ecosystem, rather than optimizing your UI for each device, we recommended designing for a few key width categories (also called "breakpoints"):

- Small (smaller than 640px)
- Medium (641px to 1007px)
- Large (1008px and larger)

> **Tip**
>
> When designing for specific breakpoints, design for the amount of screen space available to your app (the app's window), not the screen size. When the app is running full-screen, the app window is the same size as the screen, but when the app is not full-screen, the window is smaller than the screen.

Breakpoints

This table describes the different size classes and breakpoints.



Expand table

Size class	Breakpoints	Typical screen size	Devices	Window Sizes
Small	up to 640px	20" to 65"	TVs	320x569, 360x640, 480x854
Medium	641 - 1007px	7" to 12"	Tablets	960x540

Size class	Breakpoints	Typical screen size	Devices	Window Sizes
Large	1008px and up	13" and up	PCs, Laptops, Surface Hub	1024x640, 1366x768, 1920x1080

Why are TVs considered "small"?

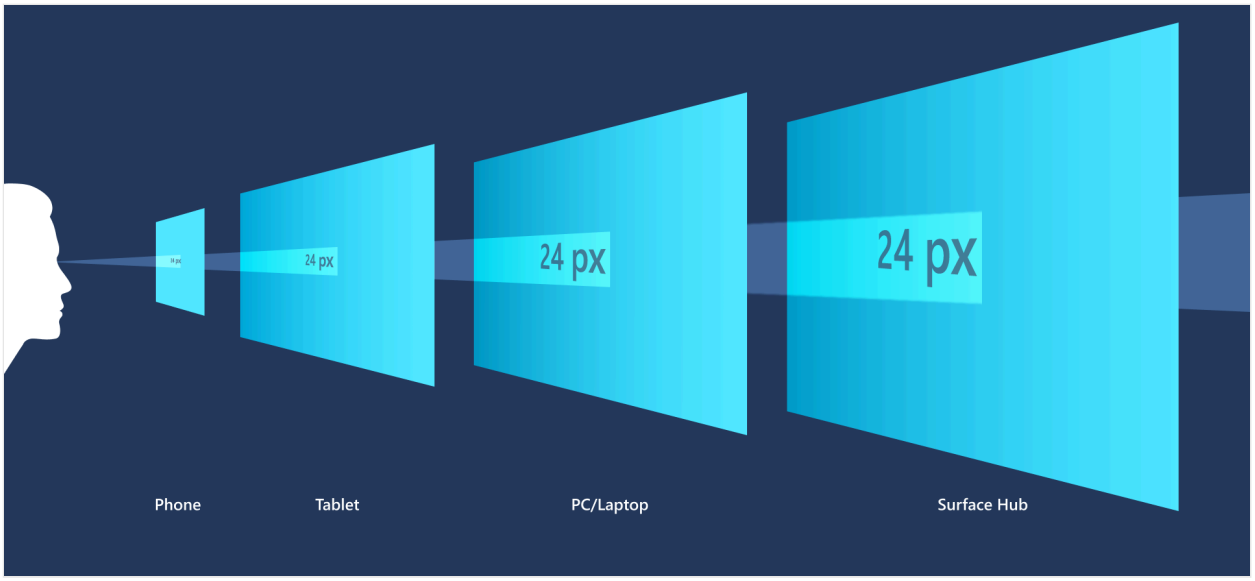
While most TVs are physically quite large (40 to 65 inches is common) and have high resolutions (HD or 4k), designing for a 1080P TV that you view from 10 feet away is different from designing for a 1080p monitor sitting a foot away on your desk. When you account for distance, the TV's 1080 pixels are more like a 540-pixel monitor that's much closer.

XAML's effective pixel system automatically takes viewing distance in account for you. When you specify a size for a control or a breakpoint range, you're actually using "effective" pixels. For example, if you create responsive code for 1080 pixels or more, a 1080 monitor will use that code, but a 1080p TV will not--because although a 1080p TV has 1080 physical pixels, it only has 540 effective pixels. Which makes designing for a TV similar to designing for a small screen.

Effective pixels and scale factor

XAML helps by automatically adjusting UI elements so that they're legible and easy to interact with on all devices and screen sizes.

When your app runs on a device, the system uses an algorithm to normalize the way UI elements display on the screen. This scaling algorithm takes into account viewing distance and screen density (pixels per inch) to optimize for perceived size (rather than physical size). The scaling algorithm ensures that a 24 px font on a large presentation screen 10 feet away is just as legible to the user as a 24 px font on a small portable screen that's a few inches away.

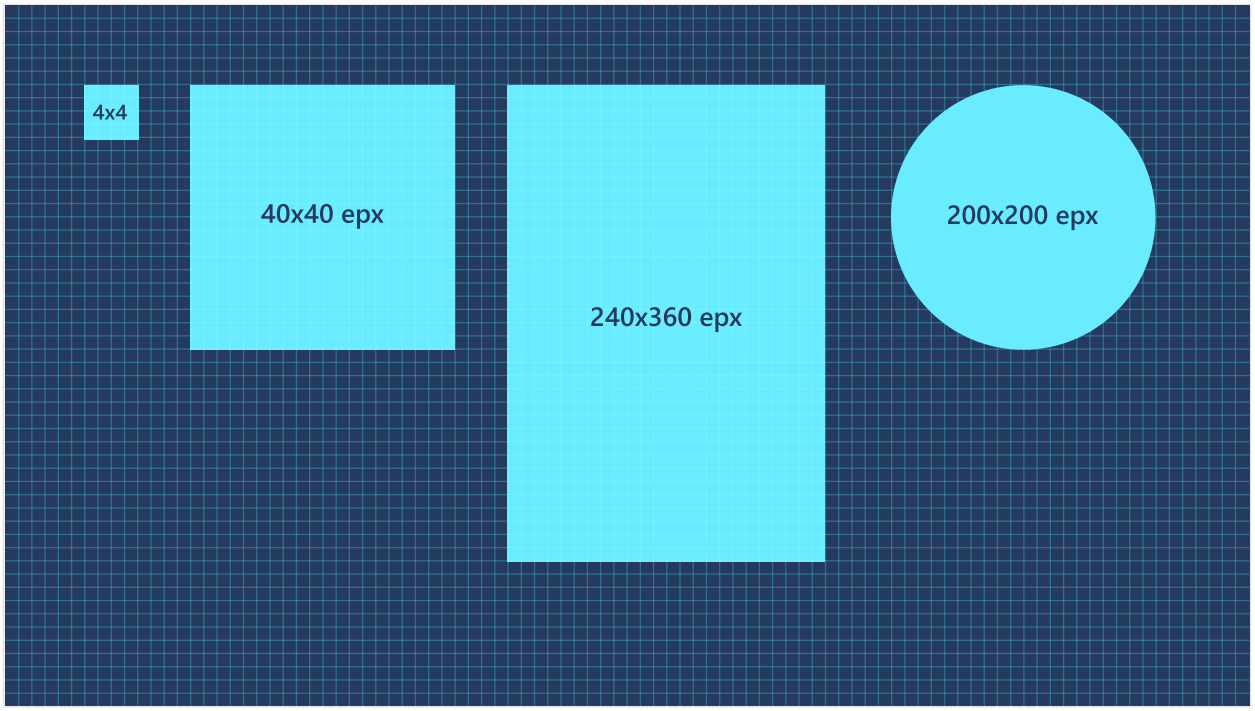


Because of how the scaling system works, when you design your XAML app, you're designing in effective pixels, not actual physical pixels. Effective pixels (epx) are a virtual unit of measurement, and they're used to express layout dimensions and spacing, independent of screen density. (In our guidelines, epx, ep, and px are used interchangeably.)

You can ignore the pixel density and the actual screen resolution when designing. Instead, design for the effective resolution (the resolution in effective pixels) for a size class.

> **Tip**
>
> When creating screen mockups in image editing programs, set the DPI to 72 and set the image dimensions to the effective resolution for the size class you're targeting.

Multiples of Four



The sizes, margins, and positions of UI elements should always be in multiples of 4 epx in your XAML apps.

XAML scales across a range of devices with scaling plateaus of 100%, 125%, 150%, 175%, 200%, 225%, 250%, 300%, 350%, and 400%. The base unit is 4 because it can be scaled to these plateaus as a whole number (for example; 4 x 125% = 5, 4 x 150% = 6). Using multiples of four aligns all UI elements with whole pixels and ensures UI elements have crisp, sharp edges. (Note that text doesn't have this requirement; text can have any size and position.)

Related topics

- Fluent Design - Layout
- Responsive design
- Responsive layouts with XAML
- XAML controls

Responsive design techniques

Article • 11/05/2024

Responsive design uses just one layout where the content is fluid and can adapt to changing window sizes. Responsive design lets you build a feature one time and expect it to work across all screen sizes. *Adaptive* design is similar, but replaces one layout with another layout.

XAML apps use effective pixels to guarantee that your UI will be legible and usable on all Windows-powered devices. So, why would you ever want to customize your app's UI for a specific device or screen size?

- **To make the most effective use of space and reduce the need to navigate**

  If you design an app to look good on a device that has a small screen, such as a tablet, the app will be usable on a PC with a much bigger display, but there will probably be some wasted space. You can customize the app to display more content when the screen is above a certain size. For example, a shopping app might display one merchandise category at a time on a tablet, but show multiple categories and products simultaneously on a PC or laptop.

  By putting more content on the screen, you reduce the amount of navigation that the user needs to perform.

- **To take advantage of devices' capabilities**

  Certain devices are more likely to have certain device capabilities. For example, laptops are likely to have a location sensor and a camera, while a TV might not have either. Your app can detect which capabilities are available and enable features that use them.

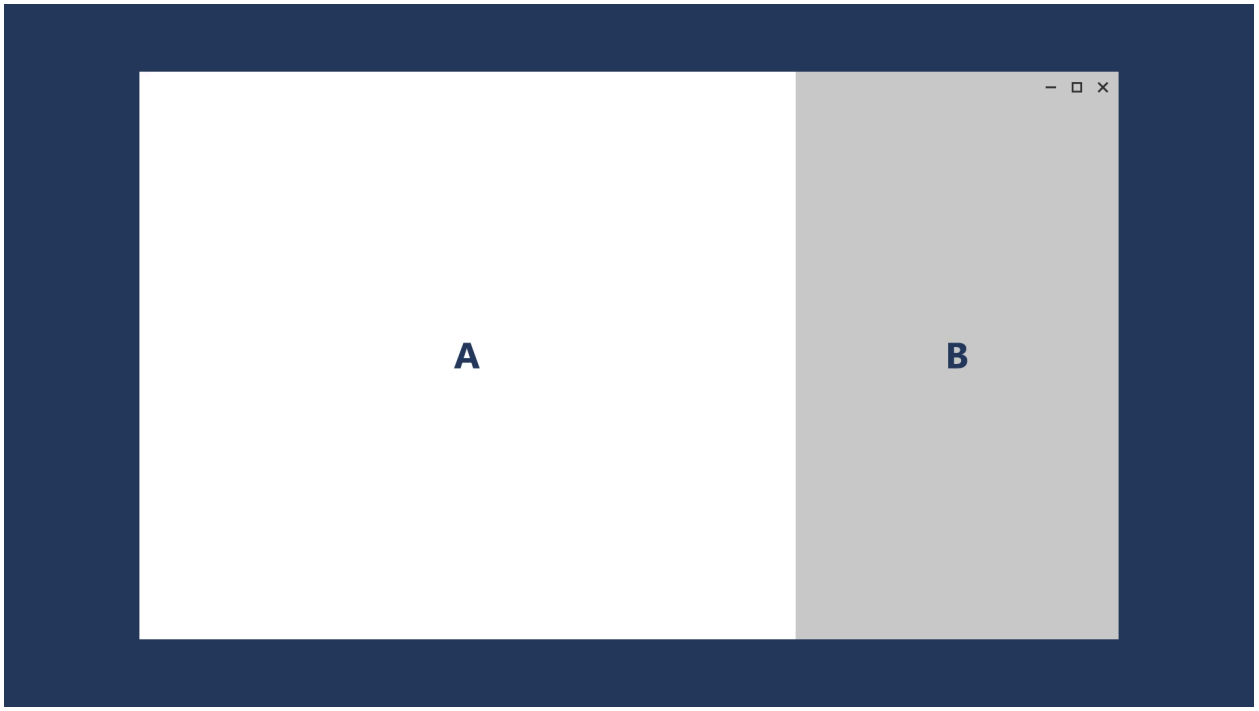
- **To optimize for input**

  The universal control library works with all input types (touch, pen, keyboard, mouse), but you can still optimize for certain input types by re-arranging your UI elements.

When you optimize your app's UI for specific screen widths, we say that you're creating a responsive design. Here are some responsive design techniques you can use to customize your app's UI.

Reposition

You can alter the location and position of UI elements to make the most of the window size. In this example, the smaller window stacks elements vertically. When the app translates to a larger window, elements can take advantage of the wider window width.



In this example design for a photo app, the photo app repositions its content on larger screens.

Resize

You can optimize for the window size by adjusting the margins and size of UI elements. For example, this could augment the reading experience on a larger screen by simply growing the content frame.

Reflow

By changing the flow of UI elements based on device and orientation, your app can offer an optimal display of content. For instance, when going to a larger screen, it might make sense to add columns, use larger containers, or generate list items in a different way.

This example shows how a single column of vertically scrolling content on a smaller screen that can be reflowed on a larger screen to display two columns of text.

Show/hide

You can show or hide UI elements based on screen real estate, or when the device supports additional functionality, specific situations, or preferred screen orientations.

For example, media player controls reduce the button set on smaller screens and expand on larger screens. The media player on a larger window can handle far more on-screen functionality than it can on a smaller window.

Part of the reveal-or-hide technique includes choosing when to display more metadata. With smaller windows, it's best to show a minimal amount of metadata. With larger windows, a significant amount of metadata can be surfaced. Some examples of when to show or hide metadata include:

- In an email app, you can display the user's avatar.
- In a music app, you can display more info about an album or artist.
- In a video app, you can display more info about a film or a show, such as showing cast and crew details.
- In any app, you can break apart columns and reveal more details.
- In any app, you can take something that's vertically stacked and lay it out horizontally. When going from a small window to a larger window, stacked list items can change to reveal rows of list items and columns of metadata.

Re-architect

You can collapse or fork the architecture of your app to better target specific devices. In this example, expanding the window shows the entire list/details pattern.

Adaptive layout

An adaptive layout is similar to responsive layout, but entirely replaces UI based on the format it's presented in. Adaptive design has multiple fixed layout sizes and triggers the page to load a given layout based on the available space.

This technique lets you switch the user interface for a specific breakpoints. In this example, the nav pane and its compact, transient UI works well for a smaller screen, but on a larger screen, tabs might be a better choice.

The NavigationView control supports this technique by letting users set the pane position to either top or left.

Related topics

- Fluent Design - Layout
- Screen sizes and breakpoints
- Responsive layouts with XAML
- XAML controls

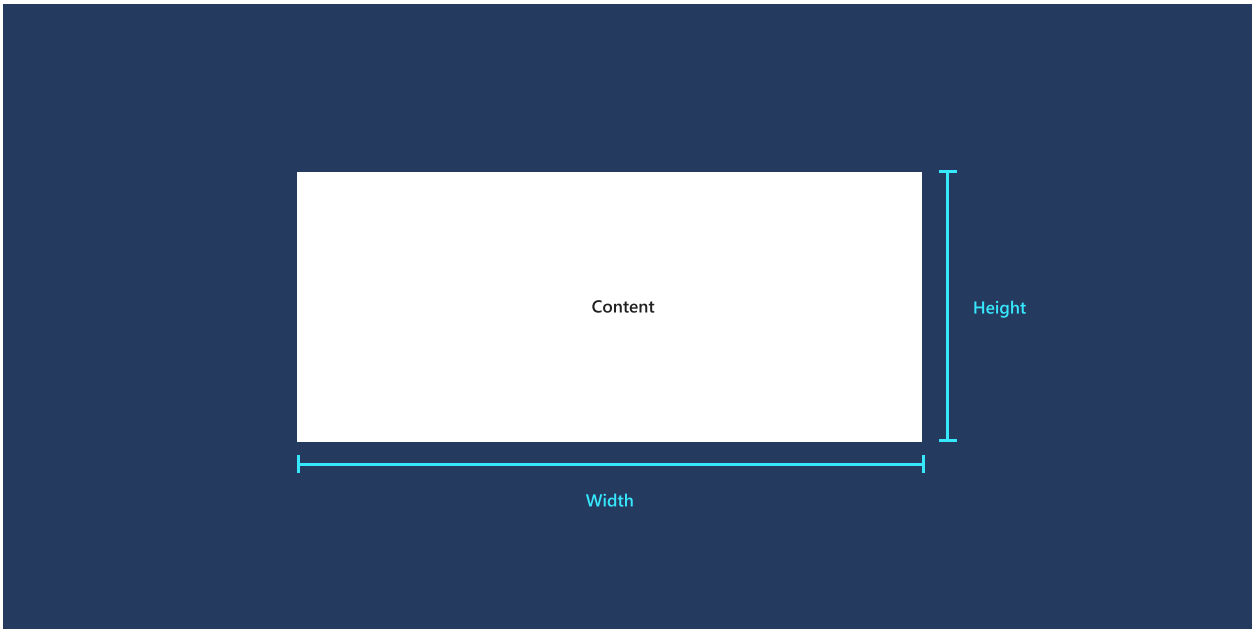
Alignment, margin, padding

Article • 09/26/2024

In XAML apps, most user interface (UI) elements inherit from the **FrameworkElement** class. Every FrameworkElement has dimensions, alignment, margin, and padding properties, which influence layout behavior. The following guidance provides an overview of how to use these layout properties to make sure your app's UI is legible and easy to use in any context.

Dimensions (Height, Width)

Proper sizing ensures all content is clear and legible. Users shouldn't have to scroll or zoom to decipher primary content.

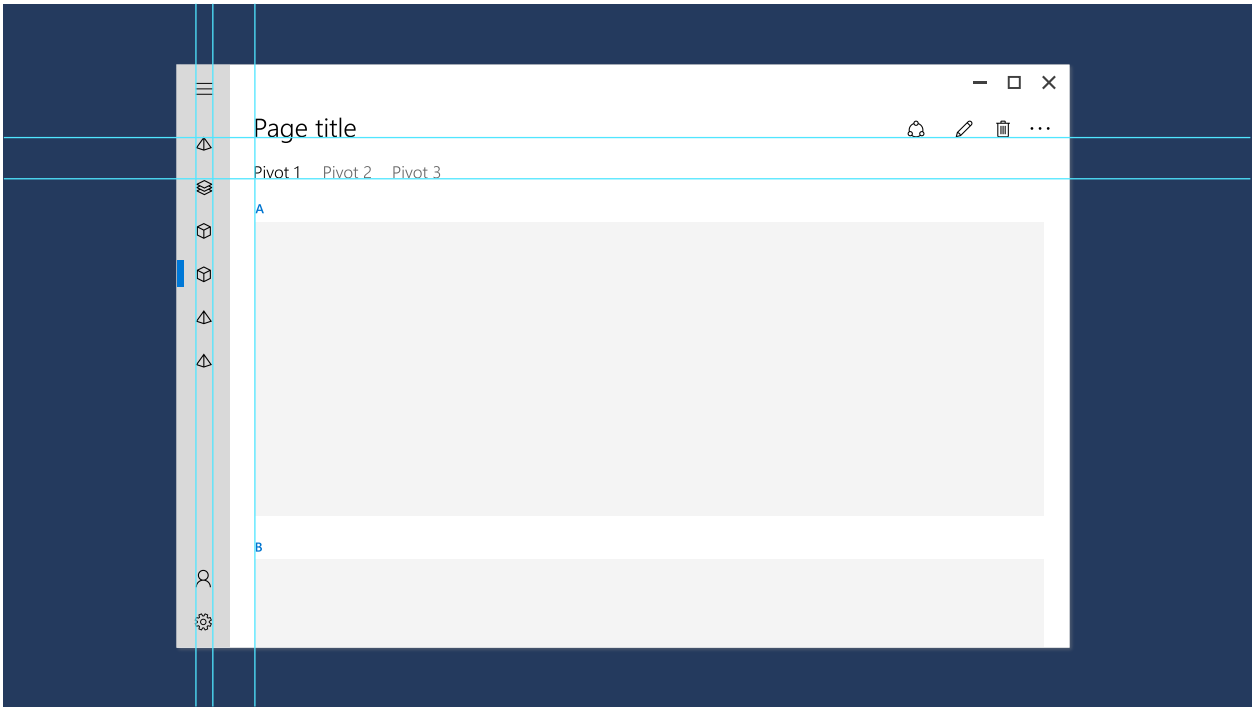


- **Height** and **Width** specify the size of an element. The default values are mathematically `NaN` (Not A Number). You can set fixed values measured in effective pixels, or you can use **Auto** or proportional sizing for fluid behavior.
- **ActualHeight** and **ActualWidth** are read-only properties that provide the size of an element at runtime. If fluid layouts grow or shrink, then the values change in a **SizeChanged** event. Note that a **RenderTransform** will not change the ActualHeight and ActualWidth values.
- **MinWidth**/**MaxWidth** and **MinHeight**/**MaxHeight** specify values that constrain the size of an element while allowing fluid resizing.
- **FontSize** and other text properties control layout size for text elements. While text elements don't have explicitly declared dimensions, they do have calculated

ActualWidth and ActualHeight.

Alignment

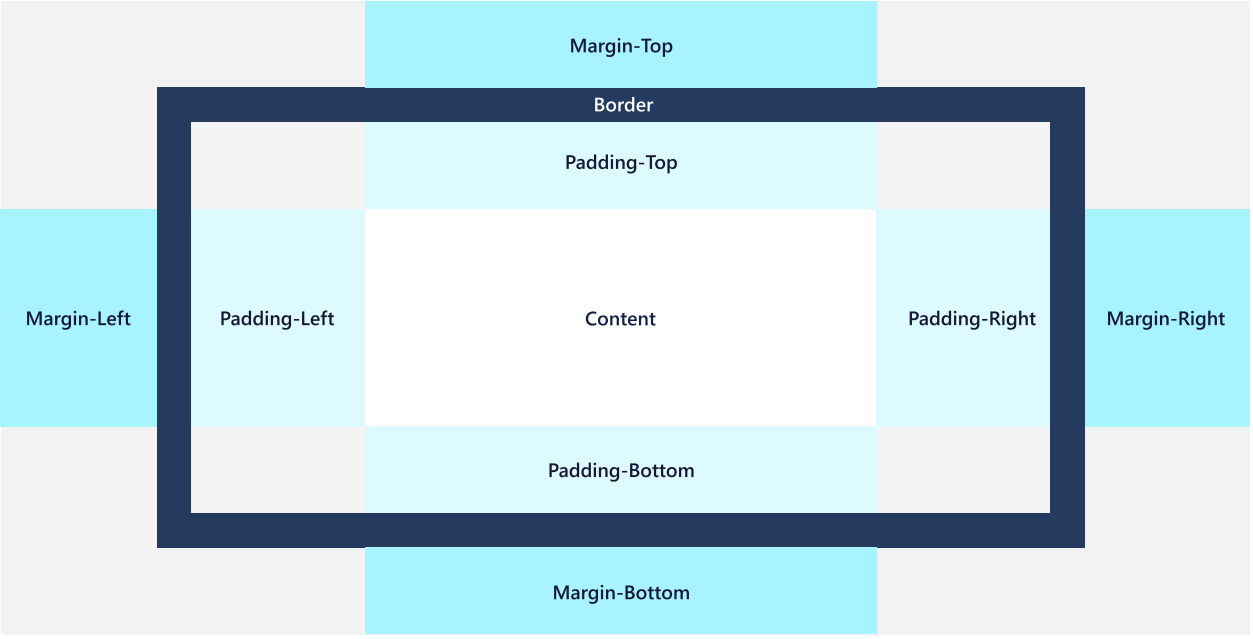
Alignment makes your UI look neat, organized, and balanced and can also be used to establish visual hierarchy and relationships.



- HorizontalAlignment and VerticalAlignment specify how an element should be positioned within its parent container.
  - The values for **HorizontalAlignment** are **Left**, **Center**, **Right**, and **Stretch**.
  - The values for **VerticalAlignment** are **Top**, **Center**, **Bottom**, and **Stretch**.
- **Stretch** is the default for both properties, and elements fill all of the space they're provided in the parent container. Real-number Height and Width cancel a Stretch value, which will instead act as a Center value. Some controls, like Button, override the default Stretch value in their default style.
- HorizontalContentAlignment and VerticalContentAlignment specify how child elements are positioned within a container.
- Alignment can affect clipping within a layout panel. For example, with `HorizontalAlignment="Left"`, the right side of the element gets clipped if the content is larger than the ActualWidth.
- Text elements use the TextAlignment property. Generally, we recommend using left-alignment, the default value. For more information about styling text, see Typography.

Margin and padding

Margin and padding properties keep UI from looking too cluttered or too sparse, and they can also make it easier to use certain inputs like pen and touch. Here's an illustration displaying margins and padding for a container and its content.



Margin

Margin controls the amount of empty space around an element. Margin does not add pixels to ActualHeight and ActualWidth and is not considered part of the element for hit testing and sourcing input events.

- Margin values can be uniform or distinct. With `Margin="20"`, a uniform margin of 20 pixels would be applied to the element on the left, top, right, and bottom sides. With `Margin="0,10,5,25"`, the values are applied to the left, top, right, and bottom (in that order).
- Margins are additive. If two elements both specify a uniform margin of 10 pixels and are adjacent peers in any orientation, the distance between them is 20 pixels.
- Negative margins are permitted. However, using a negative margin can often cause clipping, or overdraws of peers, so it's not a common technique to use negative margins.
- Margin values are constrained last, so be careful with margins because containers can clip or constrain elements. A Margin value could be the cause of an element not appearing to render; with a Margin applied, an element's dimension can be constrained to 0.

Padding

Padding controls the amount of space between the inner border of an element and its child content or elements. A positive Padding value decreases the content area of the element.

Unlike Margin, Padding is not a property of FrameworkElement. There are several classes which each define their own Padding property:

- Control.Padding: inherits to all Control derived classes. Not all controls have content, so for those controls, setting the property does nothing. If the control has a border, the padding applies inside that border.
- Border.Padding: defines space between the rectangle line created by BorderThickness/BorderBrush and the Child element.
- ItemsPresenter.Padding: contributes to appearance of the items in item controls, placing the specified padding around each item.
- TextBlock.Padding and RichTextBlock.Padding: expand the bounding box around the text of the text element. These text elements don't have a **Background**, so it can be visually difficult to see. For that reason, use Margin settings on Block containers instead.

In each of these cases, elements also have a Margin property. If both Margin and Padding are applied, they are additive: the apparent distance between an outer container and any inner content will be margin plus padding.

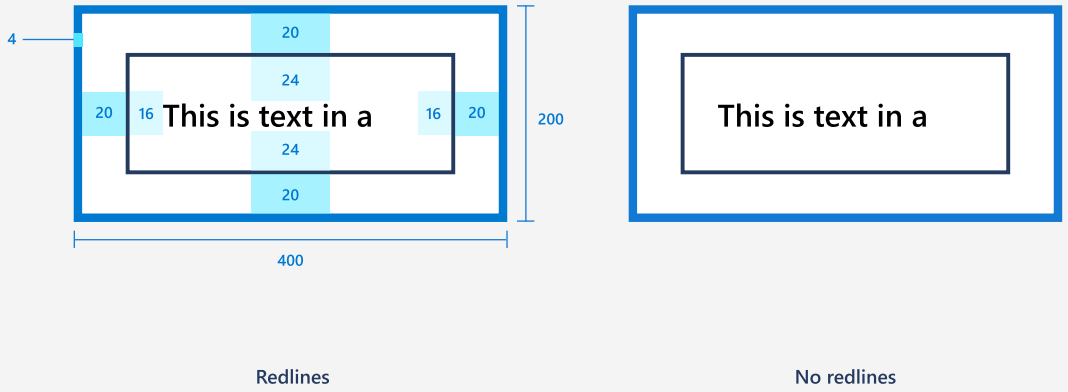
Example

Let's look at the effects of Margin and Padding on real controls. Here's a TextBox inside of a Grid with the default Margin and Padding values of 0.

Here's the same TextBox and Grid with Margin and Padding values on the TextBox as shown in this XAML.

XAML

```
<Grid BorderBrush="Blue" BorderThickness="4" Width="200">
    <TextBox Text="This is text in a TextBox." Margin="20" Padding="16,24"/>
</Grid>
```



Redlines

No redlines

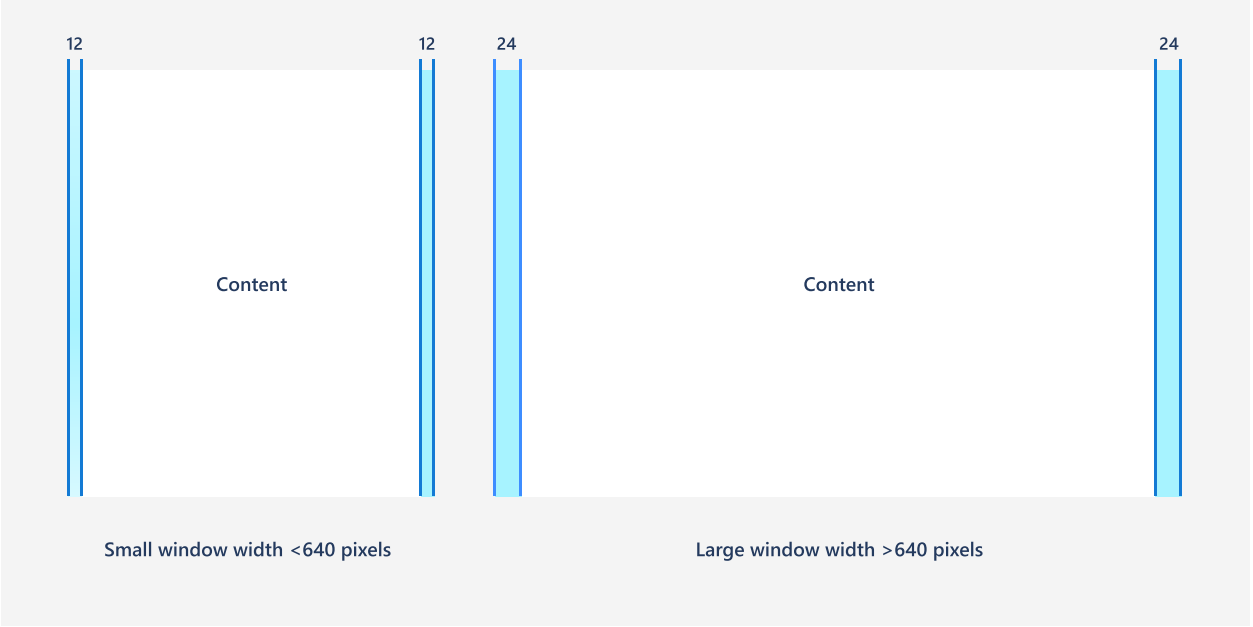
Style resources

You don't have to set each property value individually on a control. It's typically more efficient to group property values into a **Style** resource and apply the Style to a control. This is especially true when you need to apply the same property values to many controls. For more info about using styles, see XAML styles.

General recommendations

- Only apply measurement values to certain key elements and use fluid layout behavior for the other elements. This provides for responsive UI when the window width changes.
- If you do use measurement values, **all dimensions, margins, and padding should be in increments of 4 epx**. When XAML uses effective pixels and scaling to make your app legible on all devices and screen sizes, it scales UI elements by multiples of 4. Using values in increments of 4 results in the best rendering by aligning with whole pixels.

- For small window widths (less than 640 pixels), we recommend 12 epx gutters, and for larger window widths, we recommend 24 epx gutters.



Related topics

- FrameworkElement.Height
- FrameworkElement.Width
- FrameworkElement.HorizontalAlignment
- FrameworkElement.VerticalAlignment
- FrameworkElement.Margin
- Control.Padding

Responsive layouts with XAML

Article • 09/26/2024

The XAML layout system provides automatic sizing of elements, layout panels, and visual states to help you create a responsive UI. With a responsive layout, you can make your app look great on screens with different app window sizes, resolutions, pixel densities, and orientations. You can also use XAML to reposition, resize, reflow, show/hide, replace, or re-architect your app's UI, as discussed in Responsive design techniques. Here, we discuss how to implement responsive layouts with XAML.

Fluid layouts with properties and panels

The foundation of a responsive layout is the appropriate use of XAML layout properties and panels to reposition, resize, and reflow content in a fluid manner.

The XAML layout system supports both static and fluid layouts. In a static layout, you give controls explicit pixel sizes and positions. When the user changes the resolution or orientation of their device, the UI doesn't change. Static layouts can become clipped across different form factors and display sizes. On the other hand, fluid layouts shrink, grow, and reflow to respond to the visual space available on a device.

In practice, you use a combination of static and fluid elements to create your UI. You still use static elements and values in some places, but make sure that the overall UI is responsive to different resolutions, screen sizes, and views.

Here, we discuss how to use XAML properties and layout panels to create a fluid layout.

Layout properties

Layout properties control the size and position of an element. To create a fluid layout, use automatic or proportional sizing for elements, and allow layout panels to position their children as needed.

Here are some common layout properties and how to use them to create fluid layouts.

Height and Width

The **Height** and **Width** properties specify the size of an element. You can use fixed values measured in effective pixels, or you can use auto or proportional sizing.

Auto sizing resizes UI elements to fit their content or parent container. You can also use auto sizing with the rows and columns of a grid. To use auto sizing, set the Height

and/or Width of UI elements to **Auto**.

> **Note**
>
> Whether an element resizes to its content or its container depends on how the parent container handles sizing of its children. For more info, see Layout panels later in this article.

Proportional sizing, also called *star sizing*, distributes available space among the rows and columns of a grid by weighted proportions. In XAML, star values are expressed as * (or *n** for weighted star sizing). For example, to specify that one column is 5 times wider than the second column in a 2-column layout, use "5*" and "*" for the **Width** properties in the **ColumnDefinition** elements.

This example combines fixed, auto, and proportional sizing in a **Grid** with 4 columns.

Expand table

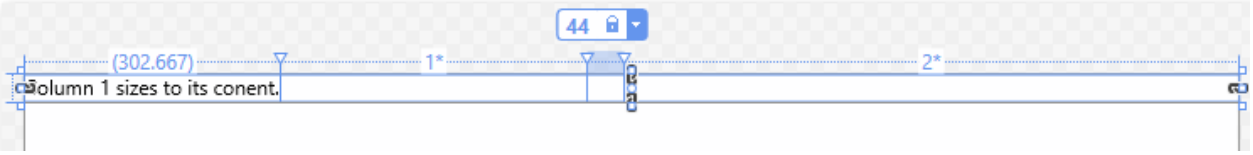
Column	Sizing	Description
Column_1	**Auto**	The column will size to fit its content.
Column_2	*	After the Auto columns are calculated, the column gets part of the remaining width. Column_2 will be one-half as wide as Column_4.
Column_3	**44**	The column will be 44 pixels wide.
Column_4	**2***	After the Auto columns are calculated, the column gets part of the remaining width. Column_4 will be twice as wide as Column_2.

The default column width is "*", so you don't need to explicitly set this value for the second column.

XAML

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition/>
        <ColumnDefinition Width="44"/>
        <ColumnDefinition Width="2*"/>
    </Grid.ColumnDefinitions>
    <TextBlock Text="Column 1 sizes to its content." FontSize="24"/>
</Grid>
```

In the Visual Studio XAML designer, the result looks like this.



To get the size of an element at runtime, use the read-only **ActualHeight** and **ActualWidth** properties instead of Height and Width.

Size constraints

When you use auto sizing in your UI, you might still need to place constraints on the size of an element. You can set the **MinWidth**/**MaxWidth** and **MinHeight**/**MaxHeight** properties to specify values that constrain the size of an element while allowing fluid resizing.

In a Grid, MinWidth/MaxWidth can also be used with column definitions, and MinHeight/MaxHeight can be used with row definitions.

Alignment

Use the **HorizontalAlignment** and **VerticalAlignment** properties to specify how an element should be positioned within its parent container.

- The values for **HorizontalAlignment** are **Left**, **Center**, **Right**, and **Stretch**.
- The values for **VerticalAlignment** are **Top**, **Center**, **Bottom**, and **Stretch**.

With the **Stretch** alignment, elements fill all the space they're provided in the parent container. Stretch is the default for both alignment properties. However, some controls, like **Button**, override this value in their default style. Any element that can have child elements can treat the Stretch value for HorizontalAlignment and VerticalAlignment properties uniquely. For example, an element using the default Stretch values placed in a Grid stretches to fill the cell that contains it. The same element placed in a Canvas sizes to its content. For more info about how each panel handles the Stretch value, see the Layout panels article.

For more info, see the Alignment, margin, and padding article, and the **HorizontalAlignment** and **VerticalAlignment** reference pages.

Visibility

You can reveal or hide an element by setting its **Visibility** property to one of the **Visibility** enumeration values: **Visible** or **Collapsed**. When an element is Collapsed, it doesn't take up any space in the UI layout.

You can change an element's Visibility property in code or in a visual state. When the Visibility of an element is changed, all of its child elements are also changed. You can

replace sections of your UI by revealing one panel while collapsing another.

> **Tip**
>
> When you have elements in your UI that are **Collapsed** by default, the objects are still created at startup, even though they aren't visible. You can defer loading these elements until they are shown by using the **x:Load attribute** to delay the creation of the objects. This can improve startup performance. For more info, see **x:Load attribute**.

Style resources

You don't have to set each property value individually on a control. It's typically more efficient to group property values into a **Style** resource and apply the Style to a control. This is especially true when you need to apply the same property values to many controls. For more info about using styles, see Styling controls.

Layout panels

To position visual objects, you must put them in a panel or other container object. The XAML framework provides various panel classes, such as **Canvas**, **Grid**, **RelativePanel** and **StackPanel**, which serve as containers and enable you to position and arrange the UI elements within them.

The main thing to consider when choosing a layout panel is how the panel positions and sizes its child elements. You might also need to consider how overlapping child elements are layered on top of each other.

Here's a comparison of the main features of the panel controls provided in the XAML framework.

Expand table

Panel Control	Description
Canvas	**Canvas** doesn't support fluid UI; you control all aspects of positioning and sizing child elements. You typically use it for special cases like creating graphics or to define small static areas of a larger adaptive UI. You can use code or visual states to reposition elements at runtime. • Elements are positioned absolutely using Canvas.Top and Canvas.Left attached properties. • Layering can be explicitly specified using the Canvas.ZIndex attached property.

Panel Control	Description
	• Stretch values for HorizontalAlignment/VerticalAlignment are ignored. If an element's size is not set explicitly, it sizes to its content. • Child content is not visually clipped if larger than the panel. • Child content is not constrained by the bounds of the panel.
Grid	**Grid** supports fluid resizing of child elements. You can use code or visual states to reposition and reflow elements. • Elements are arranged in rows and columns using Grid.Row and Grid.Column attached properties. • Elements can span multiple rows and columns using Grid.RowSpan and Grid.ColumnSpan attached properties. • Stretch values for HorizontalAlignment/VerticalAlignment are respected. If an element's size is not set explicitly, it stretches to fill the available space in the grid cell. • Child content is visually clipped if larger than the panel. • Content size is constrained by the bounds of the panel, so scrollable content shows scroll bars if needed.
RelativePanel	• Elements are arranged in relation to the edge or center of the panel, and in relation to each other. • Elements are positioned using a variety of attached properties that control panel alignment, sibling alignment, and sibling position. • Stretch values for HorizontalAlignment/VerticalAlignment are ignored unless RelativePanel attached properties for alignment cause stretching (for example, an element is aligned to both the right and left edges of the panel). If an element's size is not set explicitly and it's not stretched, it sizes to its content. • Child content is visually clipped if larger than the panel. • Content size is constrained by the bounds of the panel, so scrollable content shows scroll bars if needed.
StackPanel	• Elements are stacked in a single line either vertically or horizontally. • Stretch values for HorizontalAlignment/VerticalAlignment are respected in the direction opposite the Orientation property. If an element's size is not set explicitly, it stretches to fill the available width (or height if the Orientation is Horizontal). In the direction specified by the Orientation property, an element sizes to its content. • Child content is visually clipped if larger than the panel. • Content size is not constrained by the bounds of the panel in the direction specified by the Orientation property, so scrollable content stretches beyond the panel bounds and doesn't show scrollbars. You must explicitly constrain the height (or width) of the child content to make its scrollbars show.
VariableSizedWrapGrid	• Elements are arranged in rows or columns that automatically wrap to a new row or column when the MaximumRowsOrColumns value is reached.

Panel Control	Description
	• Whether elements are arranged in rows or columns is specified by the Orientation property. • Elements can span multiple rows and columns using VariableSizedWrapGrid.RowSpan and VariableSizedWrapGrid.ColumnSpan attached properties. • Stretch values for HorizontalAlignment and VerticalAlignment are ignored. Elements are sized as specified by the ItemHeight and ItemWidth properties. If these properties are not set, they take their values from the size of the first cell. • Child content is visually clipped if larger than the panel. • Content size is constrained by the bounds of the panel, so scrollable content shows scroll bars if needed.

For detailed information and examples of these panels, see Layout panels.

Layout panels let you organize your UI into logical groups of controls. When you use them with appropriate property settings, you get some support for automatic resizing, repositioning, and reflowing of UI elements. However, most UI layouts need further modification when there are significant changes to the window size. For this, you can use visual states.

Adaptive layouts with visual states and state triggers

Use visual states to make significant alterations to your UI based on window size or other changes.

When your app window grows or shrinks beyond a certain amount, you might want to alter layout properties to reposition, resize, reflow, reveal, or replace sections of your UI. You can define different visual states for your UI, and apply them when the window width or window height crosses a specified threshold.

A **VisualState** defines property values that are applied to an element when it's in a particular state. You group visual states in a **VisualStateManager** that applies the appropriate VisualState when the specified conditions are met. An **AdaptiveTrigger** provides an easy way to set the threshold (also called 'breakpoint') where a state is applied in XAML. Or, you can call the **VisualStateManager.GoToState** method in your code to apply the visual state. Examples of both ways are shown in the next sections.

Set visual states in code

To apply a visual state from code, you call the VisualStateManager.GoToState method. For example, to apply a state when the app window is a particular size, handle the SizeChanged event and call **GoToState** to apply the appropriate state.

Here, a VisualStateGroup contains two VisualState definitions. The first, `DefaultState`, is empty. When it's applied, the values defined in the XAML page are applied. The second, `WideState`, changes the DisplayMode property of the SplitView to **Inline** and opens the pane. This state is applied in the SizeChanged event handler if the window width is greater than 640 effective pixels.

> **Note**
>
> Windows doesn't provide a way for your app to detect the specific device your app is running on. It can tell you the device family (desktop, etc) the app is running on, the effective resolution, and the amount of screen space available to the app (the size of the app's window). We recommend defining visual states for screen sizes and break points.

XAML

```
<Page ...
    SizeChanged="CurrentWindow_SizeChanged">
    <Grid>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup>
                <VisualState x:Name="DefaultState">
                        <Storyboard>
                        </Storyboard>
                    </VisualState>

                <VisualState x:Name="WideState">
                    <Storyboard>
                        <ObjectAnimationUsingKeyFrames

Storyboard.TargetProperty="SplitView.DisplayMode"
                            Storyboard.TargetName="mySplitView">
                            <DiscreteObjectKeyFrame KeyTime="0">
                                <DiscreteObjectKeyFrame.Value>

<SplitViewDisplayMode>Inline</SplitViewDisplayMode>
                                </DiscreteObjectKeyFrame.Value>
                            </DiscreteObjectKeyFrame>
                        </ObjectAnimationUsingKeyFrames>
                        <ObjectAnimationUsingKeyFrames
                            Storyboard.TargetProperty="SplitView.IsPaneOpen"
                            Storyboard.TargetName="mySplitView">
                            <DiscreteObjectKeyFrame KeyTime="0"
Value="True"/>
```

```
                        </ObjectAnimationUsingKeyFrames>
                    </Storyboard>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>

        <SplitView x:Name="mySplitView" DisplayMode="CompactInline"
                   IsPaneOpen="False" CompactPaneLength="20">
            <!-- SplitView content -->

            <SplitView.Pane>
                <!-- Pane content -->
            </SplitView.Pane>
        </SplitView>
    </Grid>
</Page>
```

C#

```
private void CurrentWindow_SizeChanged(object sender,
Windows.UI.Core.WindowSizeChangedEventArgs e)
{
    if (e.Size.Width > 640)
        VisualStateManager.GoToState(this, "WideState", false);
    else
        VisualStateManager.GoToState(this, "DefaultState", false);
}
```

Set visual states in XAML markup

Prior to Windows 10, VisualState definitions required **Storyboard** objects for property changes, and you had to call **GoToState** in code to apply the state. This is shown in the previous example. You will still see many examples that use this syntax, or you might have existing code that uses it.

Starting in Windows 10, you can use the simplified **Setter** syntax shown here, and you can use a **StateTrigger** in your XAML markup to apply the state. You use state triggers to create simple rules that automatically trigger visual state changes in response to an app event.

This example does the same thing as the previous example, but uses the simplified **Setter** syntax instead of a Storyboard to define property changes. And instead of calling GoToState, it uses the built in **AdaptiveTrigger** state trigger to apply the state. When you use state triggers, you don't need to define an empty `DefaultState`. The default settings are reapplied automatically when the conditions of the state trigger are no longer met.

XAML

```
<Page ...>
    <Grid>
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup>
                <VisualState>
                    <VisualState.StateTriggers>
                        <!-- VisualState to be triggered when the
                             window width is >=640 effective pixels. -->
                        <AdaptiveTrigger MinWindowWidth="640" />
                    </VisualState.StateTriggers>

                    <VisualState.Setters>
                        <Setter Target="mySplitView.DisplayMode"
Value="Inline"/>
                        <Setter Target="mySplitView.IsPaneOpen"
Value="True"/>
                    </VisualState.Setters>
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>

        <SplitView x:Name="mySplitView" DisplayMode="CompactInline"
                   IsPaneOpen="False" CompactPaneLength="20">
            <!-- SplitView content -->

            <SplitView.Pane>
                <!-- Pane content -->
            </SplitView.Pane>
        </SplitView>
    </Grid>
</Page>
```

ⓘ Important

In the previous example, the VisualStateManager.VisualStateGroups attached property is set on the **Grid** element. When you use StateTriggers, always ensure that VisualStateGroups is attached to the first child of the root in order for the triggers to take effect automatically. (Here, **Grid** is the first child of the root **Page** element.)

Attached property syntax

In a VisualState, you typically set a value for a control property, or for one of the attached properties of the panel that contains the control. When you set an attached property, use parentheses around the attached property name.

This example shows how to set the RelativePanel.AlignHorizontalCenterWithPanel attached property on a TextBox named `myTextBox`. The first XAML uses ObjectAnimationUsingKeyFrames syntax and the second uses **Setter** syntax.

XAML

```
<!-- Set an attached property using ObjectAnimationUsingKeyFrames. -->
<ObjectAnimationUsingKeyFrames
    Storyboard.TargetProperty="
(RelativePanel.AlignHorizontalCenterWithPanel)"
    Storyboard.TargetName="myTextBox">
    <DiscreteObjectKeyFrame KeyTime="0" Value="True"/>
</ObjectAnimationUsingKeyFrames>

<!-- Set an attached property using Setter. -->
<Setter Target="myTextBox.(RelativePanel.AlignHorizontalCenterWithPanel)"
Value="True"/>
```

Custom state triggers

You can extend the **StateTrigger** class to create custom triggers for a wide range of scenarios. For example, you can create a StateTrigger to trigger different states based on input type, then increase the margins around a control when the input type is touch. Or create a StateTrigger to apply different states based on the device family the app is run on. For examples of how to build custom triggers and use them to create optimized UI experiences from within a single XAML view, see the State triggers sample.

Visual states and styles

You can use Style resources in visual states to apply a set of property changes to multiple controls. For more info about using styles, see Styling controls.

In this simplified XAML from the State triggers sample, a Style resource is applied to a Button to adjust the size and margins for mouse or touch input. For the complete code and the definition of the custom state trigger, see the State triggers sample.

XAML

```
<Page ... >
    <Page.Resources>
        <!-- Styles to be used for mouse vs. touch/pen hit targets -->
        <Style x:Key="MouseStyle" TargetType="Rectangle">
            <Setter Property="Margin" Value="5" />
            <Setter Property="Height" Value="20" />
            <Setter Property="Width" Value="20" />
        </Style>
```

```
        <Style x:Key="TouchPenStyle" TargetType="Rectangle">
            <Setter Property="Margin" Value="15" />
            <Setter Property="Height" Value="40" />
            <Setter Property="Width" Value="40" />
        </Style>
    </Page.Resources>

    <RelativePanel>
        <!-- ... -->
        <Button Content="Color Palette Button" x:Name="MenuButton">
            <Button.Flyout>
                <Flyout Placement="Bottom">
                    <RelativePanel>
                        <Rectangle Name="BlueRect" Fill="Blue"/>
                        <Rectangle Name="GreenRect" Fill="Green"
RelativePanel.RightOf="BlueRect" />
                        <!-- ... -->
                    </RelativePanel>
                </Flyout>
            </Button.Flyout>
        </Button>
        <!-- ... -->
    </RelativePanel>
    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="InputTypeStates">
            <!-- Second set of VisualStates for building responsive UI
optimized for input type.
                 Take a look at InputTypeTrigger.cs class in CustomTriggers
folder to see how this is implemented. -->
            <VisualState>
                <VisualState.StateTriggers>
                    <!-- This trigger indicates that this VisualState is to
be applied when MenuButton is invoked using a mouse. -->
                    <triggers:InputTypeTrigger TargetElement="{x:Bind
MenuButton}" PointerType="Mouse" />
                </VisualState.StateTriggers>
                <VisualState.Setters>
                    <Setter Target="BlueRect.Style" Value="{StaticResource
MouseStyle}" />
                    <Setter Target="GreenRect.Style" Value="{StaticResource
MouseStyle}" />
                    <!-- ... -->
                </VisualState.Setters>
            </VisualState>
            <VisualState>
                <VisualState.StateTriggers>
                    <!-- Multiple trigger statements can be declared in the
following way to imply OR usage.
                         For example, the following statements indicate that
this VisualState is to be applied when MenuButton is invoked using Touch OR
Pen.-->
                    <triggers:InputTypeTrigger TargetElement="{x:Bind
MenuButton}" PointerType="Touch" />
                    <triggers:InputTypeTrigger TargetElement="{x:Bind
MenuButton}" PointerType="Pen" />
```

```
                </VisualState.StateTriggers>
                <VisualState.Setters>
                    <Setter Target="BlueRect.Style" Value="{StaticResource
TouchPenStyle}" />
                    <Setter Target="GreenRect.Style" Value="{StaticResource
TouchPenStyle}" />
                    <!-- ... -->
                </VisualState.Setters>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
</Page>
```

Related topics

- State triggers sample (GitHub)
- Tailored multiple views sample (GitHub)

Layout panels

Article • 05/24/2022

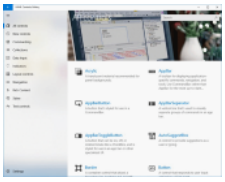
Layout panels are containers that allow you to arrange and group UI elements in your app. The built-in XAML layout panels include **RelativePanel**, **StackPanel**, **Grid**, **VariableSizedWrapGrid**, and **Canvas**. Here, we describe each panel and show how to use it to layout XAML UI elements.

There are several things to consider when choosing a layout panel:

- How the panel positions its child elements.
- How the panel sizes its child elements.
- How overlapping child elements are layered on top of each other (z-order).
- The number and complexity of nested panel elements needed to create your desired layout.

Examples

Expand table

WinUI 2 Gallery
 If you have the **WinUI 2 Gallery** app installed, see the RelativePanel, StackPanel, Grid, VariableSizedWrapGrid, and Canvas in action. • Get the WinUI 2 Gallery app (Microsoft Store) • Get the source code (GitHub)

Panel properties

Before we discuss individual panels, let's go over some common properties that all panels have.

Panel attached properties

Most XAML layout panels use attached properties to let their child elements inform the parent panel about how they should be positioned in the UI. Attached properties use the syntax *AttachedPropertyProvider.PropertyName*. If you have panels that are nested inside other panels, attached properties on UI elements that specify layout characteristics to a parent are interpreted by the most immediate parent panel only.

Here is an example of how you can set the **Canvas.Left** attached property on a Button control in XAML. This informs the parent Canvas that the Button should be positioned 50 effective pixels from the left edge of the Canvas.

XAML

```
<Canvas>
  <Button Canvas.Left="50">Hello</Button>
</Canvas>
```

For more info about attached properties, see Attached properties overview.

Panel borders

The RelativePanel, StackPanel, and Grid panels define border properties that let you draw a border around the panel without wrapping them in an additional Border element. The border properties are **BorderBrush**, **BorderThickness**, **CornerRadius**, and **Padding**.

Here's an example of how to set border properties on a Grid.

XAML

```
<Grid BorderBrush="Blue" BorderThickness="12" CornerRadius="12"
Padding="12">
    <TextBlock Text="Hello World!"/>
</Grid>
```

Using the built-in border properties reduces the XAML element count, which can improve the UI performance of your app. For more info about layout panels and UI performance, see Optimize your XAML layout.

RelativePanel

RelativePanel lets you layout UI elements by specifying where they go in relation to other elements and in relation to the panel. By default, an element is positioned in the upper left corner of the panel. You can use RelativePanel with VisualStateManager and AdaptiveTrigger to rearrange your UI for different window sizes.

This table shows the attached properties you can use to align an element in relation to the panel or other elements.

Expand table

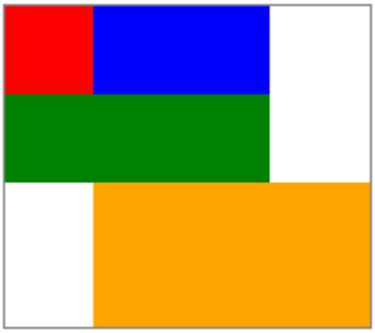
Panel alignment	Sibling alignment	Sibling position
AlignTopWithPanel	AlignTopWith	Above
AlignBottomWithPanel	AlignBottomWith	Below
AlignLeftWithPanel	AlignLeftWith	LeftOf
AlignRightWithPanel	AlignRightWith	RightOf
AlignHorizontalCenterWithPanel	AlignHorizontalCenterWith	
AlignVerticalCenterWithPanel	AlignVerticalCenterWith	

This XAML shows how to arrange elements in a RelativePanel.

XAML

```
<RelativePanel BorderBrush="Gray" BorderThickness="1">
    <Rectangle x:Name="RedRect" Fill="Red" Height="44" Width="44"/>
    <Rectangle x:Name="BlueRect" Fill="Blue"
               Height="44" Width="88"
               RelativePanel.RightOf="RedRect" />

    <Rectangle x:Name="GreenRect" Fill="Green"
               Height="44"
               RelativePanel.Below="RedRect"
               RelativePanel.AlignLeftWith="RedRect"
               RelativePanel.AlignRightWith="BlueRect"/>
    <Rectangle Fill="Orange"
               RelativePanel.Below="GreenRect"
               RelativePanel.AlignLeftWith="BlueRect"
               RelativePanel.AlignRightWithPanel="True"
               RelativePanel.AlignBottomWithPanel="True"/>
</RelativePanel>
```

The result looks like this.



Here are a few things to note about the sizing of the rectangles:

- The red rectangle is given an explicit size of 44x44. It's placed in the upper left corner of the panel, which is the default position.
- The green rectangle is given an explicit height of 44. Its left side is aligned with the red rectangle, and its right side is aligned with the blue rectangle, which determines its width.
- The orange rectangle isn't given an explicit size. Its left side is aligned with the blue rectangle. Its right and bottom edges are aligned with the edge of the panel. Its size is determined by these alignments and it will resize as the panel resizes.

StackPanel

StackPanel arranges its child elements into a single line that can be oriented horizontally or vertically. StackPanel is typically used to arrange a small subsection of the UI on a page.

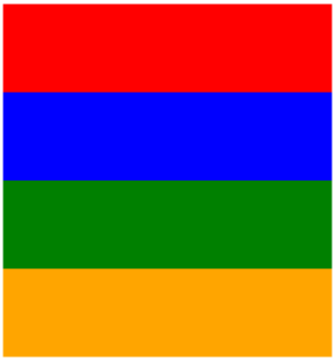
You can use the **Orientation** property to specify the direction of the child elements. The default orientation is **Vertical**.

The following XAML shows how to create a vertical StackPanel of items.

XAML

```
<StackPanel>
    <Rectangle Fill="Red" Height="44"/>
    <Rectangle Fill="Blue" Height="44"/>
    <Rectangle Fill="Green" Height="44"/>
    <Rectangle Fill="Orange" Height="44"/>
</StackPanel>
```

The result looks like this.



In a StackPanel, if a child element's size is not set explicitly, it stretches to fill the available width (or height if the Orientation is **Horizontal**). In this example, the width of the rectangles is not set. The rectangles expand to fill the entire width of the StackPanel.

Grid

The **Grid** panel supports fluid layouts and allows you to arrange controls in multi-row and multi-column layouts. You specify a Grid's rows and columns by using the **RowDefinitions** and **ColumnDefinitions** properties.

To position objects in specific cells of the Grid, use the **Grid.Column** and **Grid.Row** attached properties.

To make content span across multiple rows and columns, use the **Grid.RowSpan** and **Grid.ColumnSpan** attached properties.

This XAML example shows how to create a Grid with two rows and two columns.

XAML

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition Height="44"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <Rectangle Fill="Red" Width="44"/>
    <Rectangle Fill="Blue" Grid.Row="1"/>
    <Rectangle Fill="Green" Grid.Column="1"/>
    <Rectangle Fill="Orange" Grid.Row="1" Grid.Column="1"/>
</Grid>
```

The result looks like this.

In this example, the sizing works like this:

- The second row has an explicit height of 44 effective pixels. By default, the height of the first row fills whatever space is left over.
- The width of the first column is set to **Auto**, so it's as wide as needed for its children. In this case, it's 44 effective pixels wide to accommodate the width of the red rectangle.
- There are no other size constraints on the rectangles, so each one stretches to fill the grid cell it's in.

You can distribute space within a column or a row by using **Auto** or star sizing. You use auto sizing to let UI elements resize to fit their content or parent container. You can also use auto sizing with the rows and columns of a grid. To use auto sizing, set the Height and/or Width of UI elements to **Auto**.

You use proportional sizing, also called *star sizing*, to distribute available space among the rows and columns of a grid by weighted proportions. In XAML, star values are expressed as * (or *n** for weighted star sizing). For example, to specify that one column is 5 times wider than the second column in a 2-column layout, use "5*" and "*" for the **Width** properties in the **ColumnDefinition** elements.

This example combines fixed, auto, and proportional sizing in a **Grid** with 4 columns.

Expand table

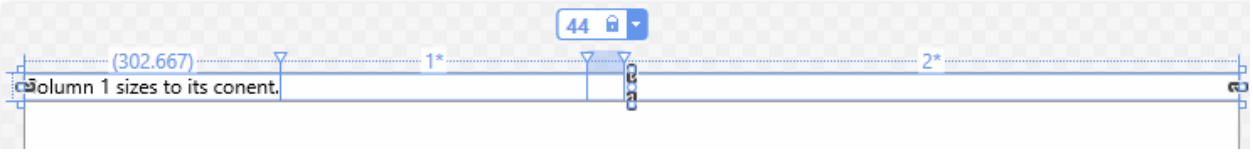
Column	Sizing	Description
Column_1	**Auto**	The column will size to fit its content.
Column_2	*	After the Auto columns are calculated, the column gets part of the remaining width. Column_2 will be one-half as wide as Column_4.
Column_3	**44**	The column will be 44 pixels wide.
Column_4	**2***	After the Auto columns are calculated, the column gets part of the remaining width. Column_4 will be twice as wide as Column_2.

The default column width is "*", so you don't need to explicitly set this value for the second column.

XAML

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition/>
        <ColumnDefinition Width="44"/>
        <ColumnDefinition Width="2*"/>
    </Grid.ColumnDefinitions>
    <TextBlock Text="Column 1 sizes to its content." FontSize="24"/>
</Grid>
```

In the Visual Studio XAML designer, the result looks like this.



VariableSizedWrapGrid

VariableSizedWrapGrid is a Grid-style layout panel where rows or columns automatically wrap to a new row or column when the MaximumRowsOrColumns value is reached.

The Orientation property specifies whether the grid adds its items in rows or columns before wrapping. The default orientation is **Vertical**, which means the grid adds items from top to bottom until a column is full, then wraps to a new column. When the value is **Horizontal**, the grid adds items from left to right, then wraps to a new row.

Cell dimensions are specified by the ItemHeight and ItemWidth. Each cell is the same size. If ItemHeight or ItemWidth is not specified, then the first cell sizes to fit its content, and every other cell is the size of the first cell.

You can use the VariableSizedWrapGrid.ColumnSpan and VariableSizedWrapGrid.RowSpan attached properties to specify how many adjacent cells a child element should fill.

Here's how to use a VariableSizedWrapGrid in XAML.

XAML

```
<VariableSizedWrapGrid MaximumRowsOrColumns="3" ItemHeight="44"
ItemWidth="44">
    <Rectangle Fill="Red"/>
    <Rectangle Fill="Blue"
               VariableSizedWrapGrid.RowSpan="2"/>
    <Rectangle Fill="Green"
               VariableSizedWrapGrid.ColumnSpan="2"/>
    <Rectangle Fill="Orange"
               VariableSizedWrapGrid.RowSpan="2"
               VariableSizedWrapGrid.ColumnSpan="2"/>
</VariableSizedWrapGrid>
```

The result looks like this.

In this example, the maximum number of rows in each column is 3. The first column contains only 2 items (the red and blue rectangles) because the blue rectangle spans 2 rows. The green rectangle then wraps to the top of the next column.

Canvas

The **Canvas** panel positions its child elements using fixed coordinate points and does not support fluid layouts. You specify the points on individual child elements by setting the **Canvas.Left** and **Canvas.Top** attached properties on each element. The parent Canvas reads these attached property values from its children during the Arrange pass of layout.

Objects in a Canvas can overlap, where one object is drawn on top of another object. By default, the Canvas renders child objects in the order in which they're declared, so the last child is rendered on top (each element has a default z-index of 0). This is the same as other built-in panels. However, Canvas also supports the **Canvas.ZIndex** attached property that you can set on each of the child elements. You can set this property in code to change the draw order of elements during run time. The element with the highest Canvas.ZIndex value draws last and therefore draws over any other elements that share the same space or overlap in any way. Note that alpha value (transparency) is respected, so even if elements overlap, the contents shown in overlap areas might be blended if the top one has a non-maximum alpha value.

The Canvas does not do any sizing of its children. Each element must specify its size.

Here's an example of a Canvas in XAML.

XAML

```
<Canvas Width="120" Height="120">
    <Rectangle Fill="Red" Height="44" Width="44"/>
    <Rectangle Fill="Blue" Height="44" Width="44" Canvas.Left="20"
Canvas.Top="20"/>
    <Rectangle Fill="Green" Height="44" Width="44" Canvas.Left="40"
Canvas.Top="40"/>
    <Rectangle Fill="Orange" Height="44" Width="44" Canvas.Left="60"
Canvas.Top="60"/>
</Canvas>
```

The result looks like this.

Use the Canvas panel with discretion. While it's convenient to be able to precisely control positions of elements in UI for some scenarios, a fixed positioned layout panel causes that area of your UI to be less adaptive to overall app window size changes. App window resize might come from device orientation changes, split app windows, changing monitors, and a number of other user scenarios.

Panels for ItemsControl

There are several special-purpose panels that can be used only as an **ItemsPanel** to display items in an **ItemsControl**. These are **ItemsStackPanel**, **ItemsWrapGrid**, **VirtualizingStackPanel**, and **WrapGrid**. You can't use these panels for general UI layout.

Get the sample code

- WinUI 2 Gallery sample - See all the XAML controls in an interactive format.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Materials in Windows

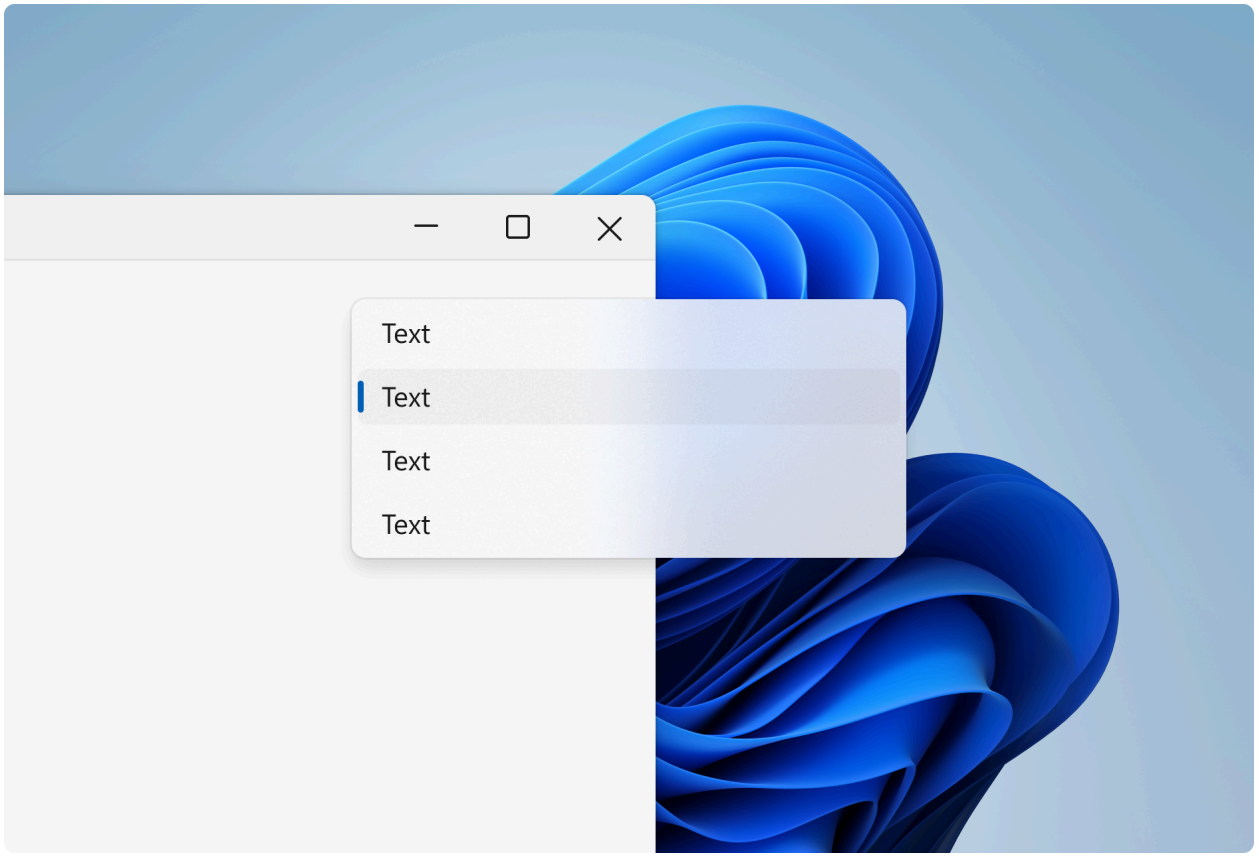
Article • 11/05/2024

Materials are visual effects applied to UX surfaces that resemble real life artifacts. Windows uses two primary types of materials: occluding and transparent. Occluding materials, like *acrylic* and *mica*, are used as base layers beneath interactive UI controls. Transparent materials such as *smoke* are used to highlight immersive surfaces.

Mica, Acrylic, and Smoke each have a specific purpose in how they are used throughout Windows.

> **Tip**
>
> This article describes how the Fluent Design language is applied to Windows apps. For more information, see Fluent Design - Material.

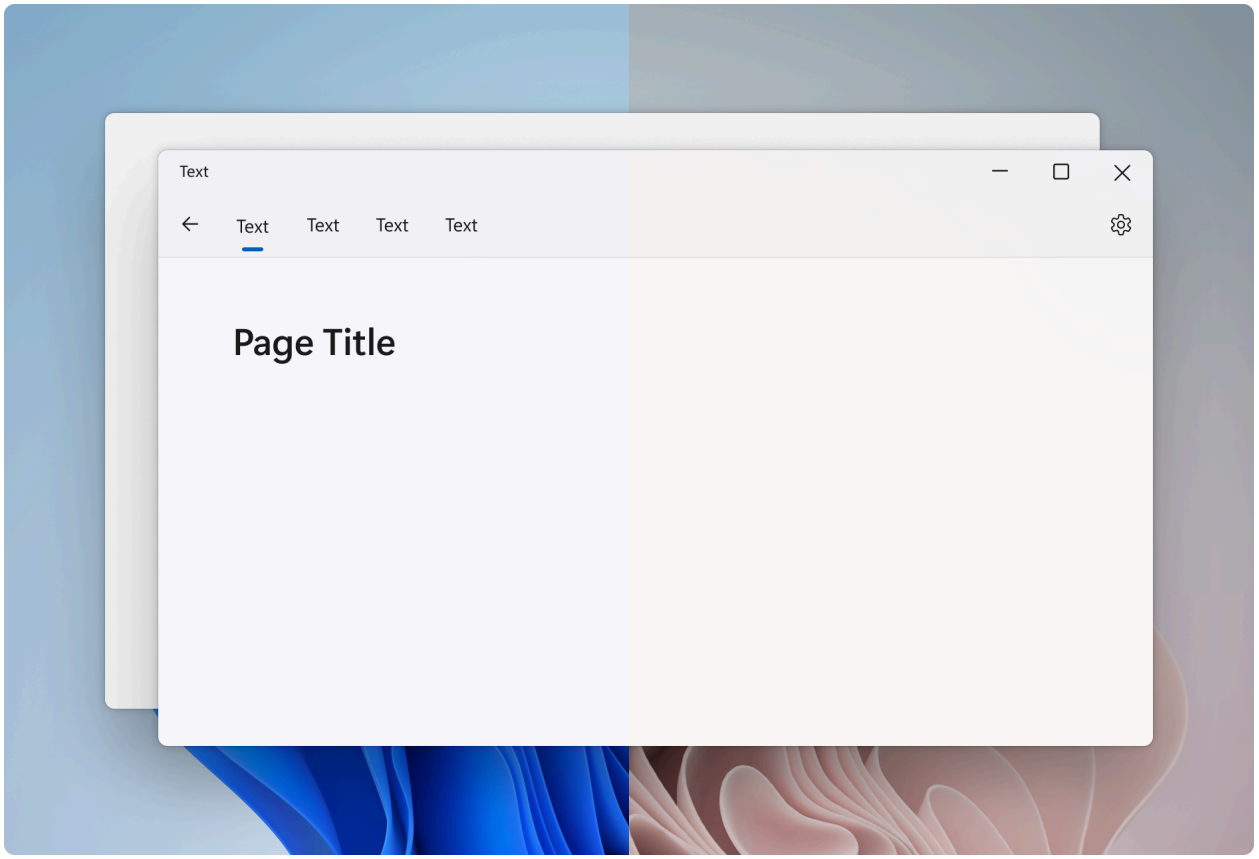
Acrylic



Acrylic is a semi-transparent material that replicates the effect of frosted glass. In Windows 11, acrylic has been updated to be brighter and more translucent, allowing for a stronger contextual relationship with the visuals behind it. Acrylic is used only for transient, light-dismiss surfaces such as flyouts and context menus.

Acrylic is mode aware; it supports both light and dark mode.

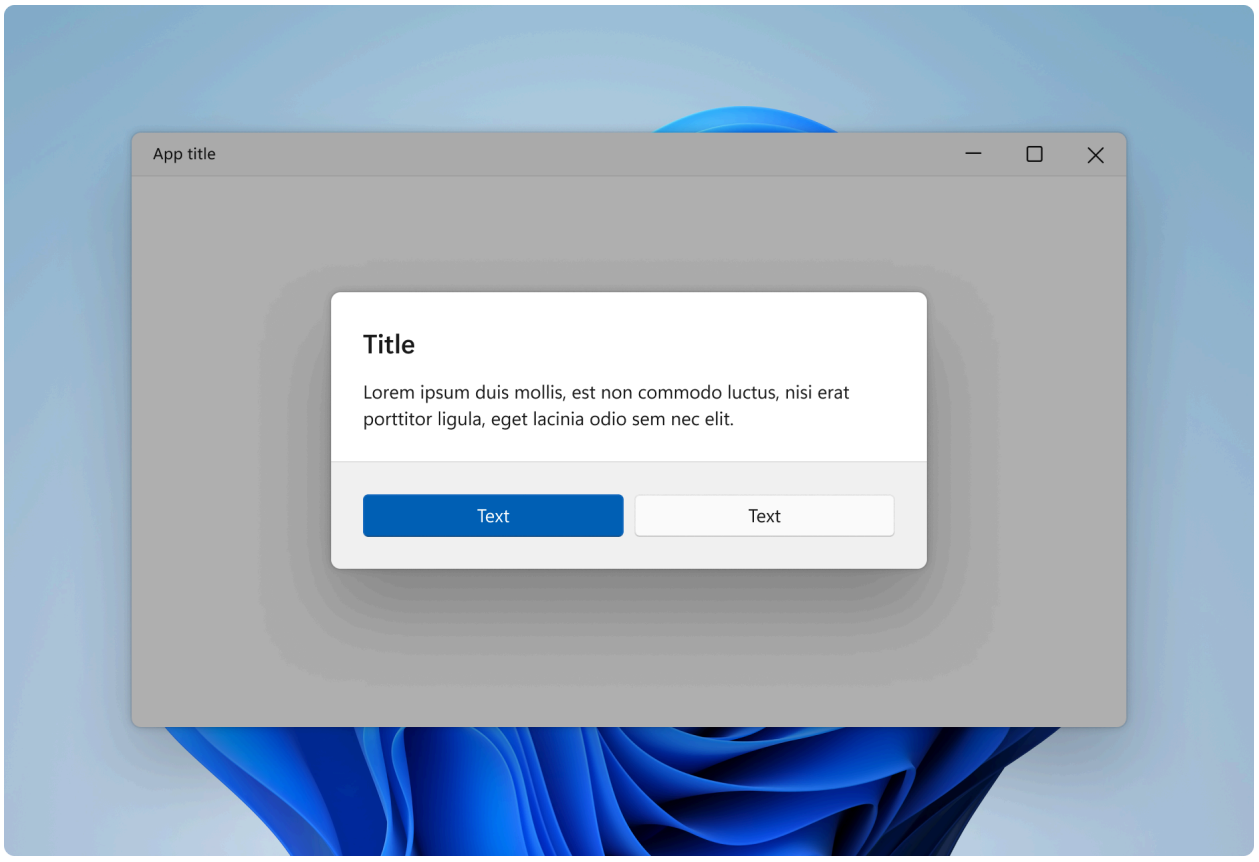
Mica



Mica is a new opaque material introduced in Windows 11. Mica surfaces are subtly tinted with the user's desktop background color.

Mica is mode aware; it supports both light and dark modes. Mica also indicates window focus with active and inactive states as a built in feature.

Smoke



Smoke emphasizes an important UI surface by dimming the surfaces beneath so that they recede into the background. Smoke is used to signal blocking interaction below a modal UI such as a dialog.

Smoke is not mode aware; it is always translucent black in both light and dark mode.

Acrylic material

Article • 07/16/2024

Acrylic is a type of Brush that creates a translucent texture. You can apply acrylic to app surfaces to add depth and help establish a visual hierarchy.

> **Important APIs**: AcrylicBrush class, Background property, Window.SystemBackdrop property, DesktopAcrylicBackdrop class

Acrylic in light theme

Acrylic in dark theme

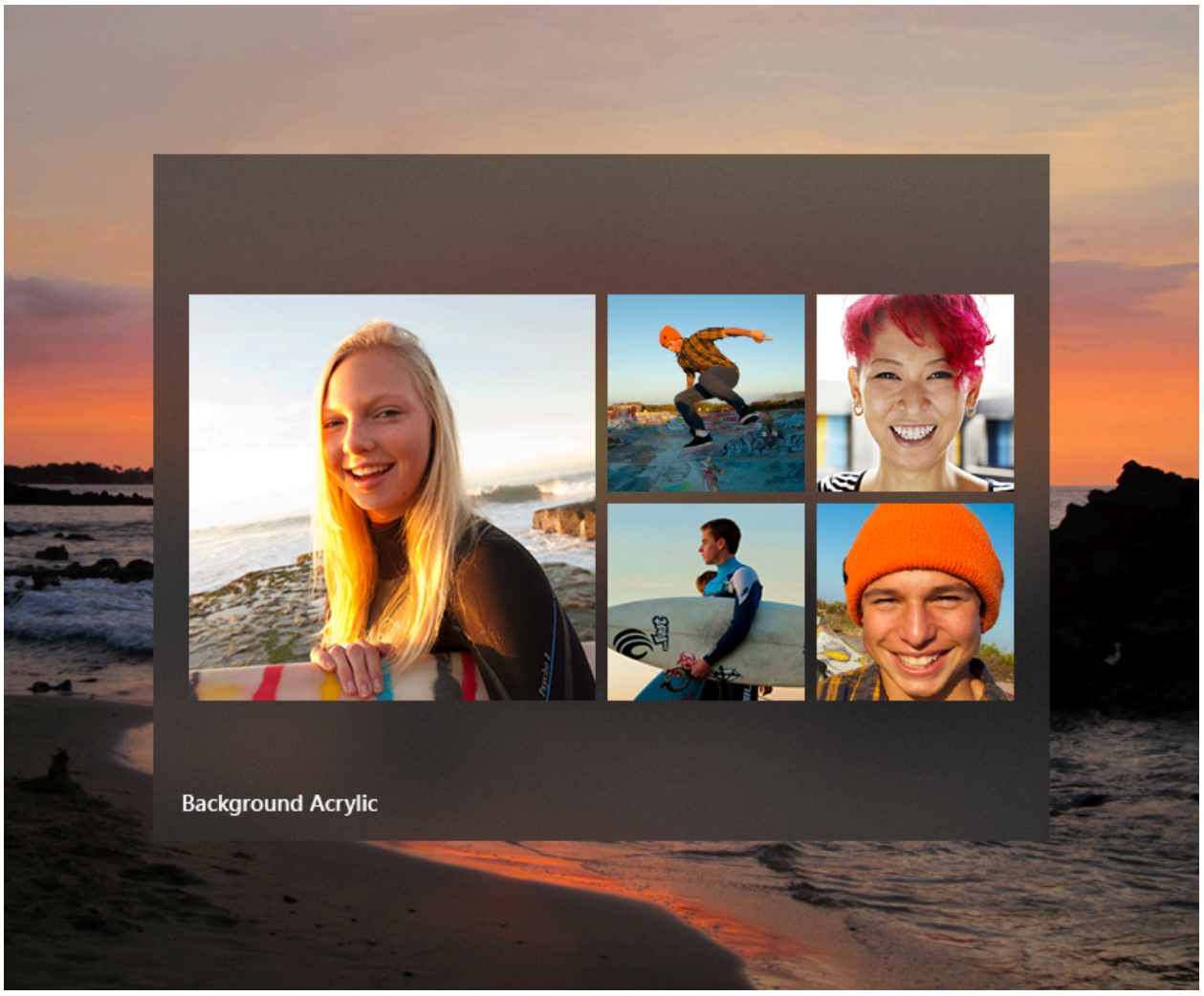
Acrylic and the Fluent Design System

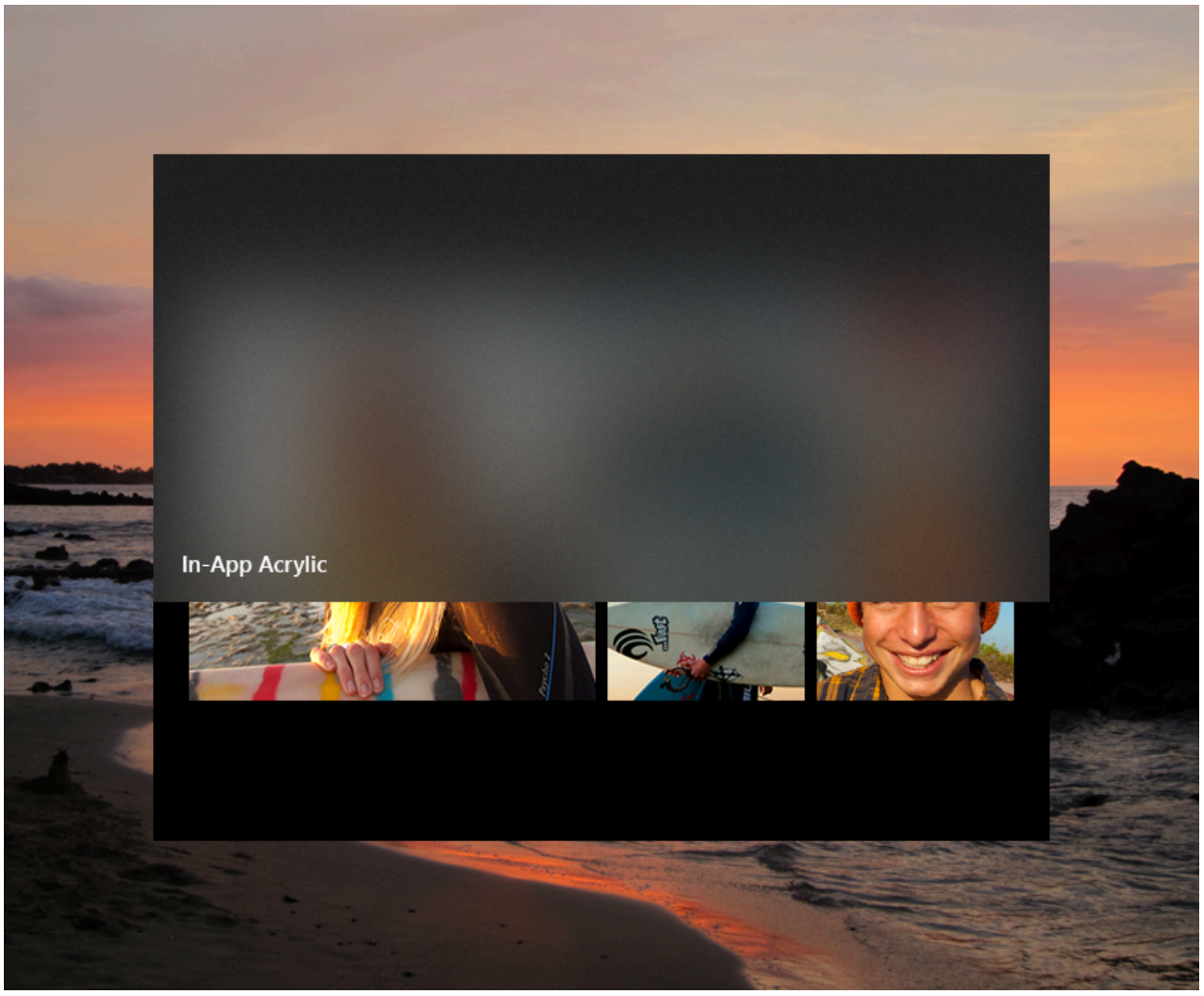
The Fluent Design System helps you create modern, bold UI that incorporates light, depth, motion, material, and scale. Acrylic is a Fluent Design System component that adds physical texture (material) and depth to your app. To learn more, see Fluent Design - Material.

Acrylic blend types

Acrylic's most noticeable characteristic is its transparency. There are two acrylic blend types that change what's visible through the material:

- **Background acrylic** reveals the desktop wallpaper and other windows that are behind the currently active app, adding depth between application windows while celebrating the user's personalization preferences.
- **In-app acrylic** adds a sense of depth within the app frame, providing both focus and hierarchy.


Background Acrylic



Avoid layering multiple acrylic surfaces: multiple layers of background acrylic can create distracting optical illusions.

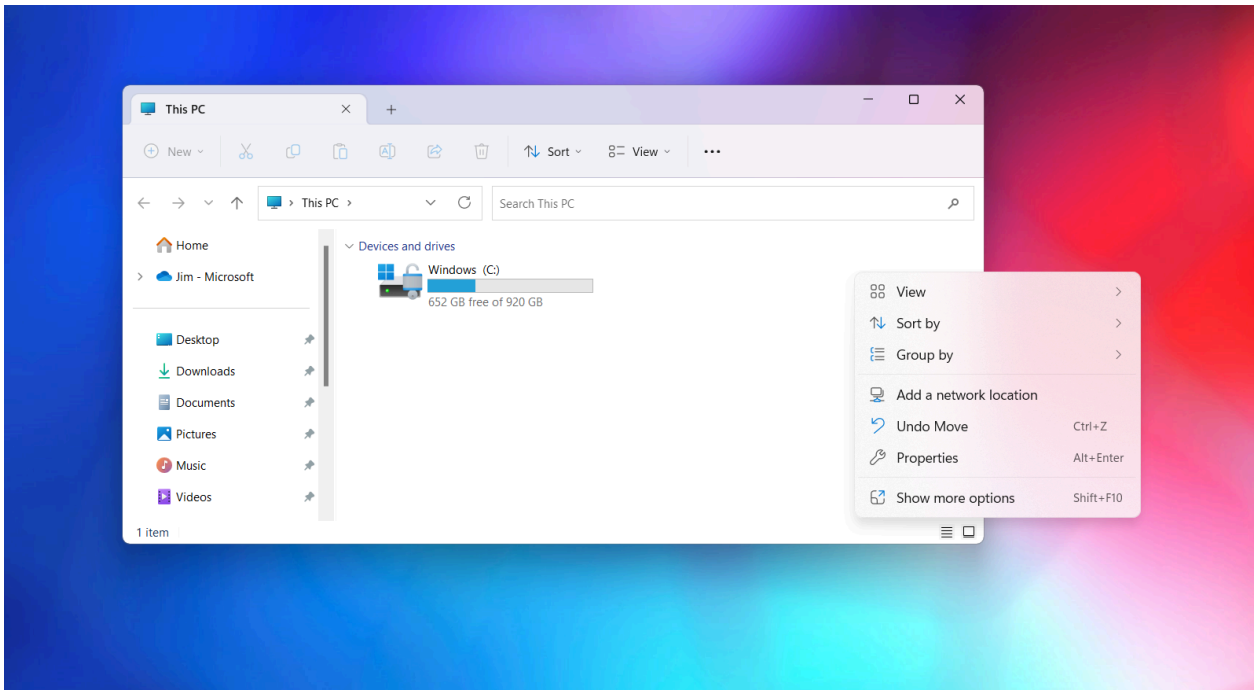
When to use acrylic

Consider the following usage patterns to decide how best to incorporate acrylic into your app.

Transient surfaces

- Use **background acrylic** for transient UI elements.

For apps with context menus, flyouts, non-modal popups, or light-dismiss panes, we recommend that you use background acrylic, especially if these surfaces draw outside the frame of the main app window. Using acrylic in transient scenarios helps maintain a visual relationship with the content that triggered the transient UI.



Many XAML controls draw acrylic by default. MenuFlyout, AutoSuggestBox, ComboBox, and similar controls with light-dismiss popups all use acrylic while open.

Supporting UI and vertical panes

- Use **in-app acrylic** for supporting UI, such as on surfaces that may overlap content when scrolled or interacted with.

If you are using in-app acrylic on navigation surfaces, consider extending content beneath the acrylic pane to improve the flow in your app. Using NavigationView will do this for you automatically. However, to avoid creating a striping effect, try not to place multiple pieces of acrylic edge-to-edge - this can create an unwanted seam between the two blurred surfaces. Acrylic is a tool to bring visual harmony to your designs, but when used incorrectly can result in visual noise.

For vertical panes or surfaces that help section off content of your app, we recommend you use an opaque background instead of acrylic. If your vertical panes open on top of content, like in NavigationView's **Compact** or **Minimal** modes, we suggest you use in-app acrylic to help maintain the page's context when the user has this pane open.

> **Note**
>
> Rendering acrylic surfaces is GPU-intensive, which can increase device power consumption and shorten battery life. Acrylic effects are automatically disabled when a device enters Battery Saver mode. Users can disable acrylic effects for all apps by turning off *Transparency effects* in Settings > Personalization > Colors.

Usability and adaptability

Acrylic automatically adapts its appearance for a wide variety of devices and contexts.

In High Contrast mode, users continue to see the familiar background color of their choosing in place of acrylic. In addition, both background acrylic and in-app acrylic appear as a solid color:

- When the user turns off *Transparency effects* in Settings > Personalization > Colors.
- When Battery Saver mode is activated.
- When the app runs on low-end hardware.

In addition, only background acrylic will replace its translucency and texture with a solid color:

- When an app window on desktop deactivates.
- When the app is running on Xbox, HoloLens, or in tablet mode.

Legibility considerations

It's important to ensure that any text your app presents to users meets contrast ratios (see Accessible text requirements). We've optimized the acrylic resources such that text meets contrast ratios on top of acrylic. We don't recommend placing accent-colored text on your acrylic surfaces because these combinations are likely to not pass minimum contrast ratio requirements at the default 14px font size. Try to avoid placing hyperlinks over acrylic elements. Also, if you choose to customize the acrylic tint color or opacity level, keep the impact on legibility in mind.

Apply acrylic in your app

> **ⓘ Important**
>
> How you apply background acrylic differs between WinUI 3 and WinUI 2/UWP.
>
> **WinUI 2/UWP**: The AcrylicBrush class has a BackgroundSource property that specifies whether you want background or in-app acrylic. You use an AcrylicBrush to apply both types of acrylic. See the AcrylicBrush classes for specific information and examples: Microsoft.UI.Xaml.Media.AcrylicBrush (WinUI 2), Windows.UI.Xaml.Media.AcrylicBrush (UWP).

WinUI 3: The AcrylicBrush class only supports in-app acrylic. You use the DesktopAcrylicBackdrop class to apply background acrylic.

Background acrylic

To apply background acrylic in a WinUI 3 app, set the element's `SystemBackdrop` property to an instance of DesktopAcrylicBackdrop. For more info, see Apply Mica or Acrylic materials in desktop apps for Windows 11.

These elements have a `SystemBackdrop` property:

- CommandBarFlyoutCommandBar.SystemBackdrop
- ContentIsland.SystemBackdrop
- DesktopWindowXamlSource.SystemBackdrop
- FlyoutBase.SystemBackdrop
- MenuFlyoutPresenter.SystemBackdrop
- Popup.SystemBackdrop
- Window.SystemBackdrop

In-app acrylic

You can apply in-app acrylic to your app's surfaces using a XAML AcrylicBrush or predefined `AcrylicBrush` theme resources.

WinUI includes a collection of brush theme resources that respect the app's theme and fall back to solid colors as needed. To paint a specific surface, apply one of the theme resources to element backgrounds just as you would apply any other brush resource.

XAML

```
<Grid Background="{ThemeResource AcrylicInAppFillColorDefaultBrush}">
```

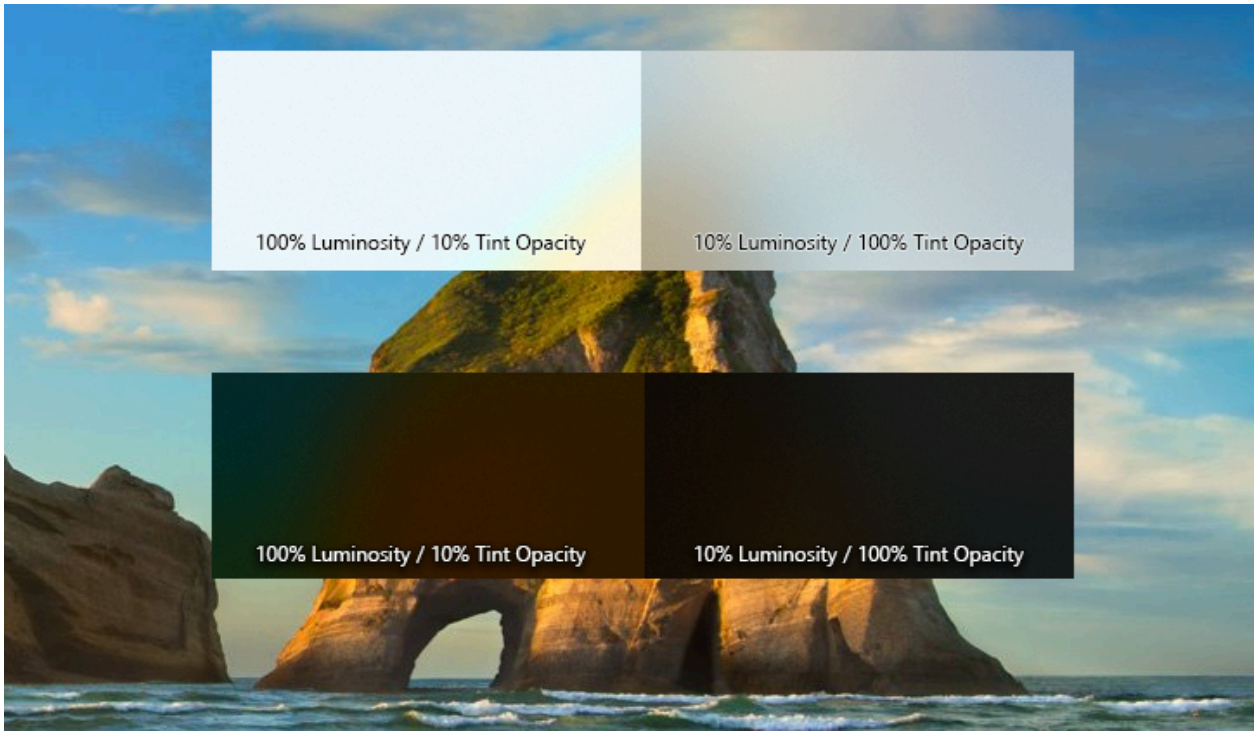
> **Note**
>
> You can view these resources in the AcrylicBrush theme resources file, in the microsoft-ui-xaml GitHub repo.
>
> For WinUI 2, resources that include *Background* in their names represent background acrylic, while *InApp* refers to in-app acrylic.

WinUI 3 includes the *Background* resources for compatibility, but they behave the same as *InApp* resources.

Custom acrylic brush

You may choose to add a color tint to your app's acrylic to show branding or provide visual balance with other elements on the page. To show color rather than grayscale, you'll need to define your own acrylic brushes using the following properties.

- **TintColor**: the color/tint overlay layer.
- **TintOpacity**: the opacity of the tint layer.
- **TintLuminosityOpacity**: controls the amount of saturation that is allowed through the acrylic surface from the background.
- **BackgroundSource**: (WinUI 2/UWP only) the flag to specify whether you want background or in-app acrylic.
- **FallbackColor**: the solid color that replaces acrylic in Battery Saver. For background acrylic, fallback color also replaces acrylic when your app isn't in the active desktop window.



To add an acrylic brush, define the three resources for dark, light, and high contrast themes. In high contrast, we recommend that you use a SolidColorBrush with the same `x:Key` as the dark/light AcrylicBrush.

> ⓘ **Note**

If you don't specify a `TintLuminosityOpacity` value, the system will automatically adjust its value based on your TintColor and TintOpacity.

XAML

```
<ResourceDictionary.ThemeDictionaries>
    <ResourceDictionary x:Key="Default">
        <AcrylicBrush x:Key="MyAcrylicBrush"
            BackgroundSource="HostBackdrop"
            TintColor="#FFFF0000"
            TintOpacity="0.8"
            TintLuminosityOpacity="0.5"
            FallbackColor="#FF7F0000"/>
    </ResourceDictionary>

    <ResourceDictionary x:Key="HighContrast">
        <SolidColorBrush x:Key="MyAcrylicBrush"
            Color="{ThemeResource SystemColorWindowColor}"/>
    </ResourceDictionary>

    <ResourceDictionary x:Key="Light">
        <AcrylicBrush x:Key="MyAcrylicBrush"
            BackgroundSource="HostBackdrop"
            TintColor="#FFFF0000"
            TintOpacity="0.8"
            TintLuminosityOpacity="0.5"
            FallbackColor="#FFFF7F7F"/>
    </ResourceDictionary>
</ResourceDictionary.ThemeDictionaries>
```

The following sample shows how to declare an AcrylicBrush in code.

C#

```
AcrylicBrush myBrush = new AcrylicBrush();
myBrush.TintColor = Color.FromArgb(255, 202, 24, 37);
myBrush.FallbackColor = Color.FromArgb(255, 202, 24, 37);
myBrush.TintOpacity = 0.6;

grid.Fill = myBrush;
```

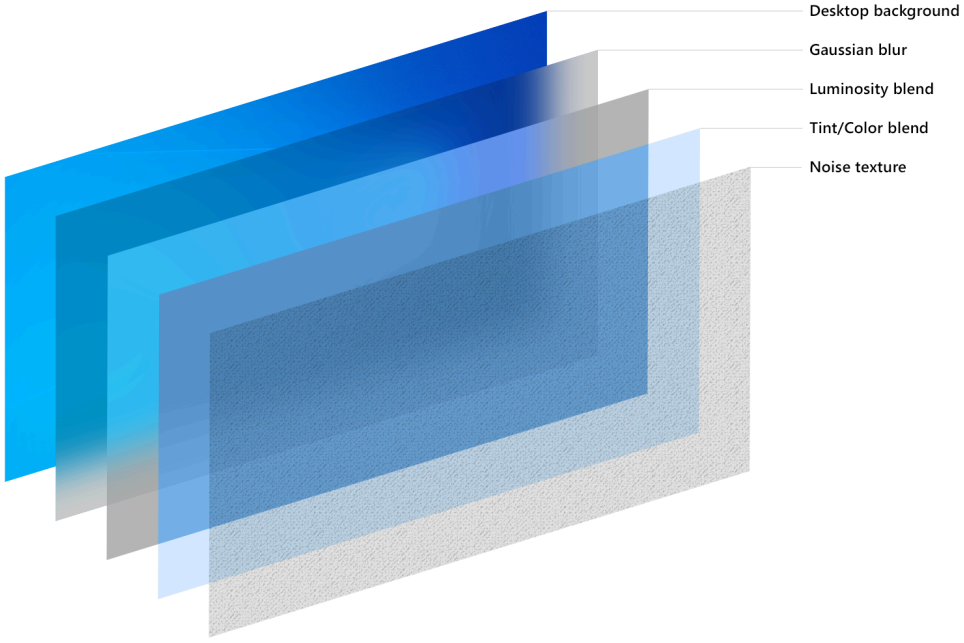
Do's and don'ts

- **Do** use acrylic on transient surfaces.
- **Do** extend acrylic to at least one edge of your app to provide a seamless experience by subtly blending with the app's surroundings.
- **Don't** put desktop acrylic on large background surfaces of your app.

- **Don't** place multiple acrylic panes next to each other because this results in an undesirable visible seam.
- **Don't** place accent-colored text over acrylic surfaces.

How we designed acrylic

We fine-tuned acrylic's key components to arrive at its unique appearance and properties. We started with translucency, blur, and noise to add visual depth and dimension to flat surfaces. We added an exclusion blend mode layer to ensure contrast and legibility of UI placed on an acrylic background. Finally, we added color tint for personalization opportunities. In concert these layers add up to a fresh, usable material.



The acrylic recipe: background, blur, exclusion blend, color/tint overlay, noise

Examples

Open the WinUI 3 Gallery app and see background Acrylic in action

Open the WinUI 3 Gallery app and see the in-app AcrylicBrush in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Related articles

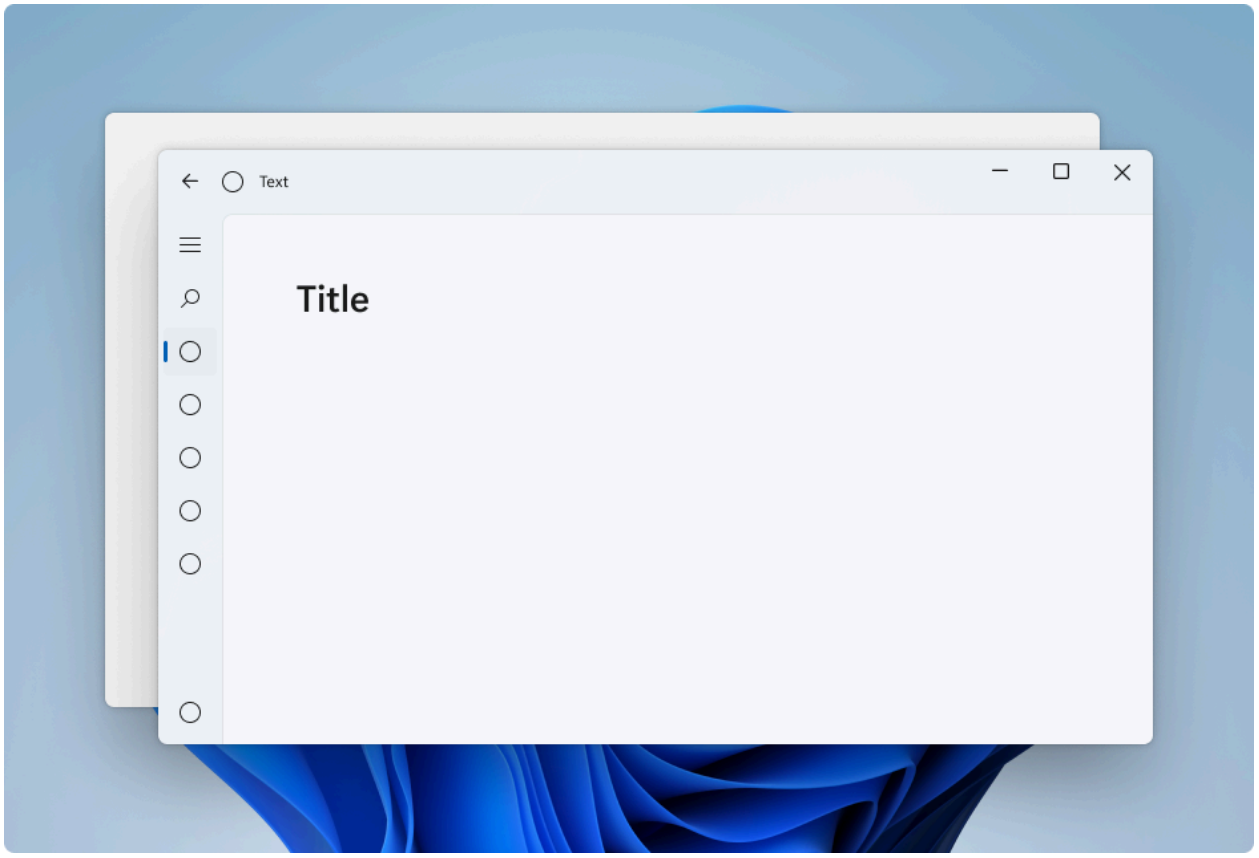
Fluent Design overview

Mica material

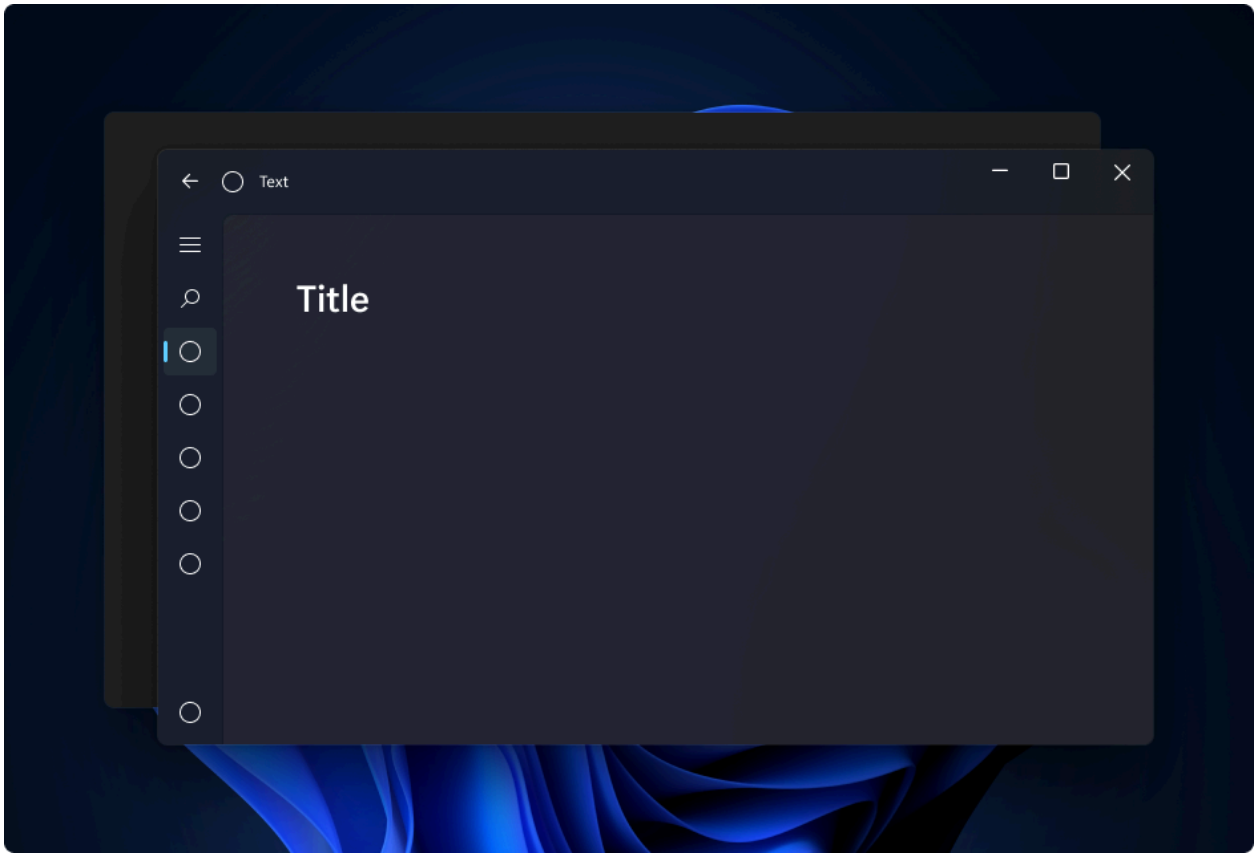
Article • 11/21/2024

Mica is an opaque, dynamic material that incorporates theme and desktop wallpaper to paint the background of long-lived windows such as apps and settings. You can apply Mica to your application backdrop to delight users and create visual hierarchy, aiding productivity, by increasing clarity about which window is in focus. Mica is specifically designed for app performance as it only samples the desktop wallpaper once to create its visualization. Mica is available for UWP apps that use WinUI 2 and apps that use Windows App SDK 1.1 or later, while running on Windows 11 version 22000 or later.

Mica in light theme



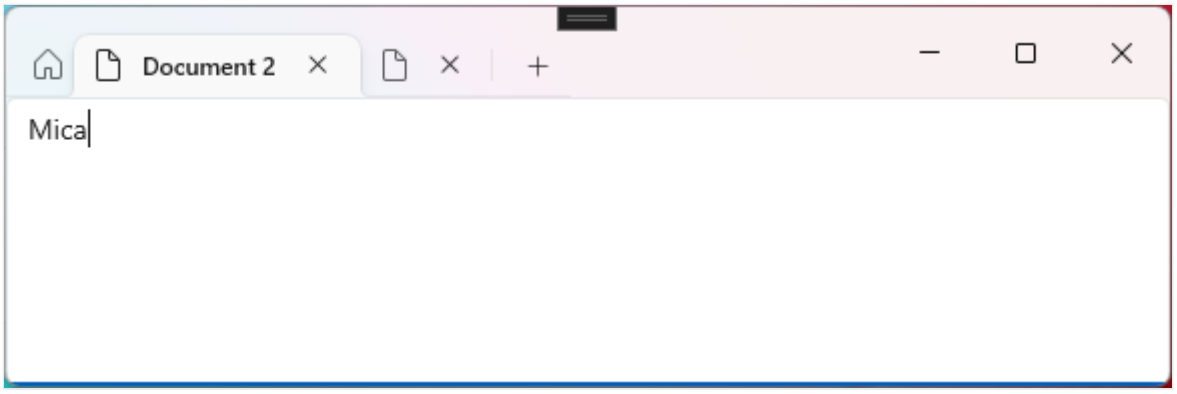
Mica in dark theme

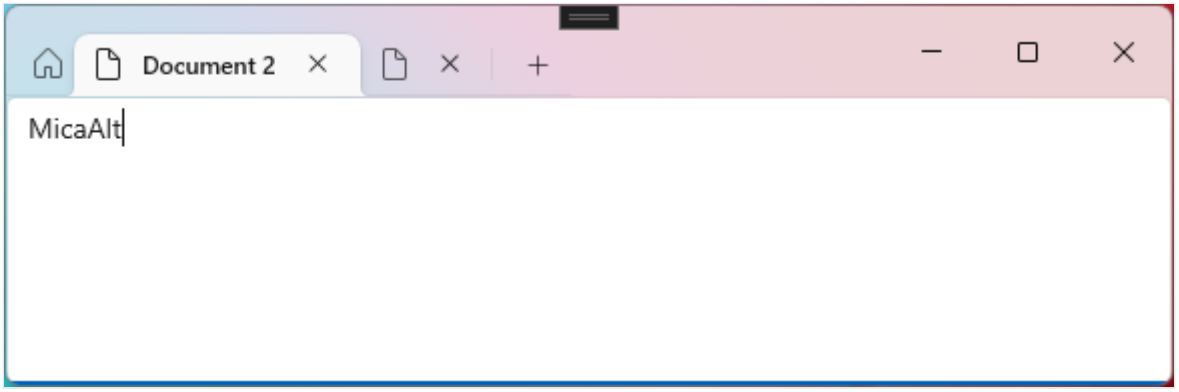


Mica Alt is a variant of Mica, with stronger tinting of the user's desktop background color. You can apply Mica Alt to your app's backdrop to provide a deeper visual hierarchy than Mica, especially when creating an app with a tabbed title bar. Mica Alt is

available for apps that use Windows App SDK 1.1 or later, while running on Windows 11 version 22000 or later.

These images show the difference between Mica and Mica Alt in a title bar with tabs. The first image uses Mica and the second image uses Mica Alt.





When to use Mica or Mica Alt

Mica and Mica Alt are materials that appear on the backdrop of your application — behind all other content. Each material is opaque and incorporates the user's theme and desktop wallpaper to create its highly personalized appearance. As the user moves the window across the screen, the Mica material dynamically adapts to create a rich visualization using the wallpaper underneath the application. In addition, the material helps users focus on the current task by falling back to a neutral color when the app is inactive.

We recommend that you apply Mica or Mica Alt as the base layer of your app, and prioritize visibility in the title bar area. For more specific layering guidance see Layering and Elevation and the App layering with Mica section of this article.

Usability and adaptability

The Mica materials automatically adapt their appearance for a wide variety of devices and contexts. They are designed for performance as they capture the background

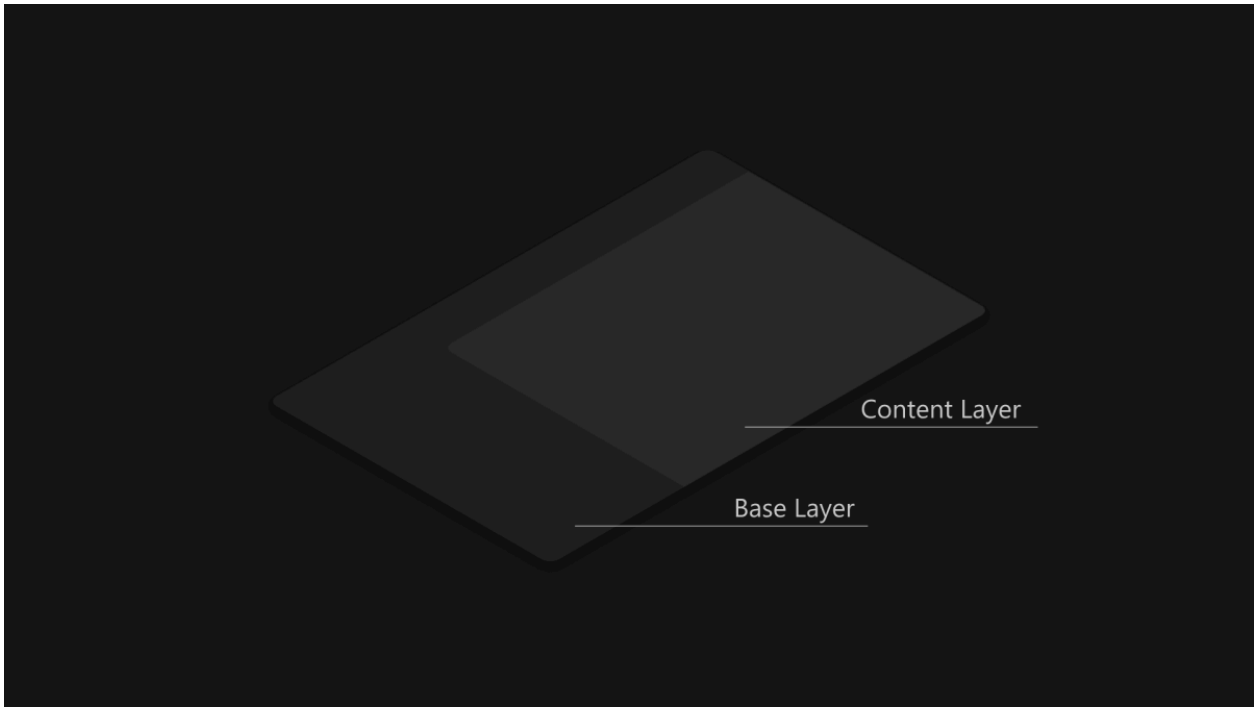
wallpaper only once to create their visualizations.

In High Contrast mode, users continue to see the familiar background color of their choosing in place of Mica or Mica Alt. In addition, the Mica materials will appear as a solid fallback color (`SolidBackgroundFillColorBase` for Mica, `SolidBackgroundFillColorBaseAlt` for Mica Alt) when:

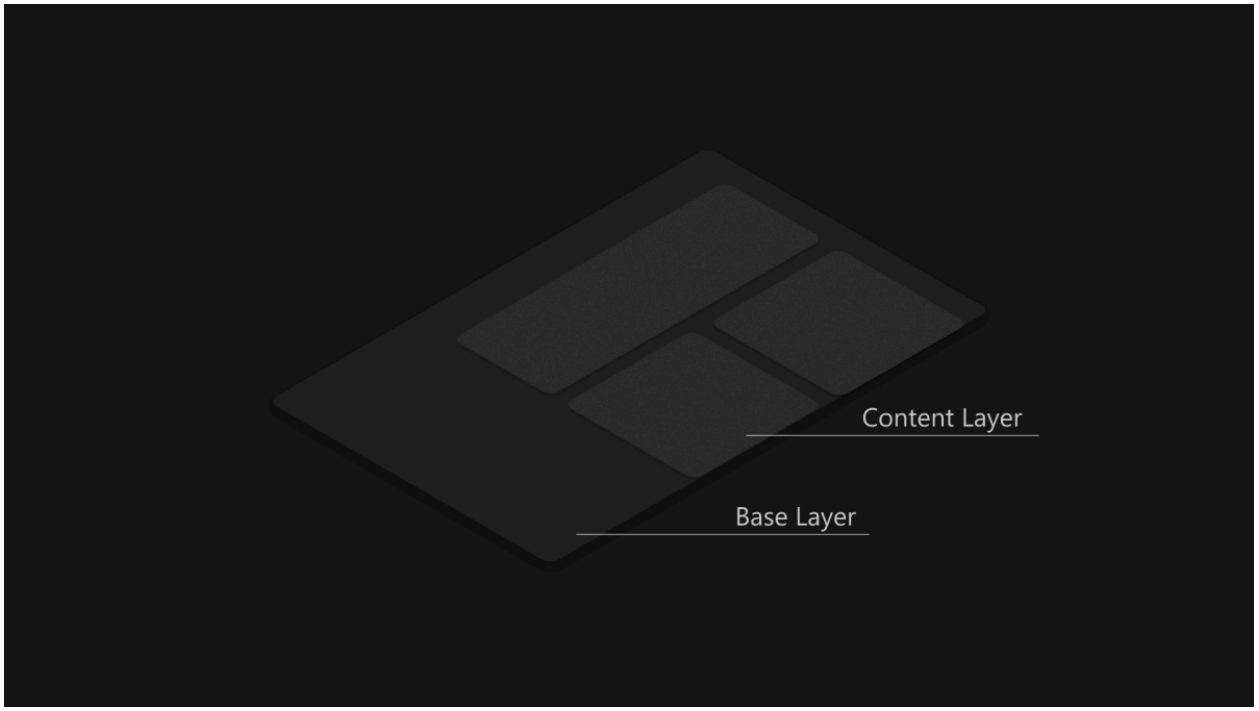
- The user turns off transparency in Settings > Personalization > Color.
- Battery Saver mode is activated.
- The app runs on low-end hardware.
- An app window on desktop deactivates.
- The Windows app is running on Xbox or HoloLens.
- The Windows version is below 22000.

App layering with Mica

Standard pattern content layer



Card pattern content layer



Mica is ideal as a foundation layer in your app's hierarchy due to its inactive and active states and subtle personalization. To follow the two-layer Layering and Elevation system, we encourage you to apply Mica as the base layer of your app and add an additional content layer that sits on top of the base layer. The content layer should pick up the material behind it, Mica, using the `LayerFillColorDefaultBrush`, a low-opacity solid color, as its background. Our recommended content layer patterns are:

- **Standard pattern**: A contiguous background for large areas that need a distinct hierarchical differentiation from the base layer. The `LayerFillColorDefaultBrush` should be applied to the container backgrounds of your WinUI app surfaces (e.g. Grids, StackPanels, Frames, etc.).
- **Card pattern**: Segmented cards for apps that are designed with multiple sectioned and discontinuous UI components. For the definition of the card UI using the `LayerFillColorDefaultBrush`, see Layering and Elevation guidance.

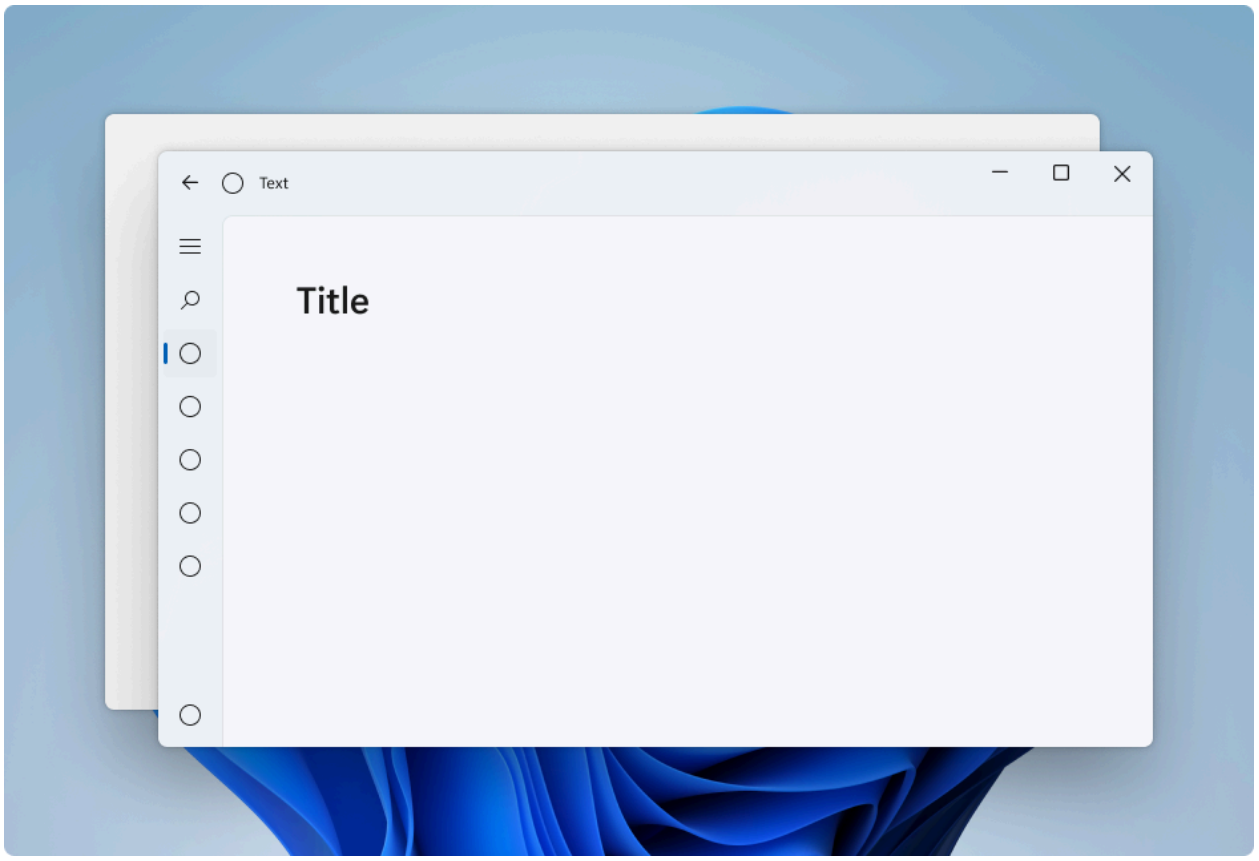
To give your app's window a seamless look, Mica should be visible in the title bar if you choose to apply the material to your app. You can show Mica in the title bar by extending your app into the non-client area and creating a transparent custom title bar. For more info, see Title bar.

The following examples showcase common implementations of the layering strategy with NavigationView where Mica is visible in the title bar area.

- Standard pattern in Left NavigationView.
- Standard pattern in Top NavigationView.
- Card pattern in Left NavigationView.

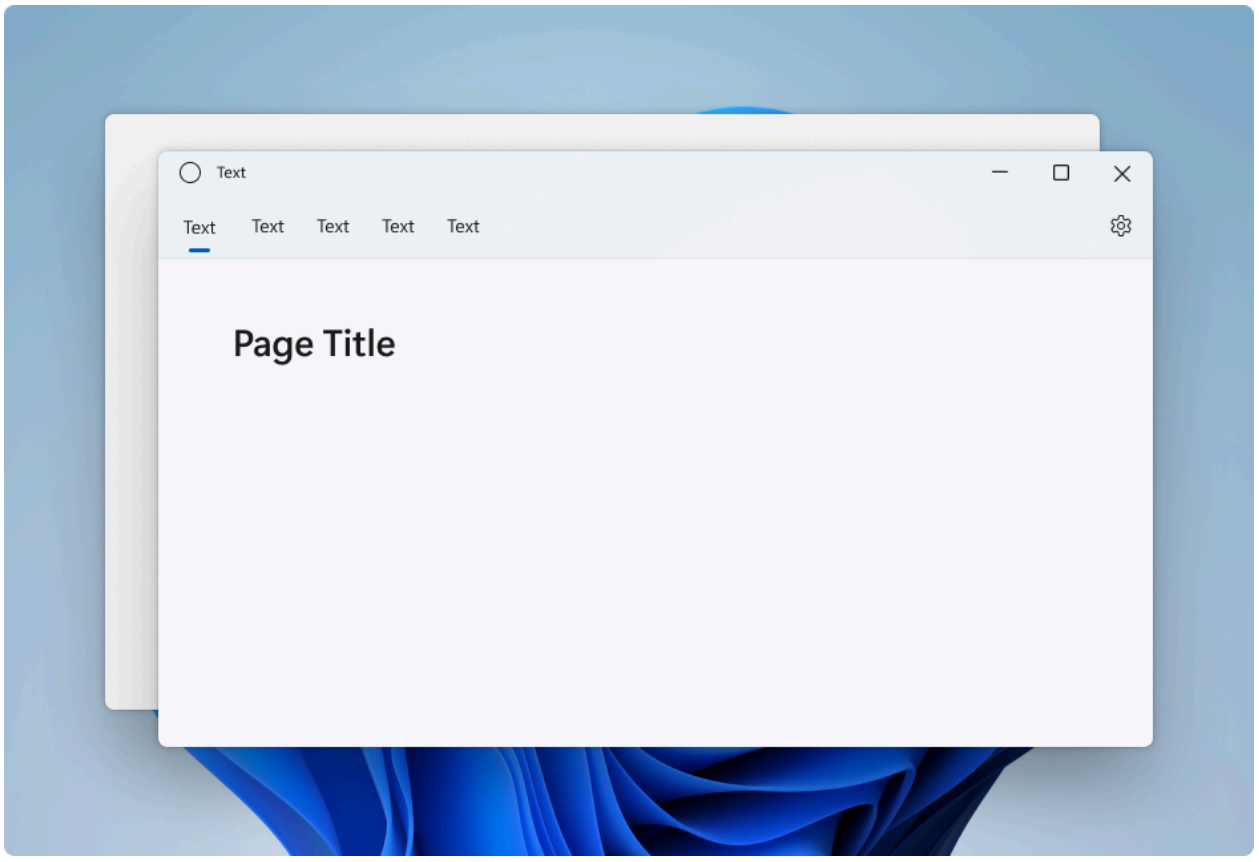
Standard pattern in Left NavigationView

By default, NavigationView in Left mode includes the content layer in its content area. This example extends Mica into the title bar area and creates a custom title bar.



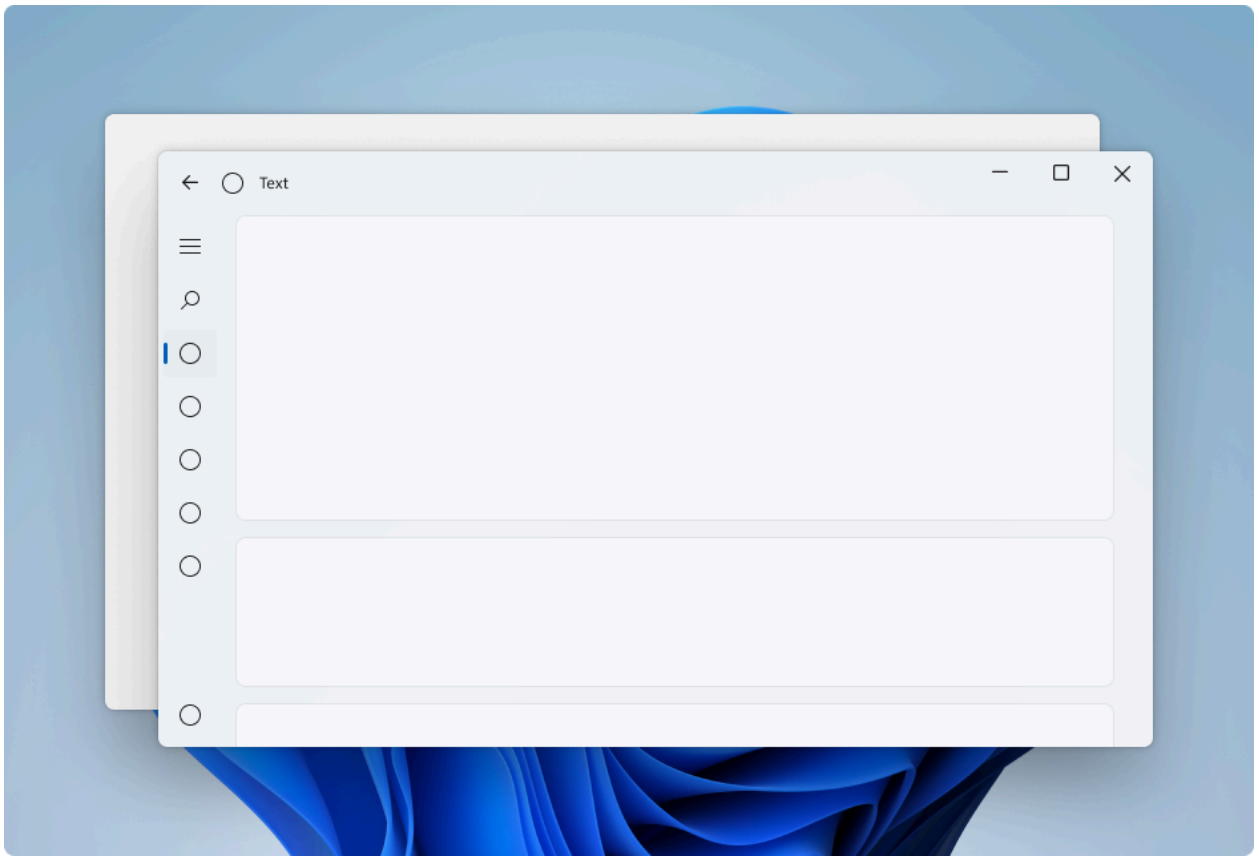
Standard pattern in Top NavigationView

By default, NavigationView in Top mode includes the content layer in its content area. This example extends Mica into the title bar area and creates a custom title bar.



Card pattern in Left NavigationView

To follow the card pattern using a NavigationView you will need to remove the default content layer by overriding the background and border theme resources. Then, you can create the cards in the content area of the control. This example creates several cards, extends Mica into the title bar area, and creates a custom title bar. For more information on card UI, see Layering and Elevation guidance.



App layering with Mica Alt

Mica Alt is an alternative to Mica as a foundation layer in your app's hierarchy with the same features like inactive and active states and subtle personalization. We encourage you to apply Mica Alt as the base layer of your app when you need contrast between title bar elements and the commanding areas of your app (e.g. navigation, menus).

A common scenario for using Mica Alt is when you are creating an application with a tabbed title bar. To follow the Layering and Elevation guidance we encourage you to apply Mica Alt as the base layer of your app, add a commanding layer that sits on top of the base layer, and finally add an additional content layer that sits on top of the commanding layer. The commanding layer should pick up the material behind it, Mica Alt, using the `LayerOnMicaBaseAltFillColorDefaultBrush`, a low-opacity solid color, as its background. The content layer should pick up the layers below it, using the `LayerFillColorDefaultBrush`, another low-opacity solid color. The layer system is as follows:

- **Mica Alt**: The base layer.
- **Commanding layer**: Requires distinct hierarchical differentiation from the base layer. The `LayerOnMicaBaseAltFillColorDefaultBrush` should be applied to the commanding areas of your WinUI app surfaces (e.g. MenuBar, navigation structure, etc.)

- **Content layer**: A contiguous background for large areas that need a distinct hierarchical differentiation from the commanding layer. The `LayerFillColorDefaultBrush` should be applied to the container backgrounds of your WinUI app surfaces (e.g. Grids, StackPanels, Frames, etc.).

To give your app's window a seamless look, Mica Alt should be visible in the title bar if you choose to apply the material to your app. You can show Mica Alt in the title bar by extending your app into the non-client area and creating a transparent custom title bar.

Recommendations

- **Do** set the background to *transparent* for all layers where you want to see Mica so the Mica shows through.
- **Don't** apply backdrop material more than once in an application.
- **Don't** apply backdrop material to a UI element. The backdrop material will not appear on the element itself. It will only appear if all layers between the UI element and the window are set to transparent.

Examples

Open the WinUI 3 Gallery app and see Mica in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

How to use Mica

You can use Mica in UWP apps that use WinUI 2, or in apps that use Windows App SDK 1.1 or later. You can use Mica Alt in apps that use Windows App SDK 1.1 or later.

Use Mica with the Windows App SDK

To use Mica in a WinUI 3 XAML app, see Apply Mica or Acrylic materials in desktop apps for Windows 11.

To use Mica in a Win32 app, see Apply Mica in Win32 desktop apps for Windows 11.

Use Mica with WinUI 2 for UWP

To use Mica in a UWP app with WinUI 2, see Apply Mica with WinUI 2 for UWP.

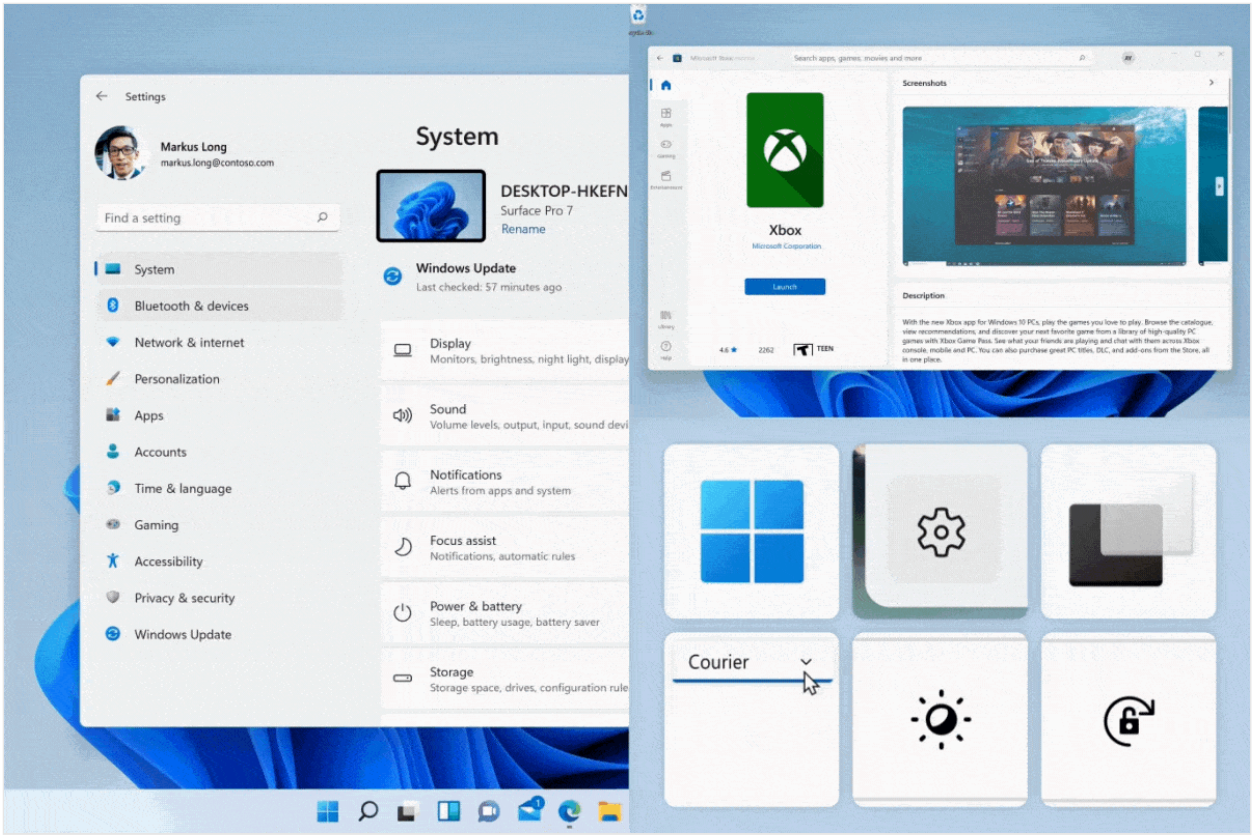
Related articles

- Materials
- Layering and Elevation
- Apply Mica or Acrylic materials in desktop apps for Windows 11
- Apply Mica in Win32 desktop apps for Windows 11
- NavigationView

Motion in Windows

Article • 11/05/2024

Motion describes the way the interface animates and responds to user interaction. Motion in Windows is reactive, direct, and context appropriate. It provides feedback to user input and reinforces spatial paradigms that support way-finding.



> **Tip**
>
> This article describes how the Fluent Design language is applied to Windows apps. For more information, see Fluent Design - Motion.

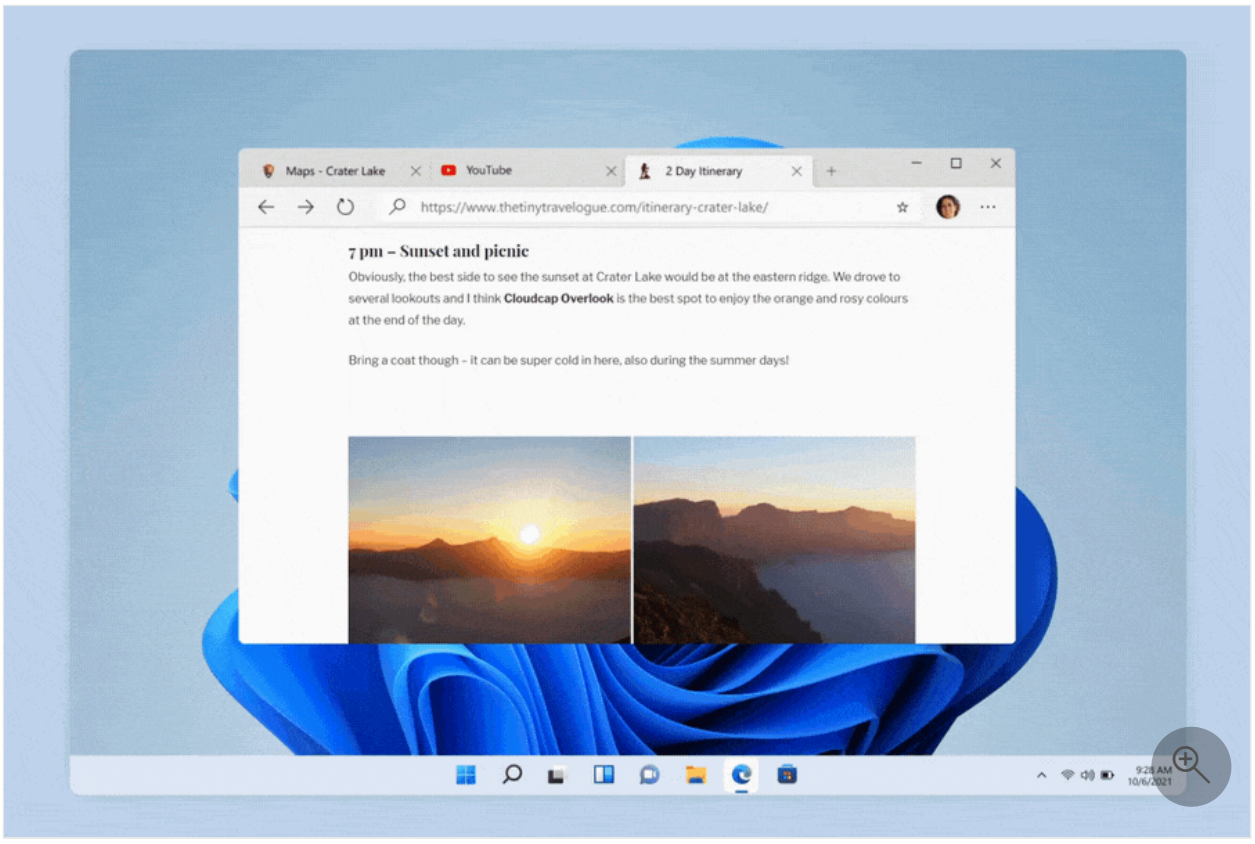
Motion principles

These principles guide the use of motion in Windows.

Connected: Elements of actions connect seamlessly

Elements that change position and size should visually connect from one state to another, even if they aren't connected under the hood. Users are guided to follow elements going from point to point, lowering the cognitive load of static state changes.

Example: When a window transitions between floating, snapped, and maximized, it always feels like the same window.

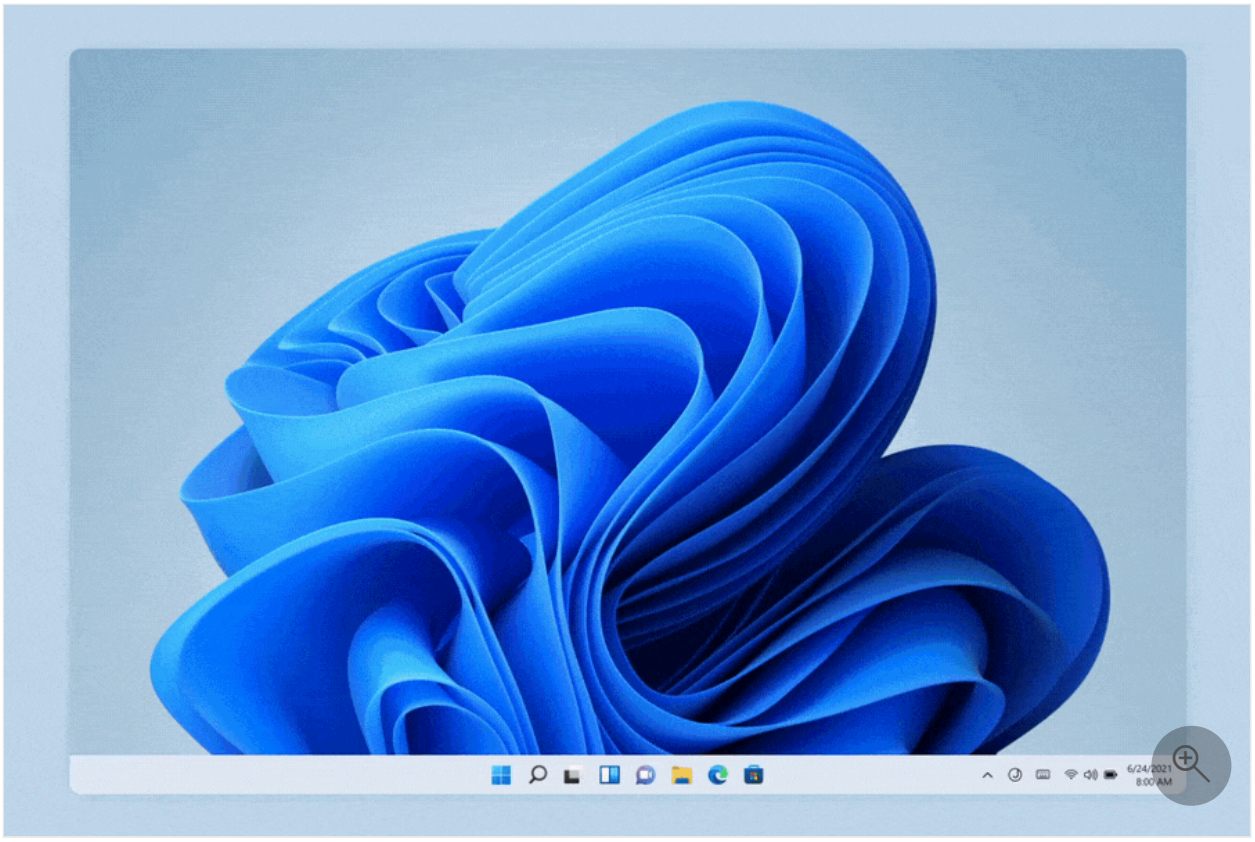


> **Tip**
>
> To improve accessibility and readability, this page uses still images in the default view. You can click an image to see the animated version.

Consistent: Elements should behave in similar ways when sharing entry points

Surfaces that share the same UI entry point should invoke and dismiss the same way to bring consistency to interactions. Each transition should respect the timing, easing, and direction of other elements so a surface feels cohesive.

Example: All taskbar flyouts slide up when invoked, and slide down when dismissed.

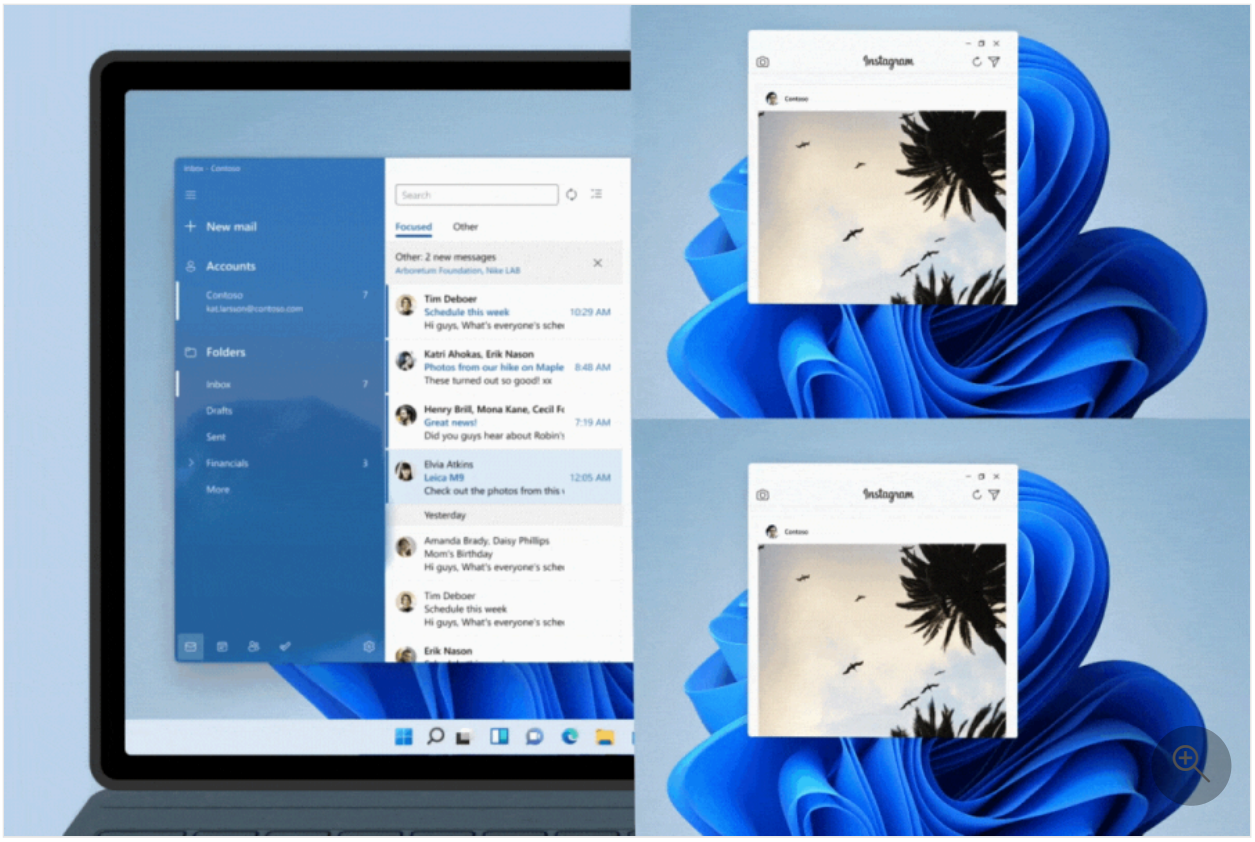


Click the image to see it animated.

Responsive: The system responds and adapts to user input and choices

Clear indicators show the system recognizes and adapts gracefully to different input, postures, and orientations. Apps should build on OS behaviors to feel responsive, alive, and aid usage depending on input methods.

Example: Taskbar icons spread out when keyboards are detached. Window edges invoke a different visual depending on cursor or touch input.

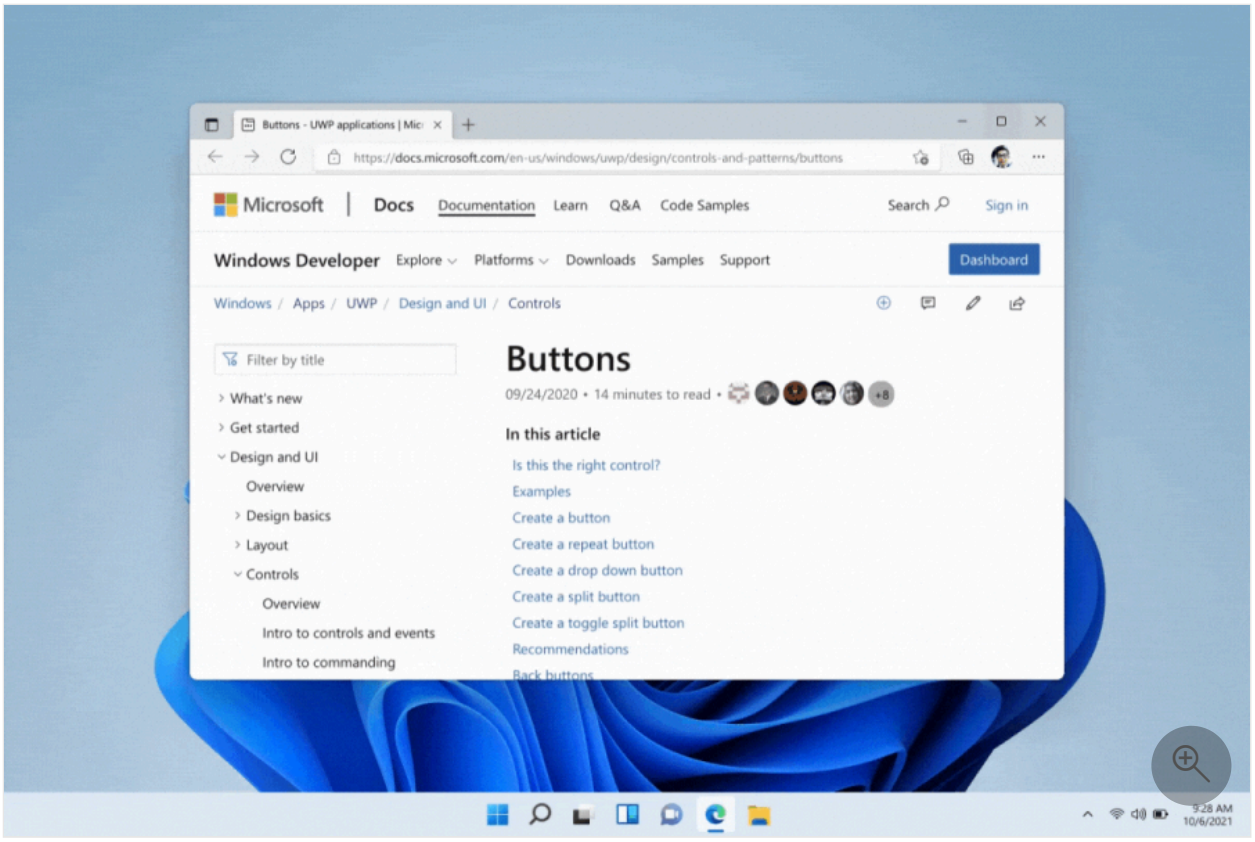


Click the image to see it animated.

Delightful: Unexpected moments of joy with purpose

Motion adds personality and energy to the experience in order to transform simple actions into moments of delight. These moments are always brief and fleeting, and help reinforce user actions.

Example: Minimizing a window causes an app icon to bounce down, while restoring bounces an app icon up.

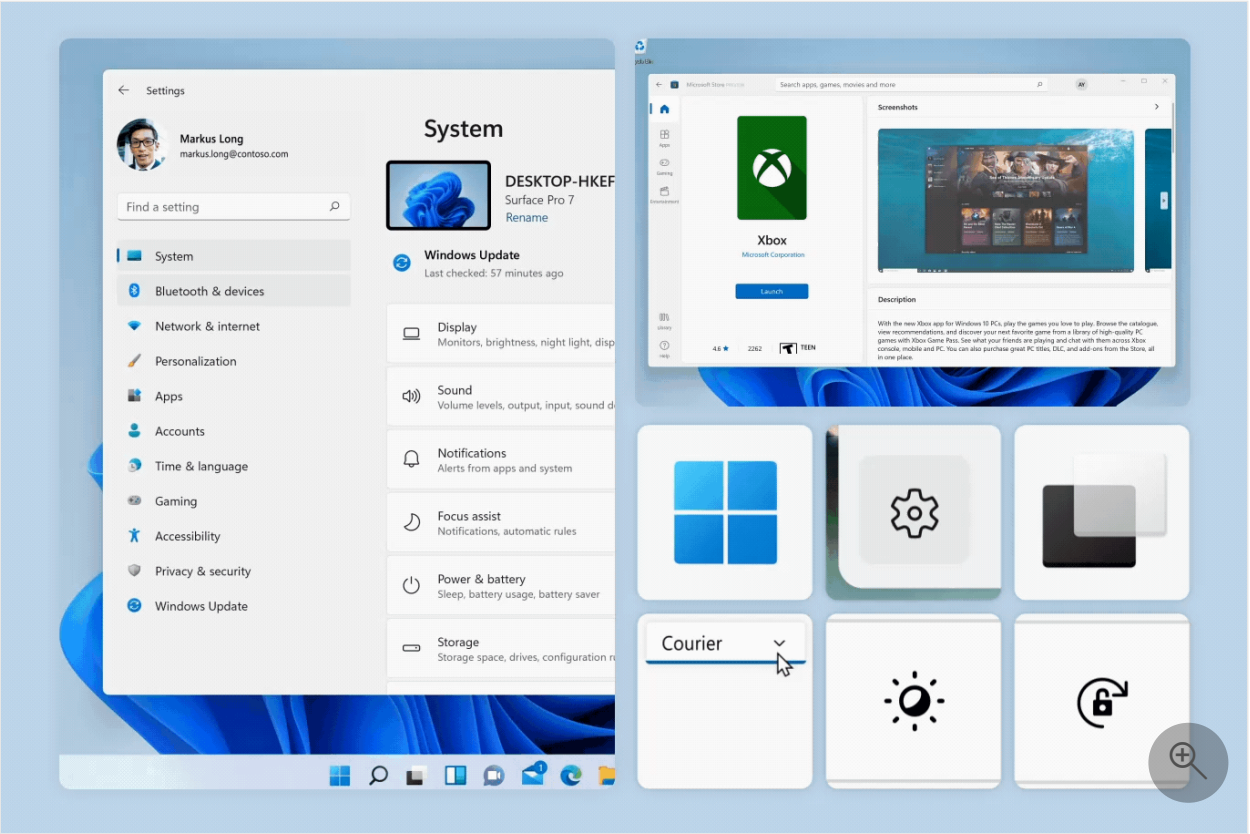


Click the image to see it animated.

Resourceful: Utilizes existing controls to bring consistency where possible

Avoid custom animations where possible. Use animation resources like WinUI controls for page transitions, in-page focus, and micro interactions. If you can't use WinUI controls, mimic existing OS behaviors based on where the app entry point lives.

Example: Page transitions, connected animations, and animated icons are the recommended WinUI controls that add delightful and necessary motion to apps.



Click the image to see it animated.

Examples

Open the WinUI 3 Gallery app and see Motion examples in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Usage

Animation properties

Windows motion is fast, direct, and context-appropriate. Timing and easing curves are adjusted based on the purpose of the animation to create a coherent experience.

Expand table

Purpose	Definition	Ease	Timing	Used For
Direct Entrance	Fast – In	Cubic-bezier(0,0,0,1)	167, 250, 333	Position, Scale, Rotation

Purpose	Definition	Ease	Timing	Used For
Existing Elements	Point to Point	Cubic-bezier(0.55,0.55,0,1)	167, 250, 333ms	Position, Scale, Rotation
Direct Exit	Fast – Out	Cubic-bezier(0,0,0,1)	167ms	Position, Scale, Rotation (ALWAYS combine with fade out)
Gentle Exit	Soft – Out	Cubic-bezier(1,0,1,1)	167ms	Position, Scale
Bare Minimum	Fade – In + Out	Linear	83ms	Opacity
Strong Entrance	Elastic In (3 Keyframes)	(3 values below)	(3 values below)	Position, Scale
	Keyframe 1	Cubic-Bezier(0.85, 0, 0, 1)	167ms	
	Keyframe 2	Cubic-Bezier(0.85, 0, 0.75, 1)	167ms	
	Keyframe 3	Cubic-Bezier(0.85, 0, 0, 1)	333ms	

Controls

This release of Windows introduces purposeful micro-interactions in WinUI controls. Add these controls to your app to help better organize information, and help your app's users transition from page to page, layer to layer, and state to state of an interaction.

Page Transition: Page-to-page transitions within the same surface

Use page transitions to transition smoothly from page to page, and configure animation directions to respect the flow of an app.

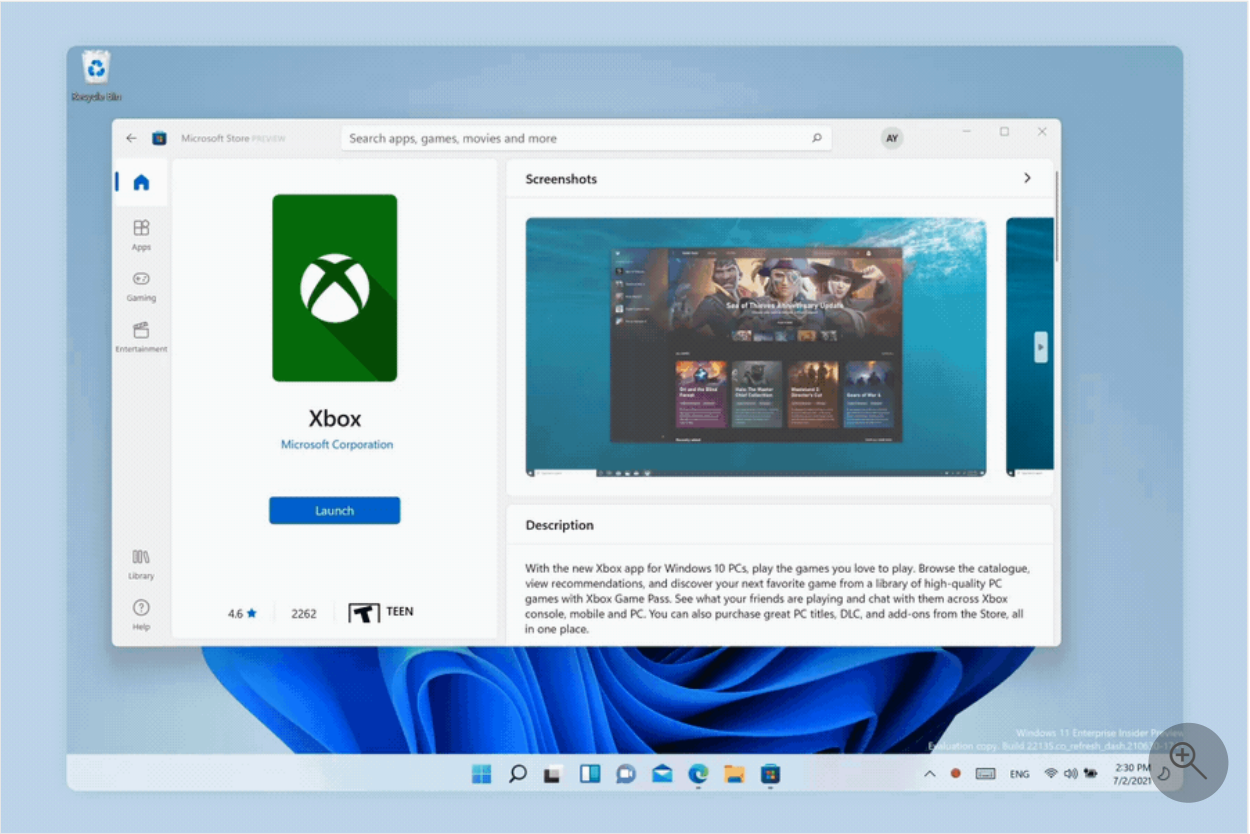
Page transitions guide your user's eyes to incoming and outgoing content, lowering cognitive load.

Click the image to see it animated.

Connected Animation: Layer-to-layer transitions within the same page

Use connected animations to highlight specific pieces of information within a page or surface, while retaining context.

Connected animations give focus to selected elements, and seamlessly transition between the focused and non-focused states.

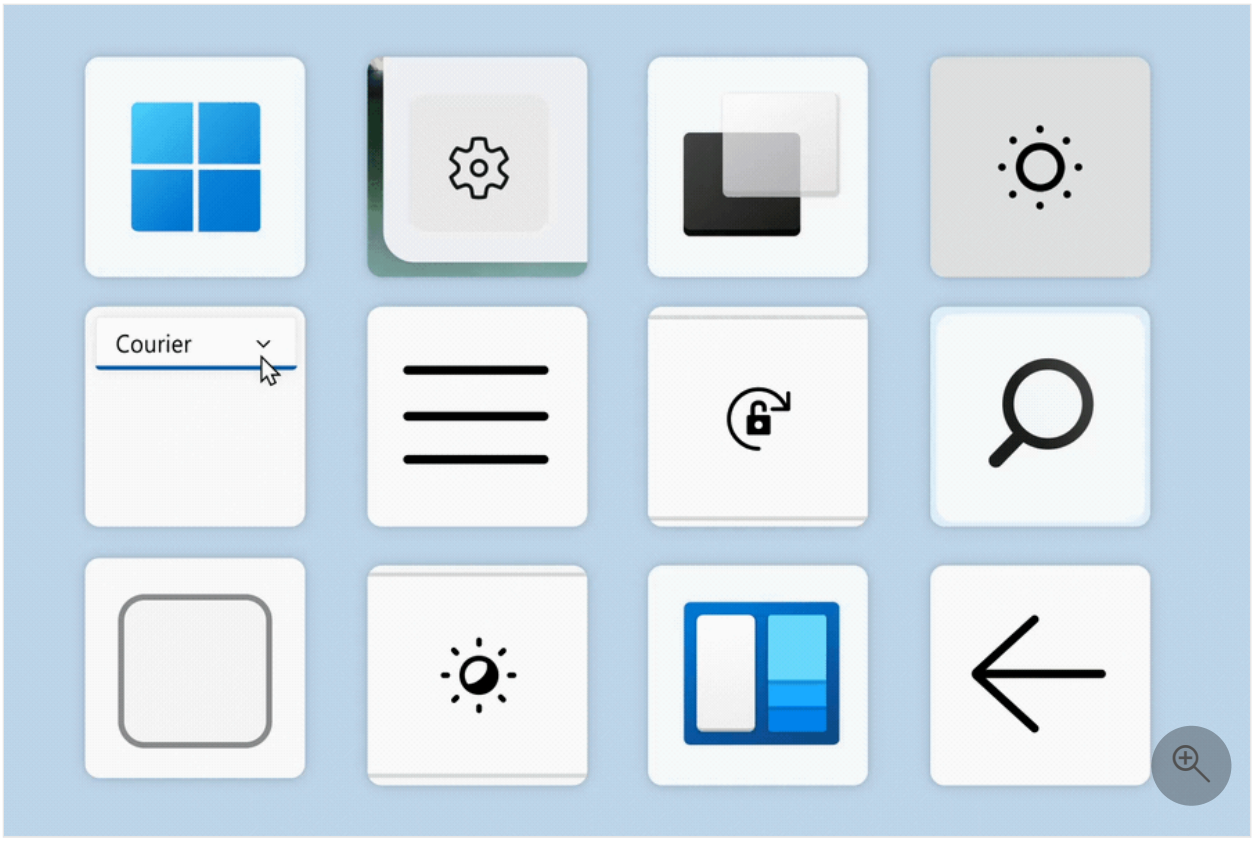


Click the image to see it animated.

Animated Icon: Adds delight and reveals information through micro interactions

Use animated icons to implement lightweight, vector-based icons and illustrations with motion using Lottie animations.

Animated icons draw attention to specific entry points, provide feedback from state to state, and add delight to an interaction.



Click the image to see it animated.

Timing and easing

Article • 11/05/2024

While motion is based in the real world, we are also a digital medium, which comes with an expectation of speed and performance.

Examples

Open the WinUI 3 Gallery app and see Easing Functions in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Standard animation duration values

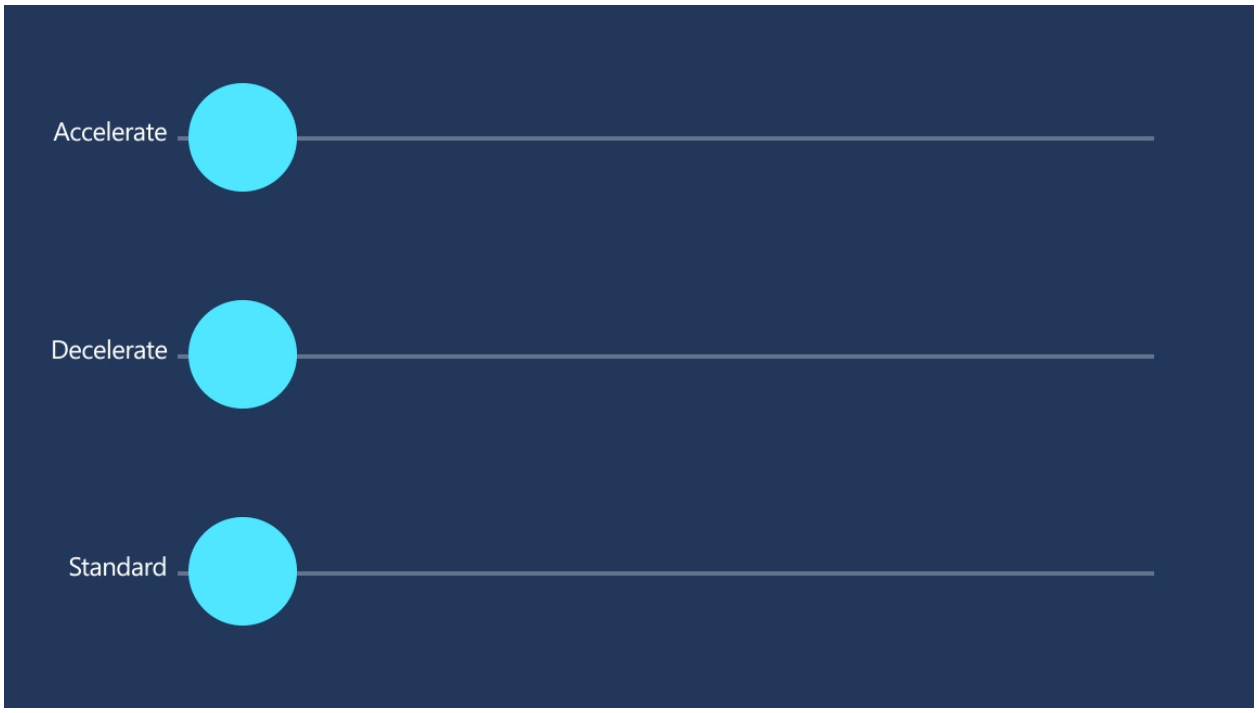
WinUI provides a set of standard animation durations that are used throughout the platform controls. You can use these named resources when building custom Storyboard animations as well.

Expand table

ThemeResource Name	Value
ControlNormalAnimationDuration	250ms
ControlFastAnimationDuration	167ms
ControlFasterAnimationDuration	83ms

Easing in Fluent motion

Easing is a way to manipulate the velocity of an object as it travels. It's the glue that ties together all the Fluent motion experiences. While extreme, the easing used in the system helps unify the physical feel of objects moving throughout the system. This is one way to mimic the real world, and make objects in motion feel like they belong in their environment.



Apply easing to motion

These easings will help you achieve a more natural feel, and are the baseline we use for Fluent motion.

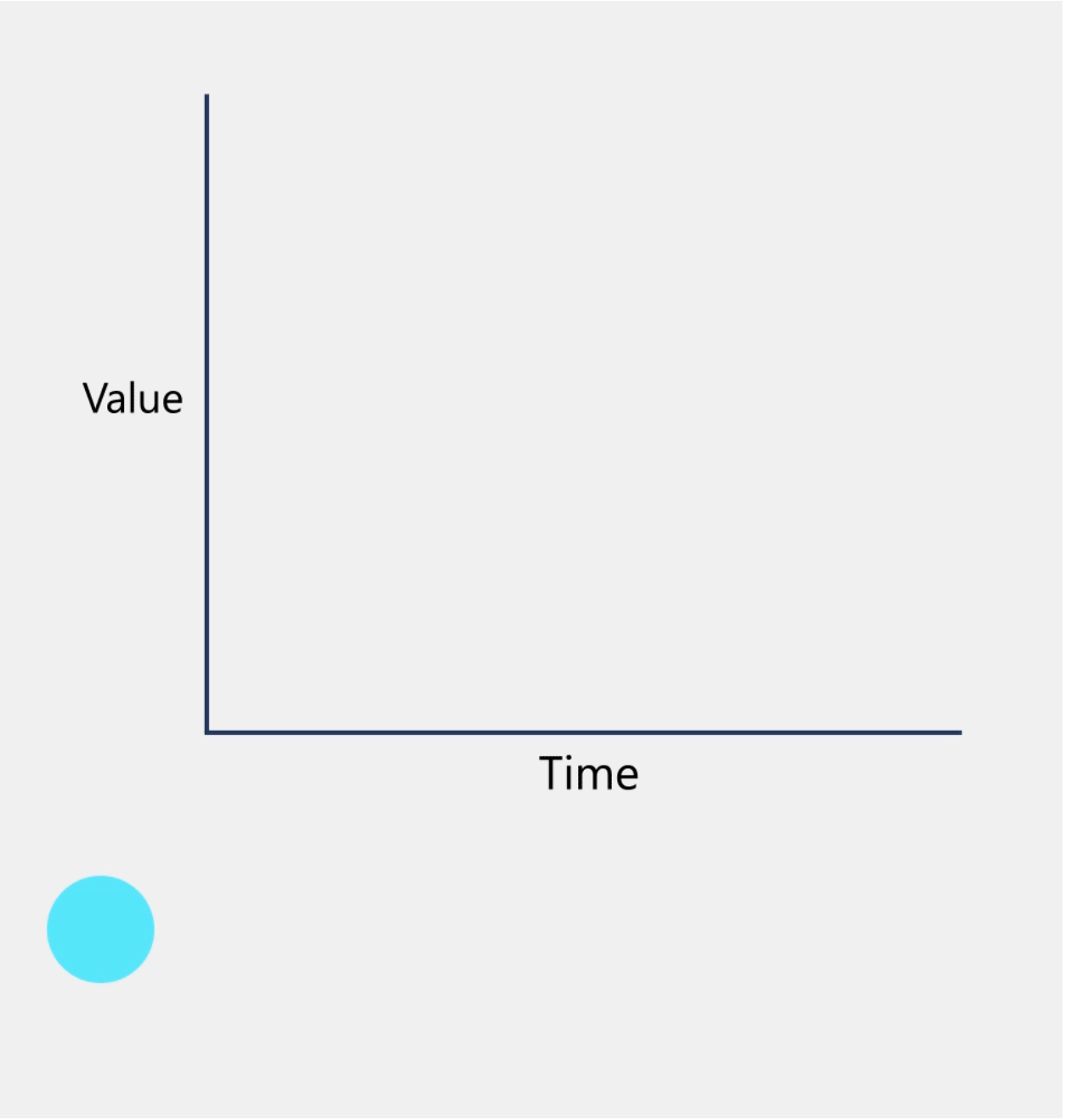
Fast Out, Slow In

```
C#
cubic-bezier(0, 0, 0, 1)
```

Use for objects or UI entering the scene, either navigating or spawning.

Once on-scene, the object is met with extreme friction, which slows the object to rest. The resulting feel is that the object traveled from a long distance away and entered at an extreme velocity, or is quickly returning to a rest state.

Even if it's preceded by a moment of unresponsiveness, the velocity of the incoming object has the effect of feeling fast and responsive.

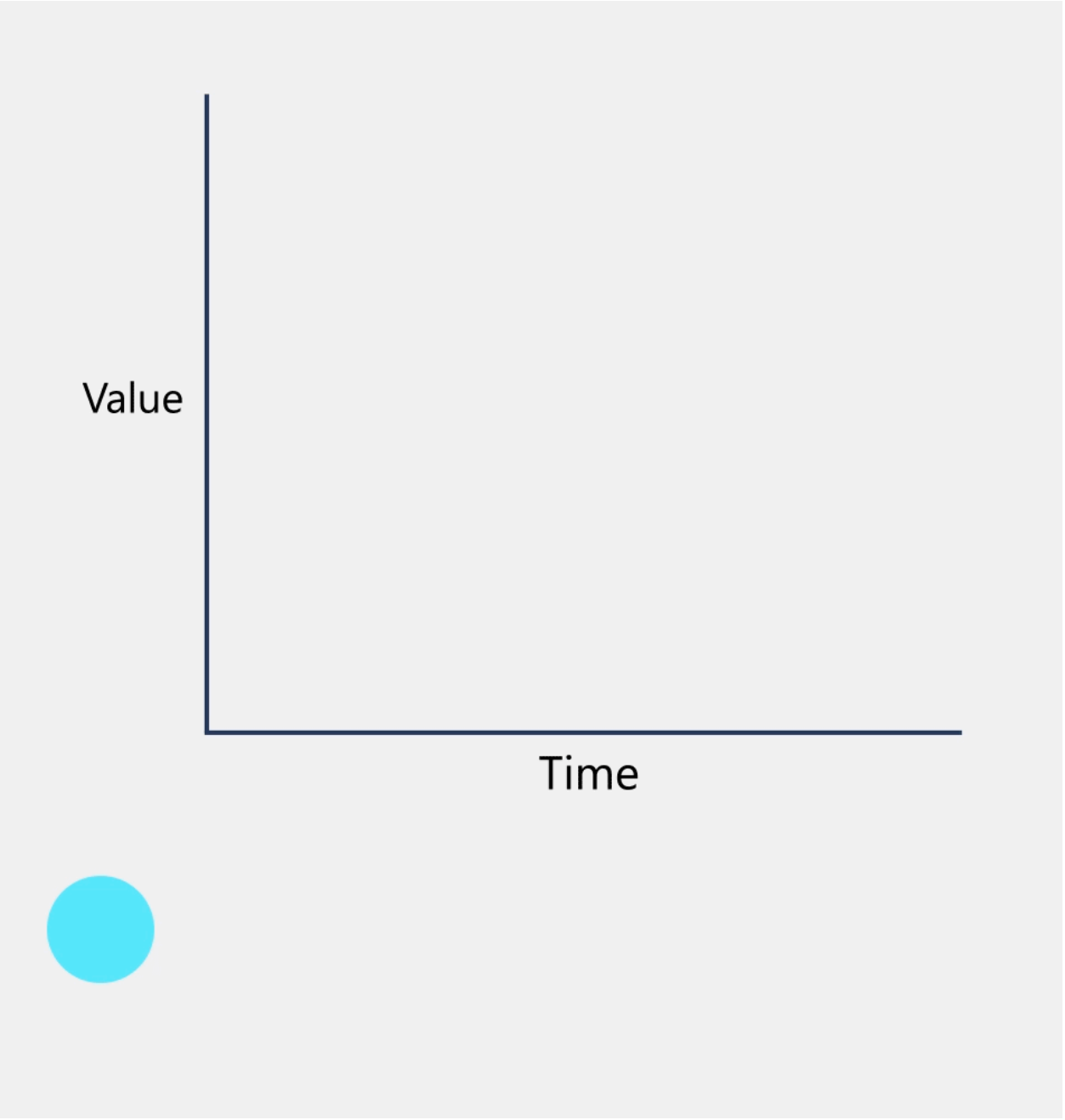


Slow Out, Fast In

```
C#
cubic-bezier(1 , 0 , 1 , 1)
```

Use for UI or objects that are exiting the scene.

Objects become powered and gain momentum until they reach escape velocity.The resulting feel is that the object is trying its hardest to get out of the user's way and make room for new content to come in.



Related articles

- Motion overview
- Fluent Design - Motion
- Directionality and gravity

Directionality and gravity

Article • 11/05/2024

Directional signals help to solidify the mental model of the journey a user takes across experiences. It is important that the direction of any motion support both the continuity of the space as well as the integrity of the objects in the space.

Directional movement is subject to forces like gravity. Applying forces to movement reinforces the natural feel of the motion.

Direction of movement

Direction of movement corresponds to physical motion. Just like in nature, objects can move in any world axis - X,Y,Z. This is how we think of the movement of objects on the screen. When you move objects, avoid unnatural collisions. Keep in mind where objects come from and go to, and always support higher level constructs that may be used in the scene, such as scroll direction or layout hierarchy.

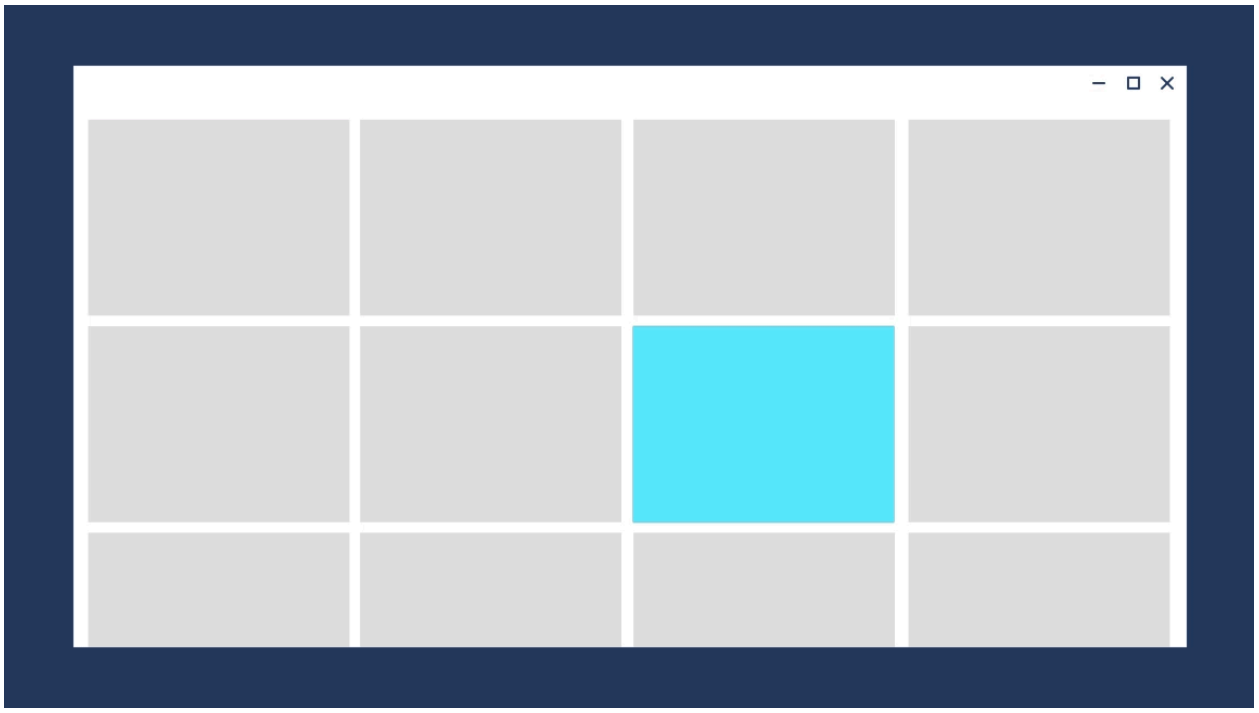
Direction of navigation

The direction of navigation between scenes in your app is conceptual. Users navigate forward and back. Scenes move in and out of view. These concepts combine with physical movement to guide the user.

When navigation causes an object to travel from the previous scene to the new scene, the object makes a simple A-to-B move on the screen. To ensure that the movement feels more physical, the standard easing is added, as well as the feeling of gravity.

For back navigation, the move is reversed (B-to-A). When the user navigates back, they have an expectation to be returned to the previous state as soon as possible. The timing is quicker, more direct, and uses the decelerate easing.

Here, these principles are applied as the selected item stays on screen during forward and back navigation.



When navigation causes items on the screen to be replaced, its important to show where the exiting scene went to, and where the new scene is coming from.

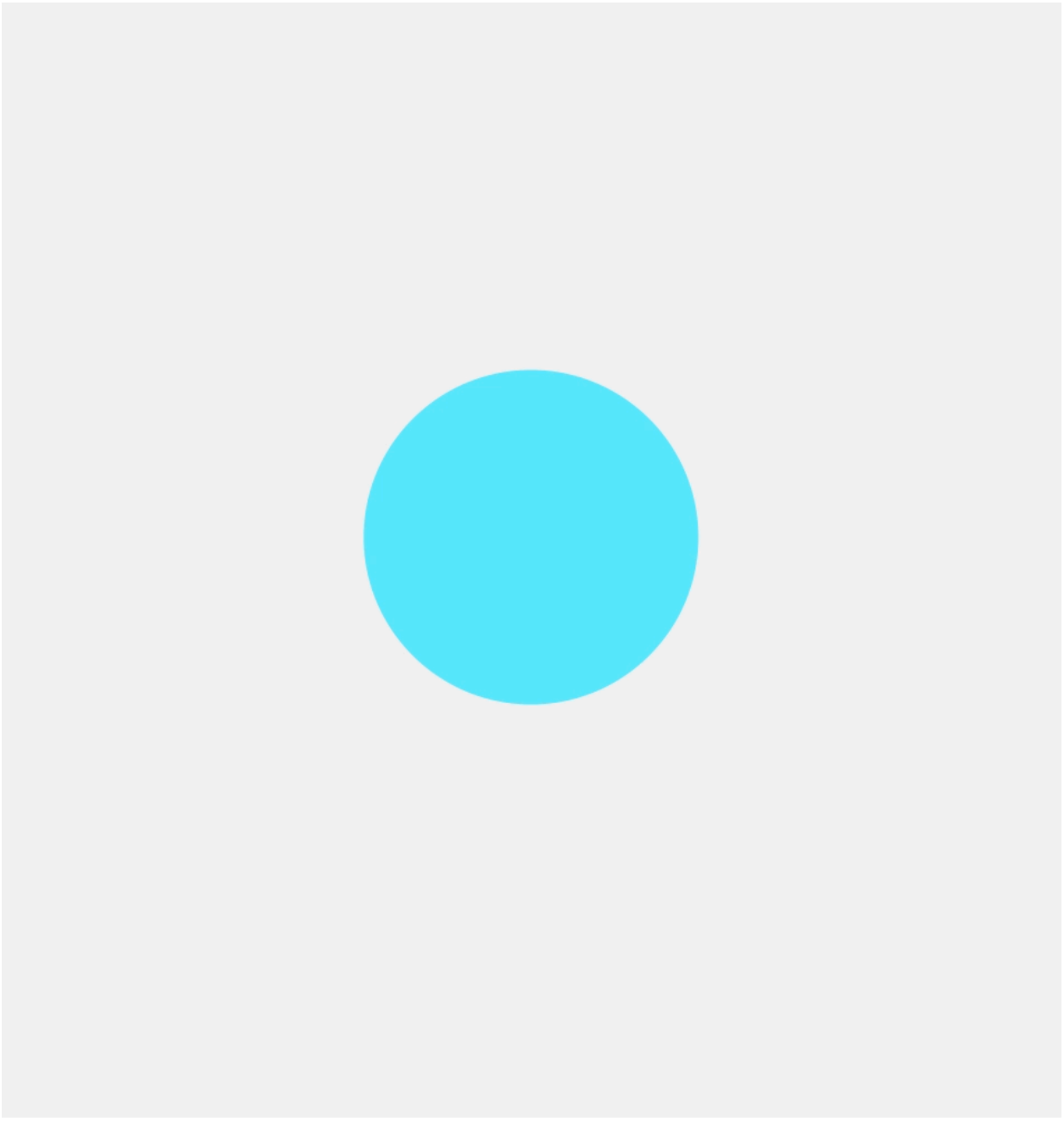
This has several benefits:

- It solidifies the user's mental model of the space.
- The duration of the exiting scene provides more time to prepare content to be animated in for the incoming scene.
- It improves the perceived performance of the app.

There are 4 discreet directions of navigation to consider.

Forward-In Celebrate content entering the scene in a manner that does not collide with outgoing content. Content decelerates into the scene.

Forward-Out Content exits quickly. Objects accelerate off screen.



Backward-In Same as Forward-In, but reversed.

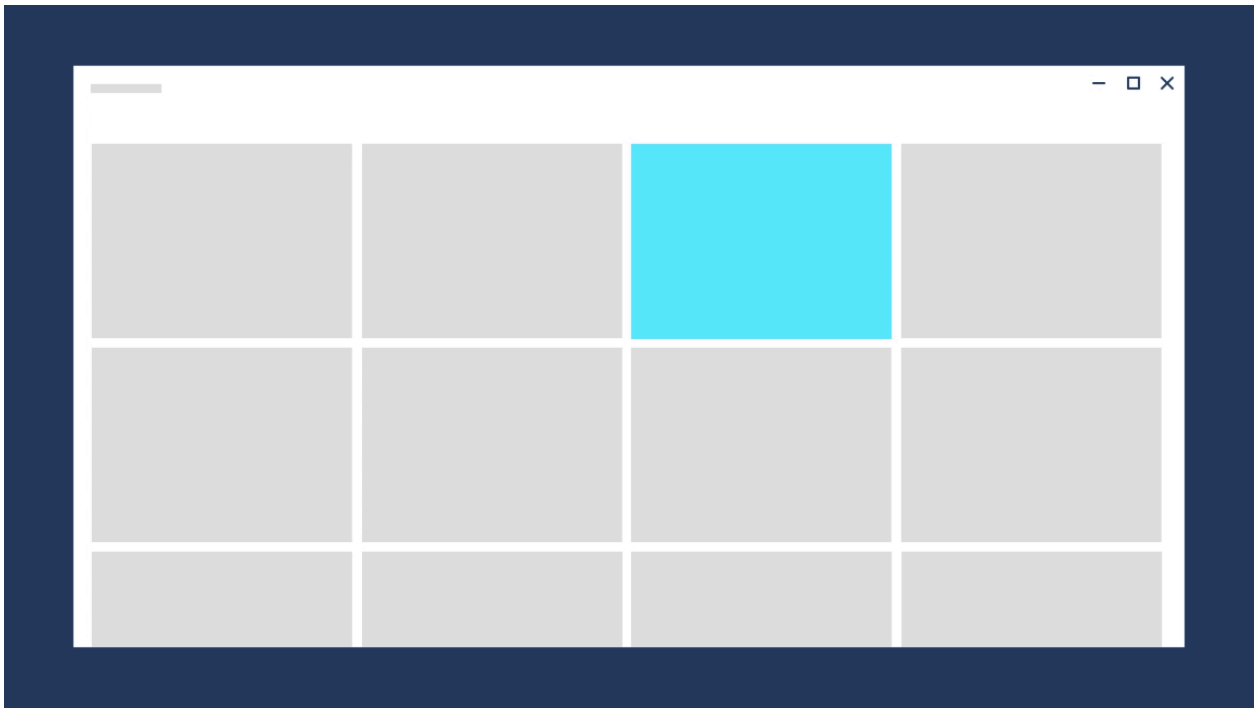
Backward-Out Same as Forward-Out, but reversed.

Gravity

Gravity makes your experiences feel more natural. Objects that move on the Z-axis and are not anchored to the scene by an onscreen affordance have the potential to be affected by gravity. As an object breaks free of the scene and before it reaches escape velocity, gravity pulls down on the object, creating a more natural curve of the object trajectory as it moves.

Gravity typically manifests when an object must jump from one scene to another. Because of this, connected animation uses the concept of gravity.

Here, an element in the top row of the grid is affected by gravity, causing it to drop slightly as it leaves its place and moves to the front.



Related articles

- Motion overview
- Timing and easing

Bringing it together

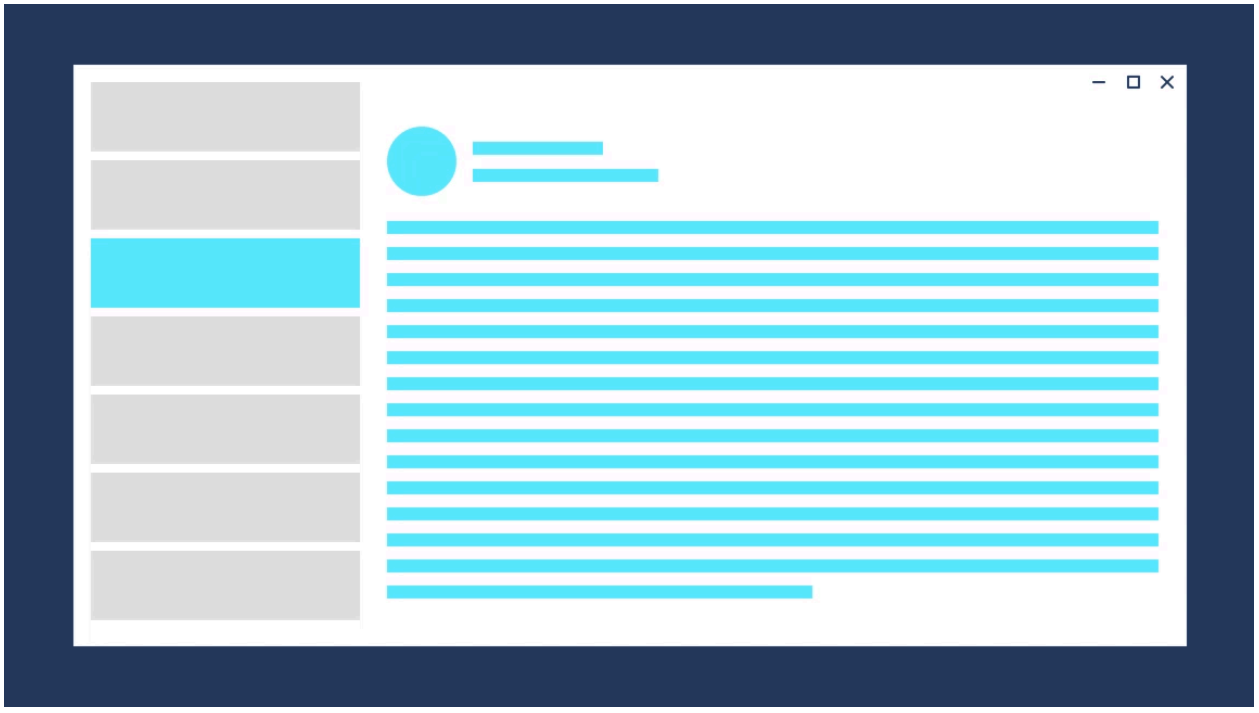
Article • 11/05/2024

Timing, easing, directionality, and gravity work together to form the foundation of Fluent motion. Each has to be considered in the context of the others, and applied appropriately in the context of your app.

Here are 3 ways to apply Fluent motion fundamentals in your app.

- **Implicit animation**
  Automatic tween and timing between values in a parameter change to achieve very simple Fluent motion using the standardized values.
- **Built-in animation**
  System components, such as common controls and shared motion, are "Fluent by default". Fundamentals have been applied in a manner consistent with their implied usage.
- **Custom animation following guidance recommendations**
  There may be times when the system does not yet provide an exact motion solution for your scenario. In those cases, use the baseline fundamental recommendations as a starting point for your experiences.

Transition example



Direction Forward Out:
Fade out: 150m; Easing: Default Accelerate **Direction Forward In:**
Slide up 150px: 300ms; Easing: Default Decelerate

Direction Backward Out:
Slide down 150px: 150ms; Easing: Default Accelerate **Direction Backward In:**
Fade in: 300ms; Easing: Default Decelerate

Object example

Direction Expand:
Grow: 300ms; Easing: Standard

Direction Contract:
Grow: 150ms; Easing: Default Accelerate

Examples

Open the WinUI 3 Gallery app and see Implicit Transitions in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Implicit Animations

Implicit animations are a simple way to achieve Fluent motion by automatically interpolating between the old and new values during a parameter change.

You can implicitly animate changes to the following properties:

- UIElement
  - **Opacity**
  - **Rotation**
  - **Scale**
  - **Translation**

- Border, ContentPresenter, or Panel
  - **Background**

Each property that can have changes implicitly animated has a corresponding *transition* property. To animate the property, you assign a transition type to the corresponding *transition* property. This table shows the *transition* properties and the transition type to use for each one.

Expand table

Animated property	Transition property	Implicit transition type
UIElement.Opacity	OpacityTransition	ScalarTransition
UIElement.Rotation	RotationTransition	ScalarTransition
UIElement.Scale	ScaleTransition	Vector3Transition
UIElement.Translation	TranslationTransition	Vector3Transition
Border.Background	BackgroundTransition	BrushTransition
ContentPresenter.Background	BackgroundTransition	BrushTransition
Panel.Background	BackgroundTransition	BrushTransition

This example shows how to use the Opacity property and transition to make a button fade in when the control is enabled and fade out when it's disabled.

XAML

```
<Button x:Name="SubmitButton"
        Content="Submit"
        Opacity="{x:Bind OpaqueIfEnabled(SubmitButton.IsEnabled),
Mode=OneWay}">
    <Button.OpacityTransition>
        <ScalarTransition />
    </Button.OpacityTransition>
</Button>
```

C#

```
public double OpaqueIfEnabled(bool IsEnabled)
{
    return IsEnabled ? 1.0 : 0.2;
}
```

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the **Windows App SDK** and **WinUI 3**, but are generally applicable to UWP apps that use **WinUI 2**. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

- ✓ **UWP APIs:** Windows.UI.Xaml.Media.Animation Namespace, Windows.UI.Xaml.Controls namespace
- ✓ **WinUI 2 Apis:** Microsoft.UI.Xaml.Controls namespace
- ✓ Open the WinUI 2 Gallery app and see Implicit Transitions in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

Implicit animations require Windows 10, version 1809 (SDK 17763) or later.

Related articles

- Motion overview
- Timing and easing
- Directionality and gravity

Page transitions

Article • 05/17/2022

Page transitions navigate users between pages in an app, providing feedback as the relationship between pages. Page transitions help users understand if they are at the top of a navigation hierarchy, moving between sibling pages, or navigating deeper into the page hierarchy.

Two different animations are provided for navigation between pages in an app, *Page refresh* and *Drill*, and are represented by subclasses of **NavigationTransitionInfo**.

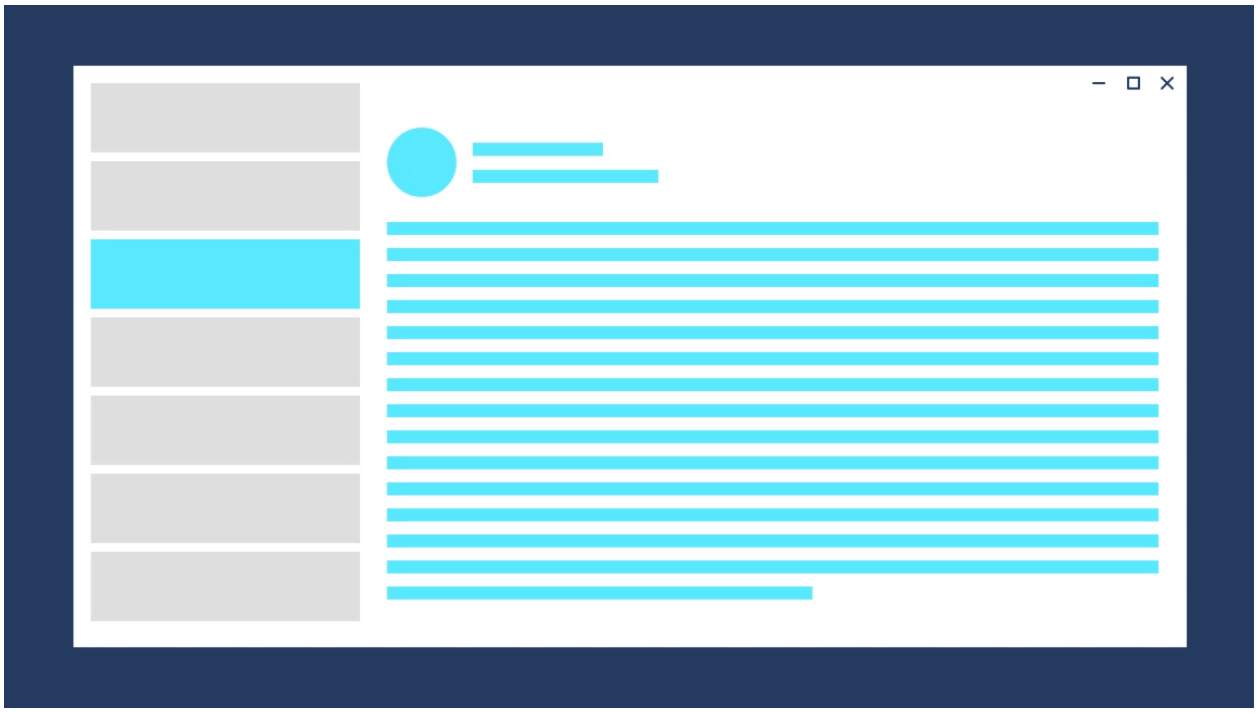
Examples

WinUI 2 Gallery	
	If you have the **WinUI 2 Gallery** app installed, click here to open the app and see Page Transitions in action. • Get the WinUI 2 Gallery app (Microsoft Store) • Get the source code (GitHub)

Page refresh

Page refresh is a combination of a slide up animation and a fade in animation for the incoming content. Use page refresh when the user is taken to the top of a navigational stack, such as navigating between tabs or left-nav items.

The desired feeling is that the user has started over.



The page refresh animation is represented by the EntranceNavigationTransitionInfoClass.

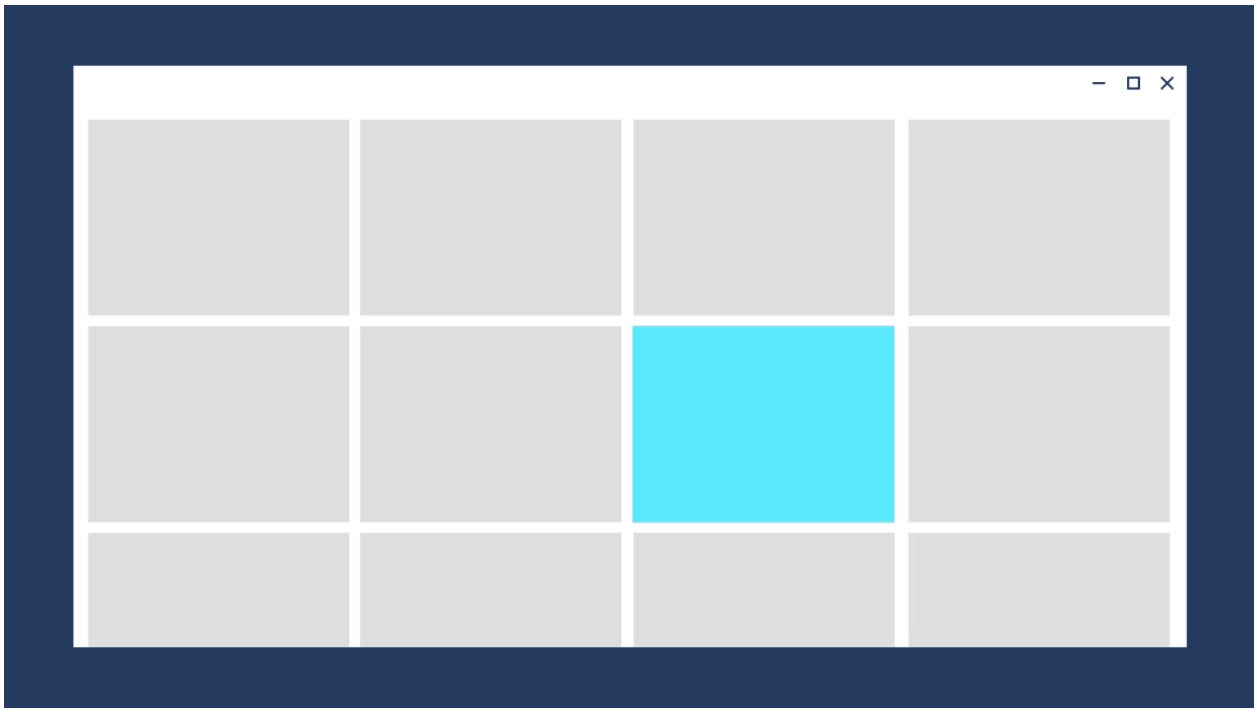
```C#
// Explicitly play the page refresh animation
myFrame.Navigate(typeof(Page2), null, new
EntranceNavigationTransitionInfo());
```

Note: A Frame automatically uses NavigationThemeTransition to animate navigation between two pages. By default, the animation is page refresh.

Drill

Use drill when users navigate deeper into an app, such as displaying more information after selecting an item.

The desired feeling is that the user has gone deeper into the app.



The drill animation is represented by the **DrillInNavigationTransitionInfo** class.

C#

```
// Play the drill in animation
myFrame.Navigate(typeof(Page2), null, new
DrillInNavigationTransitionInfo());
```

Horizontal slide

Use horizontal slide to show that sibling pages appear next to each other. The NavigationView control automatically uses this animation for top nav, but if you are building your own horizontal navigation experience, then you can implement horizontal slide with SlideNavigationTransitionInfo.

The desired feeling is that the user is navigating between pages that are next to each other.

C#

```
// Navigate to the right, ie. from LeftPage to RightPage
myFrame.Navigate(typeof(RightPage), null, new
SlideNavigationTransitionInfo() { Effect =
SlideNavigationTransitionEffect.FromRight } );

// Navigate to the left, ie. from RightPage to LeftPage
myFrame.Navigate(typeof(LeftPage), null, new SlideNavigationTransitionInfo()
{ Effect = SlideNavigationTransitionEffect.FromLeft } );
```

Suppress

To avoid playing any animation during navigation, use **SuppressNavigationTransitionInfo** in the place of other **NavigationTransitionInfo** subtypes.

```
C#
```

```
// Suppress the default animation
myFrame.Navigate(typeof(Page2), null, new
SuppressNavigationTransitionInfo());
```

Suppressing the animation is useful if you are building your own transition using Connected Animations or implicit show/hide animations.

Backwards navigation

You can use `Frame.GoBack(NavigationTransitionInfo)` to play a specific transition when navigating backwards.

This can be useful when you modify navigation behavior dynamically based on screen size; for example, in a responsive list/detail scenario.

Related topics

- Navigate between two pages
- Motion in UWP apps

Connected animation for Windows apps

Article • 05/17/2022

Connected animations let you create a dynamic and compelling navigation experience by animating the transition of an element between two different views. This helps the user maintain their context and provides continuity between the views.

In a connected animation, an element appears to "continue" between two views during a change in UI content, flying across the screen from its location in the source view to its destination in the new view. This emphasizes the common content between the views and creates a beautiful and dynamic effect as part of a transition.

> **Important APIs**: ConnectedAnimation class, ConnectedAnimationService class

Examples

WinUI 2 Gallery	
	If you have the **WinUI 2 Gallery** app installed, click here to open the app and see Connected Animation in action. • Get the WinUI 2 Gallery app (Microsoft Store) • Get the source code (GitHub)

In this short video, an app uses a connected animation to animate an item image as it "continues" to become part of the header of the next page. The effect helps maintain user context across the transition.

Connected animation and the Fluent Design System

The Fluent Design System helps you create modern, bold UI that incorporates light, depth, motion, material, and scale. Connected animation is a Fluent Design System component that adds motion to your app. To learn more, see the Fluent Design overview.

Why connected animation?

When navigating between pages, it's important for the user to understand what new content is being presented after the navigation and how it relates to their intent when navigating. Connected animations provide a powerful visual metaphor that emphasizes the relationship between two views by drawing the user's focus to the content shared between them. Additionally, connected animations add visual interest and polish to page navigation that can help differentiate the motion design of your app.

When to use connected animation

Connected animations are generally used when changing pages, though they can be applied to any experience where you are changing content in a UI and want the user to maintain context. You should consider using a connected animation instead of a drill in navigation transition whenever there is an image or other piece of UI shared between the source and destination views.

Configure connected animation

> **ⓘ Important**
>
> This feature requires that your app's Target version be Windows 10, version 1809 (**SDK 17763**) or later. The Configuration property is not available in earlier SDKs. You can target a Minimum version lower than SDK 17763 using adaptive code or conditional XAML. For more info, see **Version adaptive apps**.

Starting in Windows 10, version 1809, connected animations further embody Fluent design by providing animation configurations tailored specifically for forward and backwards page navigation.

You specify an animation configuration by setting the Configuration property on the ConnectedAnimation. (We'll show examples of this in the next section.)

This table describes the available configurations. For more information about the motion principles applied in these animations, see Directionality and gravity.

GravityConnectedAnimationConfiguration
This is the default configuration, and is recommended for forward navigation.
As the user navigates forward in the app (A to B), the connected element appears to physically "pull off the page". In doing so, the element appears to move forward in z-space and drops a bit as an effect of gravity taking hold. To overcome the effects of gravity, the element gains velocity and accelerates into its final position. The result is a "scale and dip" animation.

DirectConnectedAnimationConfiguration
As the user navigates backwards in the app (B to A), the animation is more direct. The connected element linearly translates from B to A using a decelerate cubic Bezier easing function. The backwards visual affordance returns the user to their previous state as fast as possible while still maintaining the context of the navigation flow.

BasicConnectedAnimationConfiguration
This is the default (and only) animation used in versions prior to Windows 10, version 1809 (SDK 17763).

ConnectedAnimationService configuration

The ConnectedAnimationService class has two properties that apply to the individual animations rather than the overall service.

- DefaultDuration
- DefaultEasingFunction

To achieve the various effects, some configurations ignore these properties on ConnectedAnimationService and use their own values instead, as described in this table.

Configuration	**Respects DefaultDuration?**	**Respects DefaultEasingFunction?**
Gravity	Yes	Yes* **The basic translation from A to B uses this easing function, but the "gravity dip" has its own easing function.*

Configuration	Respects DefaultDuration?	Respects DefaultEasingFunction?
Direct	No *Animates over 150ms.*	No *Uses the Decelerate easing function.*
Basic	Yes	Yes

How to implement connected animation

Setting up a connected animation involves two steps:

1. *Prepare* an animation object on the source page, which indicates to the system that the source element will participate in the connected animation.
2. *Start* the animation on the destination page, passing a reference to the destination element.

When navigating from the source page, call ConnectedAnimationService.GetForCurrentView to get an instance of ConnectedAnimationService. To prepare an animation, call PrepareToAnimate on this instance, and pass in a unique key and the UI element you want to use in the transition. The unique key lets you retrieve the animation later on the destination page.

C#

```
ConnectedAnimationService.GetForCurrentView()
    .PrepareToAnimate("forwardAnimation", SourceImage);
```

When the navigation occurs, start the animation in the destination page. To start the animation, call ConnectedAnimation.TryStart. You can retrieve the right animation instance by calling ConnectedAnimationService.GetAnimation with the unique key you provided when creating the animation.

C#

```
ConnectedAnimation animation =

ConnectedAnimationService.GetForCurrentView().GetAnimation("forwardAnimation
");
if (animation != null)
{
    animation.TryStart(DestinationImage);
}
```

Forward navigation

This example shows how to use ConnectedAnimationService to create a transition for forward navigation between two pages (Page_A to Page_B).

The recommended animation configuration for forward navigation is GravityConnectedAnimationConfiguration. This is the default, so you don't need to set the Configuration property unless you want to specify a different configuration.

Set up the animation in the source page.

XAML

```
<!-- Page_A.xaml -->

<Image x:Name="SourceImage"
       HorizontalAlignment="Left" VerticalAlignment="Top"
       Width="200" Height="200"
       Stretch="Fill"
       Source="Assets/StoreLogo.png"
       PointerPressed="SourceImage_PointerPressed"/>
```

C#

```
// Page_A.xaml.cs

private void SourceImage_PointerPressed(object sender,
PointerRoutedEventArgs e)
{
    // Navigate to detail page.
    // Suppress the default animation to avoid conflict with the connected
animation.
    Frame.Navigate(typeof(Page_B), null, new
SuppressNavigationTransitionInfo());
}

protected override void OnNavigatingFrom(NavigatingCancelEventArgs e)
{
    ConnectedAnimationService.GetForCurrentView()
        .PrepareToAnimate("forwardAnimation", SourceImage);
    // You don't need to explicitly set the Configuration property because
    // the recommended Gravity configuration is default.
    // For custom animation, use:
    // animation.Configuration = new BasicConnectedAnimationConfiguration();
}
```

Start the animation in the destination page.

XAML

```
<!-- Page_B.xaml -->

<Image x:Name="DestinationImage"
       Width="400" Height="400"
       Stretch="Fill"
       Source="Assets/StoreLogo.png" />
```

C#

```
// Page_B.xaml.cs

protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    ConnectedAnimation animation =

ConnectedAnimationService.GetForCurrentView().GetAnimation("forwardAnimation
");
    if (animation != null)
    {
        animation.TryStart(DestinationImage);
    }
}
```

Back navigation

For back navigation (Page_B to Page_A), you follow the same steps, but the source and destination pages are reversed.

When the user navigates back, they expect the app to be returned to the previous state as soon as possible. Therefore, the recommended configuration is DirectConnectedAnimationConfiguration. This animation is quicker, more direct, and uses the decelerate easing.

Set up the animation in the source page.

C#

```
// Page_B.xaml.cs

protected override void OnNavigatingFrom(NavigatingCancelEventArgs e)
{
    if (e.NavigationMode == NavigationMode.Back)
    {
        ConnectedAnimation animation =

ConnectedAnimationService.GetForCurrentView().PrepareToAnimate("backAnimatio
```

```
n", DestinationImage);

        // Use the recommended configuration for back animation.
        animation.Configuration = new
DirectConnectedAnimationConfiguration();
    }
}
```

Start the animation in the destination page.

C#

```
// Page_A.xaml.cs

protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    ConnectedAnimation animation =

ConnectedAnimationService.GetForCurrentView().GetAnimation("backAnimation");
    if (animation != null)
    {
        animation.TryStart(SourceImage);
    }
}
```

Between the time that the animation is set up and when it's started, the source element appears frozen above other UI in the app. This lets you perform any other transition animations simultaneously. For this reason, you shouldn't wait more than ~250 milliseconds in between the two steps because the presence of the source element may become distracting. If you prepare an animation and do not start it within three seconds, the system will dispose of the animation and any subsequent calls to TryStart will fail.

Connected animation in list and grid experiences

Often, you will want to create a connected animation from or to a list or grid control. You can use the two methods on ListView and GridView, PrepareConnectedAnimation and TryStartConnectedAnimationAsync, to simplify this process.

For example, say you have a **ListView** that contains an element with the name "PortraitEllipse" in its data template.

XAML

```
<ListView x:Name="ContactsListView" Loaded="ContactsListView_Loaded">
    <ListView.ItemTemplate>
        <DataTemplate x:DataType="vm:ContactsItem">
            <Grid>
                …
                <Ellipse x:Name="PortraitEllipse" … />
            </Grid>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

To prepare a connected animation with the ellipse corresponding to a given list item, call the PrepareConnectedAnimation method with a unique key, the item, and the name "PortraitEllipse".

C#

```
void PrepareAnimationWithItem(ContactsItem item)
{
     ContactsListView.PrepareConnectedAnimation("portrait", item,
"PortraitEllipse");
}
```

To start an animation with this element as the destination, such as when navigating back from a detail view, use TryStartConnectedAnimationAsync. If you have just loaded the data source for the ListView, TryStartConnectedAnimationAsync will wait to start the animation until the corresponding item container has been created.

C#

```
private async void ContactsListView_Loaded(object sender, RoutedEventArgs e)
{
    ContactsItem item = GetPersistedItem(); // Get persisted item
    if (item != null)
    {
        ContactsListView.ScrollIntoView(item);
        ConnectedAnimation animation =

ConnectedAnimationService.GetForCurrentView().GetAnimation("portrait");
        if (animation != null)
        {
            await ContactsListView.TryStartConnectedAnimationAsync(
                animation, item, "PortraitEllipse");
        }
    }
}
```

Coordinated animation

A *coordinated animation* is a special type of entrance animation where an element appears along with the connected animation target, animating in tandem with the connected animation element as it moves across the screen. Coordinated animations can add more visual interest to a transition and further draw the user's attention to the context that is shared between the source and destination views. In these images, the caption UI for the item is animating using a coordinated animation.

When a coordinated animation uses the gravity configuration, gravity is applied to both the connected animation element and the coordinated elements. The coordinated elements will "swoop" alongside the connected element so the elements stay truly coordinated.

Use the two-parameter overload of **TryStart** to add coordinated elements to a connected animation. This example demonstrates a coordinated animation of a Grid layout named "DescriptionRoot" that enters in tandem with a connected animation element named "CoverImage".

XAML

```
<!-- DestinationPage.xaml -->
<Grid>
    <Image x:Name="CoverImage" />
    <Grid x:Name="DescriptionRoot" />
</Grid>
```

C#

```
// DestinationPage.xaml.cs
void OnNavigatedTo(NavigationEventArgs e)
{
    var animationService = ConnectedAnimationService.GetForCurrentView();
    var animation = animationService.GetAnimation("coverImage");

    if (animation != null)
    {
        // Don’t need to capture the return value as we are not scheduling
any subsequent
        // animations
        animation.TryStart(CoverImage, new UIElement[] { DescriptionRoot });
     }
}
```

Do’s and don’ts

- Use a connected animation in page transitions where an element is shared between the source and destination pages.
- Use GravityConnectedAnimationConfiguration for forward navigation.
- Use DirectConnectedAnimationConfiguration for back navigation.
- Don't wait on network requests or other long-running asynchronous operations in between preparing and starting a connected animation. You may need to pre-load the necessary information to run the transition ahead of time, or use a low-resolution placeholder image while a high-resolution image loads in the destination view.
- Use SuppressNavigationTransitionInfo to prevent a transition animation in a **Frame** if you are using **ConnectedAnimationService**, since connected animations aren't meant to be used simultaneously with the default navigation transitions. See NavigationThemeTransition for more info on how to use navigation transitions.

Related articles

ConnectedAnimation

ConnectedAnimationService

NavigationThemeTransition

Parallax

Article • 05/17/2022

Parallax is a visual effect where items closer to the viewer move faster than items in the background. Parallax creates a feeling of depth, perspective, and movement. In a UWP app, you can use the ParallaxView control to create a parallax effect.

> **WinUI APIs:** ParallaxView class, VerticalShift property, HorizontalShift property
>
> **Platform APIs**: ParallaxView class, VerticalShift property, HorizontalShift property

Examples

Expand table

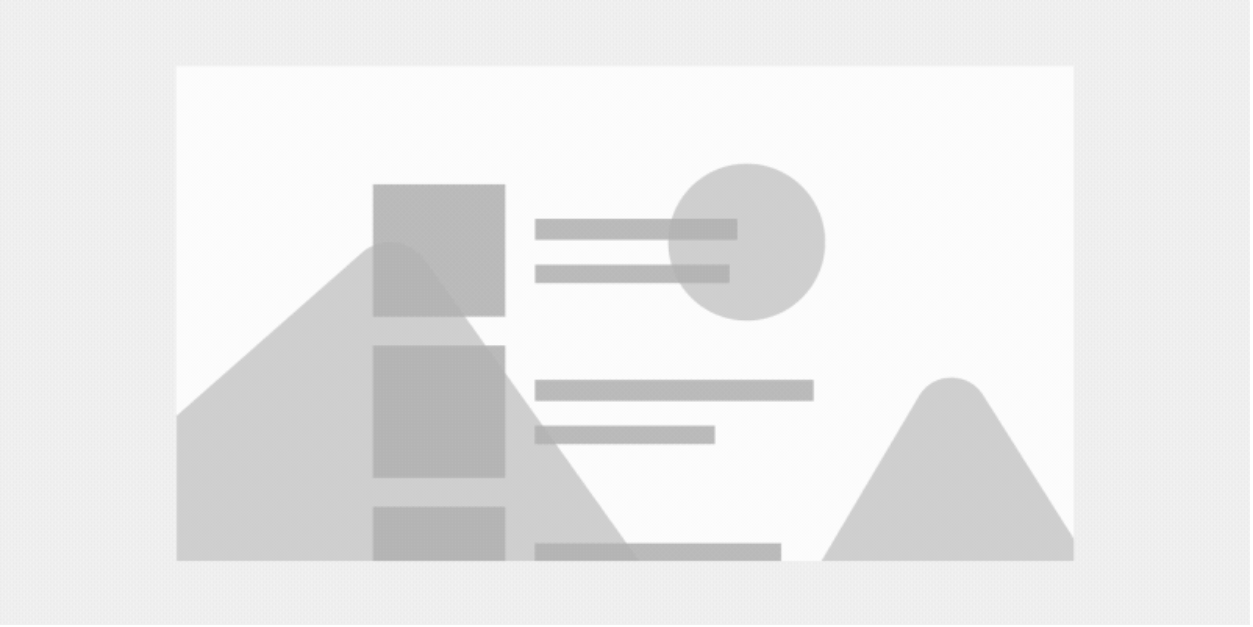
WinUI 2 Gallery	
	If you have the **WinUI 2 Gallery** app installed, click here to open the app and see the ParallaxView in action. • Get the WinUI 2 Gallery app (Microsoft Store) • Get the source code (GitHub)

Parallax and the Fluent Design System

The Fluent Design System helps you create modern, bold UI that incorporates light, depth, motion, material, and scale. Parallax is a Fluent Design System component that adds motion, depth, and scale to your app. To learn more, see the Fluent Design overview.

How it works in a user interface

In a UI, you can create a parallax effect by moving different objects at different rates when the UI scrolls or pans. To demonstrate, let's look at two layers of content, a list and a background image. The list is placed on top of the background image which already gives the illusion that the list might be closer to the viewer. Now, to achieve the parallax effect, we want the object closest to us to travel "faster" than the object that is farther away. As the user scrolls the interface, the list moves at a faster rate than the background image, which creates the illusion of depth.



Using the ParallaxView control to create a parallax effect

To create a parallax effect, you use the ParallaxView control. This control ties the scroll position of a foreground element, such as a list, to a background element, such as an image. As you scroll through the foreground element, it animates the background element to create a parallax effect.

To use the ParallaxView control, you provide a Source element, a background element, and set the VerticalShift (for vertical scrolling) and/or HorizontalShift (for horizontal scrolling) properties to a value greater than zero.

- The Source property takes a reference to the foreground element. For the parallax effect to occur, the foreground should be a ScrollViewer or an element that contains a ScrollViewer, such as a ListView or a RichTextBox.
- To set the background element, you add that element as a child of the ParallaxView control. The background element can be any UIElement, such as an Image or a panel that contains additional UI elements.

To create a parallax effect, the ParallaxView must be behind the foreground element. The Grid and Canvas panels let you layer items on top of each other, so they work well with the ParallaxView control.

This example creates a parallax effect for a list:

XAML

```
<Grid>
    <ParallaxView Source="{x:Bind ForegroundElement}" VerticalShift="50">

        <!-- Background element -->
        <Image x:Name="BackgroundImage" Source="Assets/turntable.png"
               Stretch="UniformToFill"/>
    </ParallaxView>

    <!-- Foreground element -->
    <ListView x:Name="ForegroundElement">
       <x:String>Item 1</x:String>
       <x:String>Item 2</x:String>
       <x:String>Item 3</x:String>
       <x:String>Item 4</x:String>
       <x:String>Item 5</x:String>
       <x:String>Item 6</x:String>
       <x:String>Item 7</x:String>
       <x:String>Item 8</x:String>
       <x:String>Item 9</x:String>
       <x:String>Item 10</x:String>
       <x:String>Item 11</x:String>
       <x:String>Item 13</x:String>
       <x:String>Item 14</x:String>
       <x:String>Item 15</x:String>
       <x:String>Item 16</x:String>
       <x:String>Item 17</x:String>
       <x:String>Item 18</x:String>
       <x:String>Item 19</x:String>
       <x:String>Item 20</x:String>
       <x:String>Item 21</x:String>
    </ListView>
</Grid>
```

The ParallaxView automatically adjusts the size of the image so it works for the parallax operation so you don't have to worry about the image scrolling out of view.

Customizing the parallax effect

The VerticalShift and HorizontalShift properties let you control degree of the parallax effect.

- The VerticalShift property specifies how far we want the background to vertically shift during the entire parallax operation. A value of 0 means the background doesn't move at all.
- The HorizontalShift property specifies how far we want the background to horizontally shift during the entire parallax operation. A value of 0 means the background doesn't move at all.

Larger values create a more dramatic effect.

For the complete list of ways to customize parallax, see the ParallaxView class.

Do's and don'ts

- Use parallax in lists with a background image
- Consider using parallax in ListViewItems when ListViewItems contain an image
- Don't use it everywhere, overuse can diminish its impact

Related articles

- ParallaxView class
- Fluent Design for UWP
- Science in the System: Fluent Design and Depth

Animations in XAML

Article • 01/04/2023

Animations can enhance your app by adding movement and interactivity. By using the animations from the Windows Runtime animation library, you can integrate the Windows look and feel into your app. This topic provides a summary of the animations and examples of typical scenarios where each is used.

> **Tip**
>
> The Windows Runtime controls for XAML include certain types of animations as built-in behaviors that come from an animation library. By using these controls in your app, you can get the animated look and feel without having to program it yourself.

Animations from the Windows Runtime animation library provide these benefits:

- Motions that align to the Guidelines for animations
- Fast, fluid transitions between UI states that inform but do not distract the user
- Visual behavior that indicates transitions within an app to the user

For example, when the user adds an item to a list, instead of the new item instantly appearing in the list, the new item animates into place. The other items in the list animate to their new positions over a short period of time, making room for the added item. The transition behavior here makes the control interaction more apparent to the user.

Windows 10, version 1607 introduces a new **ConnectedAnimationService** API for implementing animations where an element appears to animate between views during a navigation. This API has a different usage pattern from the other animation library API's. Usage of **ConnectedAnimationService** is covered in the reference page.

The animation library does not provide animations for every possible scenario. There are cases where you might wish to create a custom animation in XAML. For more info, see Storyboarded animations.

Additionally, for certain advanced scenarios like animating an item based on scroll position of a ScrollViewer, developers may wish to use Visual Layer interoperation to implement custom animations. See Visual Layer for more information.

Types of animations

The Windows Runtime animation system and the animation library serve the larger goal of enabling controls and other parts of UI to have an animated behavior. There are several distinct types of animations.

- *Theme transitions* are applied automatically when certain conditions change in the UI, involving controls or elements from the predefined Windows Runtime XAML UI types. These are termed *theme transitions* because the animations support the Windows look and feel, and define what all apps do for particular UI scenarios when they change from one interaction mode to another. The theme transitions are part of the animation library.
- *Theme animations* are animations to one or more properties of predefined Windows Runtime XAML UI types. Theme animations differ from theme transitions because theme animations target one specific element and exist in specific visual states within a control, whereas the theme transitions are assigned to properties of the control that exist outside of the visual states and influence the transitions between those states. Many of the Windows Runtime XAML controls include theme animations within storyboards that are part of their control template, with the animations triggered by visual states. So long as you're not modifying the templates, you'll have those built-in theme animations available for the controls in your UI. However, if you do replace templates, then you'll be removing the built-in control theme animations too. To get them back, you must define a storyboard that includes theme animations within the control's set of visual states. You can also run theme animations from storyboards that aren't within visual states and start them with the **Begin** method, but that's less common. Theme animations are part of the animation library.
- *Visual transitions* are applied when a control transitions from one of its defined visual states to another state. These are custom animations that you write, and are typically related to the custom template you write for a control and the visual state definitions within that template. The animation only runs during the time between states, and that's typically a short amount of time, a few seconds at most. For more info, see "VisualTransition" section of Storyboarded animations for visual states.
- *Storyboarded animations* animate the value of a Windows Runtime dependency property over time. Storyboards can be defined as part of a visual transition, or triggered at runtime by the application. For more info, see Storyboarded animations. For more info about dependency properties and where they exist, see Dependency properties overview.
- *Connected animations* provided by the new **ConnectedAnimationService** API allow developers to easily create an effect where an element appears to animate

between views during a navigation. This API is available starting in Windows 10, version 1607. See **ConnectedAnimationService** for more information.

Animations available in the library

The following animations are supplied in the animation library. Click on the name of an animation to learn more about their main usage scenarios, how to define them, and to see an example of the animation.

- Page transition: Animates page transitions in a **Frame**.
- Content and entrance transition: Animates one piece or set of content into or out of view.
- Fade in/out, and crossfade: Shows transient elements or controls, or refreshes a content area.
- Pointer up/down: Gives visual feedback of a tap or click on a tile.
- Reposition: Moves an element into a new position.
- Show/hide popup: Displays contextual UI on top of the view.
- Show/hide edge UI: Slides edge-based UI, including large UI such as a panel, into or out of view.
- List item changes: Adds or deletes an item from a list, or reordering of the items.
- Drag/drop: Gives visual feedback during a drag-and-drop operation.

Page transition

Use page transitions to animate navigation within an app. Since almost all apps use some kind of navigation, page transition animations are the most common type of theme animation used by apps. See **NavigationThemeTransition** for more information about the page transition APIs.

Content transition and entrance transition

Use content transition animations (**ContentThemeTransition**) to move a piece or a set of content into or out of the current view. For example, the content transition animations show content that was not ready to display when the page was first loaded, or when the content changes on a section of a page.

EntranceThemeTransition represents a motion that can apply to content when a page or large section of UI is first loaded. Thus the first appearance of content can offer different feedback than a change to content does. **EntranceThemeTransition** is equivalent to a **NavigationThemeTransition** with the default parameters, but may be used outside of a **Frame**.

Fade in/out, and crossfade

Use fade in and fade out animations to show or hide transient UI or controls. In XAML these are represented as **FadeInThemeAnimation** and **FadeOutThemeAnimation**. One example is in an app bar in which new controls can appear due to user interaction. Another example is a transient scroll bar or panning indicator that is faded out after no user input has been detected for some amount of time. Apps should also use the fade in animation when they transition from a placeholder item to the final item as content loads dynamically.

Use a crossfade animation to smooth the transition when an item's state is changing; for example, when the app refreshes the current contents of a view. The XAML animation library does not supply a dedicated crossfade animation (no equivalent for **crossFade**), but you can achieve the same result using **FadeInThemeAnimation** and **FadeOutThemeAnimation** with overlapped timing.

Pointer up/down

Use the **PointerUpThemeAnimation** and **PointerDownThemeAnimation** animations to give the user feedback for a successful tap or click on a tile. For example, when a user clicks or taps down on a tile, the pointer down animation is played. Once the click or tap has been released, the pointer up animation is played.

Reposition

Use the reposition animations (**RepositionThemeAnimation** or **RepositionThemeTransition**) to move an element into a new position. For example, moving the headers in an items control uses the reposition animation.

Show/hide popup

Use the **PopInThemeAnimation** and **PopOutThemeAnimation** when you show and hide a **Popup** or similar contextual UI on top of the current view. **PopupThemeTransition** is a theme transition that's useful feedback if you want to light dismiss a popup.

Show/hide edge UI

Use the **EdgeUIThemeTransition** animation to slide small, edge-based UI into and out of view. For example, use these animations when you show a custom app bar at the top or bottom of the screen or a UI surface for errors and warnings at the top of the screen.

Use the **PaneThemeTransition** animation to show and hide a pane or panel. This is for large edge-based UI such as a custom keyboard or a task pane.

List item changes

Use the **AddDeleteThemeTransition** animation to add animated behavior when you add or delete an item in an existing list. For add, the transition will first reposition existing items in the list to make space for the new items, and then add the new items. For delete, the transition removes items from a list and, if necessary, repositions the remaining list items once the deleted items have been removed.

There's also a separate **ReorderThemeTransition** that you apply if an item changes position in a list. This is animated differently than deleting an item and adding it in a new place with the associated delete/add animations.

Note that these animations are included in the default **ListView** and **GridView** templates so you do not need to manually add these animations if you are already using these controls.

Drag/drop

Use the drag animations (**DragItemThemeAnimation**, **DragOverThemeAnimation**) and drop animation (**DropTargetItemThemeAnimation**) to give visual feedback when the user drags or drops an item.

When active, the animations show the user that the list can be rearranged around a dropped item. It is helpful for users to know where the item will be placed in a list if it is dropped at the current location. The animations give visual feedback that an item being dragged can be dropped between two other items in the list and that those items will move out of the way.

Using animations with custom controls

The following table summarizes our recommendations for which animation you should use when you create a custom version of these Windows Runtime controls:

UI type	Recommended animation
Dialog box	FadeInThemeAnimation and FadeOutThemeAnimation

UI type	Recommended animation
Flyout	PopInThemeAnimation and PopOutThemeAnimation
Tooltip	FadeInThemeAnimation and FadeOutThemeAnimation
Context menu	PopInThemeAnimation and PopOutThemeAnimation
Command bar	EdgeUIThemeTransition
Task pane or edge-based panel	PaneThemeTransition
Contents of any UI container	ContentThemeTransition
For controls or if no other animation applies	FadeInThemeAnimation and FadeOutThemeAnimation

Transition animation examples

Ideally, your app uses animations to enhance the user interface or to make it more attractive without annoying your users. One way you can do this is to apply animated transitions to UI so that when something enters or leaves the screen or otherwise changes, the animation draws the attention of the user to the change. For example, your buttons may rapidly fade in and out of view rather than just appear and disappear. We created a number of APIs that can be used to create recommended or typical animation transitions that are consistent. The example here shows how to apply an animation to a button so that it swiftly slides into view.

XML

```
<Button Content="Transitioning Button">
     <Button.Transitions>
         <TransitionCollection>
             <EntranceThemeTransition/>
         </TransitionCollection>
     </Button.Transitions>
 </Button>
```

In this code, we add the EntranceThemeTransition object to the transition collection of the button. Now, when the button is first rendered, it swiftly slides into view rather than just appear. You can set a few properties on the animation object in order to adjust how

far it slides and from what direction, but it's really meant to be a simple API for a specific scenario, that is, to make an eye-catching entrance.

You can also define transition animation themes in the style resources of your app, allowing you to apply the effect uniformly. This example is equivalent to the previous one, only it is applied using a **Style**:

XML

```
<UserControl.Resources>
     <Style x:Key="DefaultButtonStyle" TargetType="Button">
         <Setter Property="Transitions">
             <Setter.Value>
                 <TransitionCollection>
                     <EntranceThemeTransition/>
                 </TransitionCollection>
             </Setter.Value>
         </Setter>
    </Style>
</UserControl.Resources>

<StackPanel x:Name="LayoutRoot">
    <Button Style="{StaticResource DefaultButtonStyle}"
Content="Transitioning Button"/>
</StackPanel>
```

The previous examples apply a theme transition to an individual control, however, theme transitions are even more interesting when you apply them to a container of objects. When you do this, all the child objects of the container take part in the transition. In the following example, an **EntranceThemeTransition** is applied to a **Grid** of rectangles.

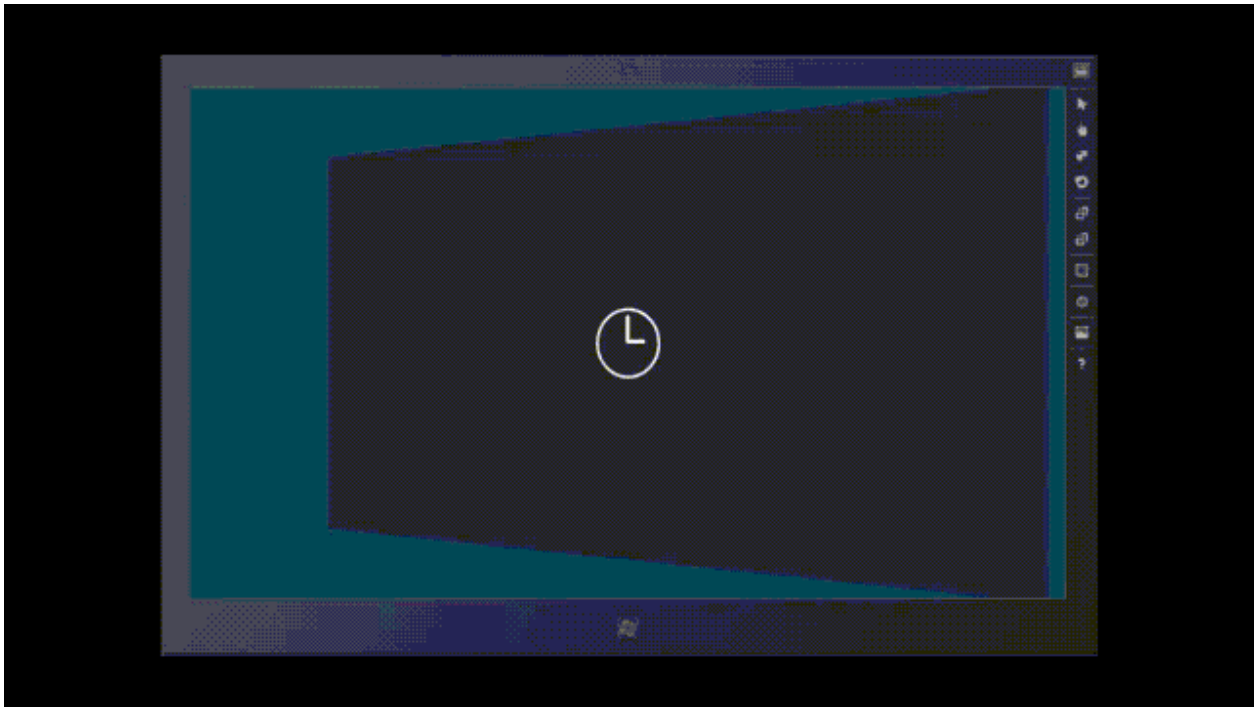
XML

```
<!-- If you set an EntranceThemeTransition animation on a panel, the
     children of the panel will automatically offset when they animate
     into view to create a visually appealing entrance. -->
<ItemsControl Grid.Row="1" x:Name="rectangleItems">
    <ItemsControl.ItemContainerTransitions>
        <TransitionCollection>
            <EntranceThemeTransition/>
        </TransitionCollection>
    </ItemsControl.ItemContainerTransitions>
    <ItemsControl.ItemsPanel>
        <ItemsPanelTemplate>
            <WrapGrid Height="400"/>
        </ItemsPanelTemplate>
    </ItemsControl.ItemsPanel>

    <!-- The sequence children appear depends on their order in
```

```
            the panel's children, not necessarily on where they render
            on the screen. Be sure to arrange your child elements in
            the order you want them to transition into view. -->
        <ItemsControl.Items>
            <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
            <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
            <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
            <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
            <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
            <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
            <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
            <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
            <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
        </ItemsControl.Items>
    </ItemsControl>
```

The child rectangles of the **Grid** transition into view one after the other in a visually pleasing way rather than all at once as would be the case if you applied this animation to the rectangles individually.

Here's a demonstration of this animation:



Child objects of a container can also re-flow when one or more of those children change position. In the following example, we apply a **RepositionThemeTransition** to a grid of rectangles. When you remove one of the rectangles, all the other rectangles re-flow into their new position.

XML

```
<Button Content="Remove Rectangle" Click="RemoveButton_Click"/>
```

```
<ItemsControl Grid.Row="1" x:Name="rectangleItems">
    <ItemsControl.ItemContainerTransitions>
        <TransitionCollection>

            <!-- Without this, there would be no animation when items
                 are removed. -->
            <RepositionThemeTransition/>
        </TransitionCollection>
    </ItemsControl.ItemContainerTransitions>
    <ItemsControl.ItemsPanel>
        <ItemsPanelTemplate>
            <WrapGrid Height="400"/>
        </ItemsPanelTemplate>
    </ItemsControl.ItemsPanel>

    <!-- All these rectangles are just to demonstrate how the items
         in the grid re-flow into position when one of the child items
         are removed. -->
    <ItemsControl.Items>
        <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
        <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
        <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
        <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
        <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
        <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
        <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
        <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
        <Rectangle Fill="Red" Width="100" Height="100" Margin="10"/>
    </ItemsControl.Items>
</ItemsControl>
```

C#

```
private void RemoveButton_Click(object sender, RoutedEventArgs e)
{
    if (rectangleItems.Items.Count > 0)
    {
        rectangleItems.Items.RemoveAt(0);
    }
}
```

C++

```
// .h
private:
void RemoveButton_Click(Platform::Object^ sender,
Windows::UI::Xaml::RoutedEventArgs^ e);

//.cpp
void BlankPage::RemoveButton_Click(Platform::Object^ sender,
Windows::UI::Xaml::RoutedEventArgs^ e)
{
```

```
    if (rectangleItems->Items->Size > 0)
    {
        rectangleItems->Items->RemoveAt(0);
    }
}
```

You can apply multiple transition animations to a single object or object container. For example, if you want the list of rectangles to animate into view and also animate when they change position, you can apply the RepositionThemeTransition and EntranceThemeTransition like this:

XML

```
...
<ItemsControl.ItemContainerTransitions>
    <TransitionCollection>
        <EntranceThemeTransition/>
        <RepositionThemeTransition/>
    </TransitionCollection>
</ItemsControl.ItemContainerTransitions>
...
```

There are several transition effects to create animations on your UI elements as they are added, removed, reordered, and so on. The names of these APIs all contain "ThemeTransition":

API	Description
NavigationThemeTransition	Provides a Windows personality animation for page navigation in a Frame.
AddDeleteThemeTransition	Provides the animated transition behavior for when controls add or delete children or content. Typically the control is an item container.
ContentThemeTransition	Provides the animated transition behavior for when the content of a control is changing. You can apply this in addition to AddDeleteThemeTransition.
EdgeUIThemeTransition	Provides the animated transition behavior for a (small) edge UI transition.
EntranceThemeTransition	Provides the animated transition behavior for when controls first appear.
PaneThemeTransition	Provides the animated transition behavior for a panel (large edge UI) UI transition.

API	Description
PopupThemeTransition	Provides the animated transition behavior that applies to pop-in components of controls (for example, tooltip-like UI on an object) as they appear.
ReorderThemeTransition	Provides the animated transition behavior for when list-view controls items change order. Typically this happens as a result of a drag-drop operation. Different controls and themes can have varying characteristics for the animations.
RepositionThemeTransition	Provides the animated transition behavior for when controls change position.

Theme animation examples

Transition animations are simple to apply. But you may want to have a bit more control over the timing and order of your animation effects. You can use theme animations to enable more control while still using a consistent theme for how your animation behaves. Theme animations also require less markup than custom animations. Here, we use the FadeOutThemeAnimation to make a rectangle fade out of view.

XML

```
<StackPanel>
    <StackPanel.Resources>
        <Storyboard x:Name="myStoryboard">
            <FadeOutThemeAnimation TargetName="myRectangle" />
        </Storyboard>
    </StackPanel.Resources>
    <Rectangle PointerPressed="Rectangle_Tapped" x:Name="myRectangle"
               Fill="Blue" Width="200" Height="300"/>
</StackPanel>
```

C#

```
// When the user taps the rectangle, the animation begins.
private void Rectangle_Tapped(object sender, PointerRoutedEventArgs e)
{
    myStoryboard.Begin();
}
```

VB

```
' When the user taps the rectangle, the animation begins.
Private Sub Rectangle_Tapped(sender As Object, e As PointerRoutedEventArgs)
    myStoryboard.Begin()
End Sub
```

C++

```
//.h
void Rectangle_Tapped(Platform::Object^ sender,
Windows::UI::Xaml::Input::PointerRoutedEventArgs^ e);

//.cpp
void BlankPage::Rectangle_Tapped(Object^ sender, PointerRoutedEventArgs^ e)
{
    myStoryboard->Begin();
}
```

Unlike transition animations, a theme animation doesn't have a built-in trigger (the transition) that runs it automatically. You must use a **Storyboard** to contain a theme animation when you define it in XAML. You can also change the default behavior of the animation. For example, you can slow down the fade-out by increasing the **Duration** time value on the **FadeOutThemeAnimation**.

Note For purposes of showing basic animation techniques, we're using app code to start the animation by calling methods of **Storyboard**. You can control how the **Storyboard** animations run using the **Begin**, **Stop**, **Pause**, and **Resume** **Storyboard** methods. However, that's not typically how you include library animations in apps. Rather, you usually integrate the library animations into the XAML styles and templates applied to controls or elements. Learning about templates and visual states is a little more involved. But we do cover how you'd use library animations in visual states as part of the Storyboarded animations for visual states topic.

You can apply several other theme animations to your UI elements to create animation effects. The names of these API all contain "ThemeAnimation":

API	Description
DragItemThemeAnimation	Represents the preconfigured animation that applies to item elements being dragged.
DragOverThemeAnimation	Represents the preconfigured animation that applies to the elements underneath an element being dragged.

API	Description
DropTargetItemThemeAnimation	The preconfigured animation that applies to potential drop target elements.
FadeInThemeAnimation	The preconfigured opacity animation that applies to controls when they first appear.
FadeOutThemeAnimation	The preconfigured opacity animation that applies to controls when they are removed from UI or hidden.
PointerDownThemeAnimation	The preconfigured animation for user action that taps or clicks an item or element.
PointerUpThemeAnimation	The preconfigured animation for user action that runs after a user taps down on an item or element and the action is released.
PopInThemeAnimation	The preconfigured animation that applies to pop-in components of controls as they appear. This animation combines opacity and translation.
PopOutThemeAnimation	The preconfigured animation that applies to pop-in components of controls as they are closed or removed. This animation combines opacity and translation.
RepositionThemeAnimation	The preconfigured animation for an object as it is repositioned.
SplitCloseThemeAnimation	The preconfigured animation that conceals a target UI using an animation in the style of a ComboBox opening and closing.
SplitOpenThemeAnimation	The preconfigured animation that reveals a target UI using an animation in the style of a ComboBox opening and closing.
DrillInThemeAnimation	Represents a preconfigured animation that runs when a user navigates forward in a logical hierarchy, like from a list page to a detail page.
DrillOutThemeAnimation	Represents a preconfigured animation that runs when a user navigates backward in a logical hierarchy, like from a detail page to a list page.

Create your own animations

When theme animations are not enough for your needs, you can create your own animations. You animate objects by animating one or more of their property values. For example, you can animate the width of a rectangle, the angle of a **RotateTransform**, or the color value of a button. We term this type of custom animation a storyboarded animation, to distinguish it from the library animations that the Windows Runtime already provides as a preconfigured animation type. For storyboarded animations, you use an animation that can change values of a particular type (for example **DoubleAnimation** to animate a **Double**) and put that animation within a **Storyboard** to control it.

In order to be animated, the property you are animating must be a *dependency property*. For more info about dependency properties, see Dependency properties overview. For more info on creating custom storyboarded animations, including how to target and control them, see Storyboarded animations.

The biggest area of app UI definition in XAML where you'll define custom storyboarded animations is if you are defining visual states for controls in XAML. You'll be doing this either because you are creating a new control class, or because you are re-templating an existing control that has visual states in its control template. For more info, see Storyboarded animations for visual states.

Animating XAML elements with composition animations

Article • 05/17/2022

This article introduces new properties that let you animate a XAML UIElement with the performance of composition animations and the ease of setting XAML properties.

Prior to Windows 10, version 1809, you had 2 choices to build animations in your UWP apps:

- use XAML constructs like Storyboarded animations, or the **ThemeTransition* and **ThemeAnimation* classes in the Windows.UI.Xaml.Media.Animation namespace.
- use composition animations as described in Using the Visual Layer with XAML.

Using the visual layer provides better performance than using the XAML constructs. But using ElementCompositionPreview to get the element's underlying composition Visual object, and then animating the Visual with composition animations, is more complex to use.

Starting in Windows 10, version 1809, you can animate properties on a UIElement directly using composition animations without the requirement to get the underlying composition Visual.

> **Note**
>
> To use these properties on UIElement, your UWP project target version must be 1809 or later. For more info about configuring your project version, see Version adaptive apps.

Examples

Expand table

WinUI 2 Gallery	
	If you have the **WinUI 2 Gallery** app installed, click here to open the app and see Animation interop in action. • Get the WinUI 2 Gallery app (Microsoft Store) • Get the source code (GitHub)

New rendering properties replace old rendering properties

This table shows the properties you can use to modify the rendering of a UIElement, that can also be animated with a CompositionAnimation.

Expand table

Property	Type	Description
Opacity	Double	The degree of the object's opacity
Translation	Vector3	Shift the X/Y/Z position of the element
TransformMatrix	Matrix4x4	The transform matrix to apply to the element
Scale	Vector3	Scale the element, centered on the CenterPoint
Rotation	Float	Rotate the element around the RotationAxis and CenterPoint
RotationAxis	Vector3	The axis of rotation
CenterPoint	Vector3	The center point of scale and rotation

The TransformMatrix property value is combined with the Scale, Rotation, and Translation properties in the following order: TransformMatrix, Scale, Rotation, Translation.

These properties don't affect the element's layout, so modifying these properties does not cause a new Measure/Arrange pass.

These properties have the same purpose and behavior as the like-named properties on the composition Visual class (except for Translation, which isn't on Visual).

Example: Setting the Scale property

This example shows how to set the Scale property on a Button.

XAML

```
<Button Scale="2,2,1" Content="I am a large button" />
```

C#

```
var button = new Button();
button.Content = "I am a large button";
```

```
button.Scale = new Vector3(2.0f,2.0f,1.0f);
```

Mutual exclusivity between new and old properties

> **Note**
>
> The **Opacity** property does not enforce the mutual exclusivity described in this section. You use the same Opacity property whether you use XAML or composition animations.

The properties that can be animated with a CompositionAnimation are replacements for several existing UIElement properties:

- RenderTransform
- RenderTransformOrigin
- Projection
- Transform3D

When you set (or animate) any of the new properties, you cannot use the old properties. Conversely, if you set (or animate) any of the old properties, you cannot use the new properties.

You also cannot use the new properties if you use ElementCompositionPreview to get and manage the Visual yourself using these methods:

- ElementCompositionPreview.GetElementVisual
- ElementCompositionPreview.SetIsTranslationEnabled

> **Important**
>
> Attempting to mix the use of the two sets of properties will cause the API call to fail and produce an error message.

It's possible to switch from one set of properties by clearing them, though for simplicity it's not recommended. If the property is backed by a DependencyProperty (for example, UIElement.Projection is backed by UIElement.ProjectionProperty), then call ClearValue to restore it to its "unused" state. Otherwise (for example, the Scale property), set the property to its default value.

Animating UIElement properties with CompositionAnimation

You can animate the rendering properties listed in the table with a CompositionAnimation. These properties can also be referenced by an ExpressionAnimation.

Use the StartAnimation and StopAnimation methods on UIElement to animate the UIElement properties.

Example: Animating the Scale property with a Vector3KeyFrameAnimation

This example shows how to animate the scale of a Button.

C#

```
var compositor = Window.Current.Compositor;
var animation = compositor.CreateVector3KeyFrameAnimation();

animation.InsertKeyFrame(1.0f, new Vector3(2.0f,2.0f,1.0f));
animation.Duration = TimeSpan.FromSeconds(1);
animation.Target = "Scale";

button.StartAnimation(animation);
```

Example: Animating the Scale property with an ExpressionAnimation

A page has two buttons. The second button animates to be twice as large (via scale) as the first button.

XAML

```
<Button x:Name="sourceButton" Content="Source"/>
<Button x:Name="destinationButton" Content="Destination"/>
```

C#

```
var compositor = Window.Current.Compositor;
var animation =
compositor.CreateExpressionAnimation("sourceButton.Scale*2");
animation.SetExpressionReferenceParameter("sourceButton", sourceButton);
```

```
animation.Target = "Scale";
destinationButton.StartAnimation(animation);
```

Related topics

- Storyboarded animations
- Using the Visual Layer with XAML
- Transforms overview

Storyboarded animations

Article • 05/24/2022

Storyboarded animations are not just animations in the visual sense. A storyboarded animation is a way to change the value of a dependency property as a function of time. One of the main reasons you might need a storyboarded animation that's not from the animation library is to define the visual state for a control, as part of a control template or page definition.

Differences with Silverlight and WPF

If you are familiar with Microsoft Silverlight or Windows Presentation Foundation (WPF), read this section; otherwise, you can skip it.

In general, creating storyboarded animations in a Windows Runtime app is like Silverlight or WPF. But there are a number of important differences:

- Storyboarded animations are not the only way to visually animate a UI, nor are they necessarily the easiest way for app developers to do so. Rather than using storyboarded animations it's often a better design practice to use theme animations and transition animations. These can quickly create recommended UI animations without getting into the intricacies of animation property targeting. For more info see Animations overview.
- In the Windows Runtime, many XAML controls include theme animations and transition animations as part of their built-in behavior. For the most part, WPF and Silverlight controls didn't have a default animation behavior.
- Not all custom animations you create can run by default in a Windows Runtime app, if the animation system determines that the animation might cause bad performance in your UI. Animations where the system determines there could be a performance impact are called *dependent animations*. It's dependent because the clocking of your animation is directly working against the UI thread, which is also where active user input and other updates are trying to apply the runtime changes to UI. A dependent animation that's consuming extensive system resources on the UI thread can make the app appear unresponsive in certain situations. If your animation causes a layout change or otherwise has the potential to impact performance on the UI thread, you often need to explicitly enable the animation to see it run. That's what the **EnableDependentAnimation** property on specific animation classes is for. See Dependent and independent animations for more info.
- Custom easing functions are not currently supported in the Windows Runtime.

Defining storyboarded animations

A storyboarded animation is a way to change the value of a dependency property as a function of time. The property you are animating is not always a property that directly affects the UI of your app. But since XAML is about defining UI for an app, usually it is a UI-related property you are animating. For example, you can animate the angle of a **RotateTransform**, or the color value of a button's background.

One of the main reasons you might be defining a storyboarded animation is if you are a control author or are re-templating a control, and you are defining visual states. For more info, see Storyboarded animations for visual states.

Whether you are defining visual states or a custom animation for an app, the concepts and APIs for storyboarded animations that are described in this topic mostly apply to either.

In order to be animated, the property you are targeting with a storyboarded animation must be a *dependency property*. A dependency property is a key feature of the Windows Runtime XAML implementation. The writeable properties of most common UI elements are typically implemented as dependency properties, so that you can animate them, apply data-bound values, or apply a **Style** and target the property with a **Setter**. For more info about how dependency properties work, see Dependency properties overview.

Most of the time, you define a storyboarded animation by writing XAML. If you use a tool such as Microsoft Visual Studio, it will produce the XAML for you. It's possible to define a storyboarded animation using code too, but that's less common.

Let's look at a simple example. In this XAML example, the **Opacity** property is animated on a particular **Rectangle** object.

XAML

```
<Page ...>
  <Page.Resources>
    <!-- Storyboard resource: Animates a rectangle's opacity. -->
    <Storyboard x:Name="myStoryboard">
      <DoubleAnimation
        Storyboard.TargetName="MyAnimatedRectangle"
        Storyboard.TargetProperty="Opacity"
        From="1.0" To="0.0" Duration="0:0:1"/>
    </Storyboard>
  </Page.Resources>

  <!--Page root element, UI definition-->
  <Grid>
```

```
    <Rectangle x:Name="MyAnimatedRectangle"
      Width="300" Height="200" Fill="Blue"/>
  </Grid>
</Page>
```

Identifying the object to animate

In the previous example, the storyboard was animating the **Opacity** property of a **Rectangle**. You don't declare the animations on the object itself. Instead, you do this within the animation definition of a storyboard. Storyboards are usually defined in XAML that's not in the immediate vicinity of the XAML UI definition of the object to animate. Instead, they're usually set up as a XAML resource.

To connect an animation to a target, you reference the target by its identifying programming name. You should always apply the x:Name attribute in the XAML UI definition to name the object that you want to animate. You then target the object to animate by setting **Storyboard.TargetName** within the animation definition. For the value of **Storyboard.TargetName**, you use the name string of the target object, which is what you set earlier and elsewhere with x:Name attribute.

Targeting the dependency property to animate

You set a value for **Storyboard.TargetProperty** in the animation. This determines which specific property of the targeted object is animated.

Sometimes you need to target a property that's not an immediate property of the target object, but that is nested more deeply in an object-property relationship. You often need to do this in order to drill down into a set of contributing object and property values until you can reference a property type that can be animated (**Double**, **Point**, **Color**). This concept is called *indirect targeting*, and the syntax for targeting a property in this way is known as a *property path*.

Here's an example. One common scenario for a storyboarded animation is to change the color of a part of an app UI or control in order to represent that the control is in a particular state. Say you want to animate the **Foreground** of a **TextBlock**, so that it turns from red to green. You'd expect that a **ColorAnimation** is involved, and that's correct. However, none of the properties on UI elements that affect the object's color are actually of type **Color**. Instead, they're of type **Brush**. So what you actually need to target for animation is the **Color** property of the **SolidColorBrush** class, which is a **Brush**-derived type that's typically used for these color-related UI properties. And here's what that looks like in terms of forming a property path for your animation's property targeting:

XAML

```
<Storyboard x:Name="myStoryboard">
  <ColorAnimation
    Storyboard.TargetName="tb1"
    Storyboard.TargetProperty="(TextBlock.Foreground).
(SolidColorBrush.Color)"
    From="Red" To="Green"/>
</Storyboard>
```

Here's how to think of this syntax in terms of its parts:

- Each set of () parentheses encloses a property name.
- Within the property name, there's a dot, and that dot separates a type name and a property name, so that the property you're identifying is unambiguous.
- The dot in the middle, the one that's not inside parentheses, is a step. This is interpreted by the syntax to mean, take the value of the first property (which is an object), step into its object model, and target a specific sub-property of the first property's value.

Here's a list of animation targeting scenarios where you'll probably be using indirect property targeting, and some property path strings that approximates the syntax you'll use:

- Animating the **X** value of a **TranslateTransform**, as applied to a **RenderTransform**: `(UIElement.RenderTransform).(TranslateTransform.X)`
- Animating a **Color** within a **GradientStop** of a **LinearGradientBrush**, as applied to a **Fill**: `(Shape.Fill).(GradientBrush.GradientStops)[0].(GradientStop.Color)`
- Animating the **X** value of a **TranslateTransform**, which is 1 of 4 transforms in a **TransformGroup**, as applied to a **RenderTransform**: `(UIElement.RenderTransform).(TransformGroup.Children)[3].(TranslateTransform.X)`

You'll notice some of these examples use square brackets around numbers. This is an indexer. It indicates that the property name preceding it has a collection as value, and that you want an item (as identified by a zero-based index) from within that collection.

You can also animate XAML attached properties. Always enclose the full attached property name in parentheses, for example `(Canvas.Left)`. For more info, see Animating XAML attached properties.

For more info on how to use a property path for indirect targeting of the property to animate, see Property-path syntax or **Storyboard.TargetProperty attached property**.

Animation types

The Windows Runtime animation system has three specific types that storyboarded animations can apply to:

- **Double**, can be animated with any **DoubleAnimation**
- **Point**, can be animated with any **PointAnimation**
- **Color**, can be animated with any **ColorAnimation**

There's also a generalized **Object** animation type for object reference values, which we'll discuss later.

Specifying the animated values

So far we've shown you how to target the object and the property to animate, but haven't yet described what the animation does to the property value when it runs.

The animation types we've described are sometimes referred to as **From/To/By** animations. This means that the animation is changing the value of a property, over time, using one or more of these inputs that come from the animation definition:

- The value starts at the **From** value. If you don't specify a **From** value, the starting value is whatever value the animated property has at the time before the animation runs. This might be a default value, a value from a style or template, or a value specifically applied by a XAML UI definition or app code.
- At the end of the animation, the value is the **To** value.
- Or, to specify an ending value relative to the starting value, set the **By** property. You'd set this instead of the **To** property.
- If you don't specify a **To** value or a **By** value, the ending value is whatever value the animated property has at the time before the animation runs. In this case you'd better have a **From** value because otherwise the animation won't change the value at all; its starting and ending values are both the same.
- An animation typically has at least one of **From**, **By** or **To** but never all three.

Let's revisit the earlier XAML example and look again at the **From** and **To** values, and the **Duration**. The example is animating the **Opacity** property, and the property type of **Opacity** is **Double**. So the animation to use here is **DoubleAnimation**.

`From="1.0" To="0.0"` specifies that when the animation runs, the **Opacity** property starts at a value of 1 and animates to 0. In other words, in terms of what these **Double** values mean to the **Opacity** property, this animation will cause the object to start opaque and then fade to transparent.

```
XAML
```

```
...
<Storyboard x:Name="myStoryboard">
  <DoubleAnimation
    Storyboard.TargetName="MyAnimatedRectangle"
    Storyboard.TargetProperty="Opacity"
    From="1.0" To="0.0" Duration="0:0:1"/>
</Storyboard>
...
```

`Duration="0:0:1"` specifies how long the animation lasts, that is, how fast the rectangle fades. A **Duration** property is specified in the form of *hours*:*minutes*:*seconds*. The time duration in this example is one second.

For more info about **Duration** values and the XAML syntax, see **Duration**.

> **Note**
>
> For the example we showed, if you were sure that the starting state of the object being animated has **Opacity** always equal to 1, either through the default or an explicit set, you could omit the **From** value, the animation would use the implicit starting value, and the result would be the same.

From/To/By are nullable

We mentioned previously that you can omit **From**, **To** or **By** and thus use current non-animated values as substitutes for a missing value. **From**, **To**, or **By** properties of an animation aren't of the type you might guess. For example the type of the **DoubleAnimation.To** property isn't **Double**. Instead, it's a **Nullable** for **Double**. And its default value is **null**, not 0. That **null** value is how the animation system distinguishes that you haven't specifically set a value for a **From**, **To**, or **By** property. Visual C++ component extensions (C++/CX) doesn't have a **Nullable** type, so it uses **IReference** instead.

Other properties of an animation

The next properties described in this section are all optional in that they have defaults that are appropriate for most animations.

AutoReverse

If you don't specify either **AutoReverse** or **RepeatBehavior** on an animation, that animation will run once, and run for the time specified as the **Duration**.

The **AutoReverse** property specifies whether a timeline plays in reverse after it reaches the end of its **Duration**. If you set it to **true**, the animation reverses after it reaches the end of its declared **Duration**, changing the value from its ending value (**To**) back to its starting value (**From**). This means that the animation effectively runs for double the time of its **Duration**.

RepeatBehavior

The **RepeatBehavior** property specifies either how many times a timeline plays, or a larger duration that the timeline should repeat within. By default, a timeline has an iteration count of "1x", which means it plays one time for its **Duration** and does not repeat.

You can cause the animation to run multiple iterations. For example, a value of "3x" causes the animation to run three times. Or, you can specify a different **Duration** for **RepeatBehavior**. That **Duration** should be longer than the **Duration** of the animation itself to be effective. For example, if you specify a **RepeatBehavior** of "0:0:10", for an animation that has a **Duration** of "0:0:2", that animation repeats five times. If these don't divide evenly, the animation gets truncated at the time that the **RepeatBehavior** time is reached, which might be partway through. Finally, you can specify the special value "Forever", which causes the animation to run infinitely until it's deliberately stopped.

For more info about **RepeatBehavior** values and the XAML syntax, see **RepeatBehavior**.

FillBehavior="Stop"

By default, when an animation ends, the animation leaves the property value as the final **To** or **By**-modified value even after its duration is surpassed. However, if you set the value of the **FillBehavior** property to **FillBehavior.Stop**, the value of the animated value reverts to whatever the value was before the animation was applied, or more precisely to the current effective value as determined by the dependency property system (for more info on this distinction, see Dependency properties overview).

BeginTime

By default, the **BeginTime** of an animation is "0:0:0", so it begins as soon as its containing **Storyboard** runs. You might change this if the **Storyboard** contains more

than one animation and you want to stagger the start times of the others versus an initial animation, or to create a deliberate short delay.

SpeedRatio

If you have more than one animation in a Storyboard you can change the time rate of one or more of the animations relative to the **Storyboard**. It's the parent **Storyboard** that ultimately controls how the Duration time elapses while the animations run. This property isn't used very often. For more info see SpeedRatio.

Defining more than one animation in a Storyboard

The contents of a Storyboard can be more than one animation definition. You might have more than one animation if you are applying related animations to two properties of the same target object. For example, you might change both the TranslateX and TranslateY properties of a TranslateTransform used as the RenderTransform of a UI element; this will cause the element to translate diagonally. You need two different animations to accomplish that, but you might want the animations to be part of the same **Storyboard** because you always want those two animations to be run together.

The animations don't have to be the same type, or target the same object. They can have different durations, and don't have to share any property values.

When the parent Storyboard runs, each of the animations within will run too.

The Storyboard class actually has a lot of the same animation properties as the animation types do, because both share the Timeline base class. Thus, a **Storyboard** can have a RepeatBehavior, or a BeginTime. You don't usually set these on a **Storyboard** though unless you want all the contained animations to have that behavior. As a general rule, any **Timeline** property as set on a **Storyboard** applies to all its child animations. If let unset, the **Storyboard** has an implicit duration that's calculated from the longest Duration value of the contained animations. An explicitly set Duration on a **Storyboard** that's shorter than one of its child animations will cause that animation to get cut off, which isn't usually desirable.

A storyboard can't contain two animations that attempt to target and animate the same property on the same object. If you try this, you'll get a runtime error when the storyboard tries to run. This restriction applies even if the animations don't overlap in time because of deliberately different BeginTime values and durations. If you really want to apply a more complex animation timeline to the same property in a single

storyboard, the way to do this is to use a key-frame animation. See Key-frame and easing function animations.

The animation system can apply more than one animation to the value of a property, if those inputs come from multiple storyboards. Using this behavior deliberately for simultaneously running storyboards isn't common. However it's possible that an app-defined animation that you apply to a control property will be modifying the **HoldEnd** value of an animation that was previously run as part of the control's visual state model.

Defining a storyboard as a resource

A **Storyboard** is the container that you put animation objects in. You typically define the **Storyboard** as a resource that is available to the object that you want to animate, either in page-level **Resources** or **Application.Resources**.

This next example shows how the previous example **Storyboard** would be contained in a page-level **Resources** definition, where the **Storyboard** is a keyed resource of the root **Page**. Note the x:Name attribute. This attribute is how you define a variable name for the **Storyboard**, so that other elements in XAML as well as code can refer to the **Storyboard** later.

XAML

```
<Page ...>
  <Page.Resources>
    <!-- Storyboard resource: Animates a rectangle's opacity. -->
    <Storyboard x:Name="myStoryboard">
      <DoubleAnimation
        Storyboard.TargetName="MyAnimatedRectangle"
        Storyboard.TargetProperty="Opacity"
        From="1.0" To="0.0" Duration="0:0:1"/>
    </Storyboard>
  </Page.Resources>
  <!--Page root element, UI definition-->
  <Grid>
    <Rectangle x:Name="MyAnimatedRectangle"
      Width="300" Height="200" Fill="Blue"/>
  </Grid>
</Page>
```

Defining resources at the XAML root of a XAML file such as page.xaml or app.xaml is a common practice for how to organize keyed resources in your XAML. You also can factor resources into separate files and merge them into apps or pages. For more info, see ResourceDictionary and XAML resource references.

> **Note**
>
> Windows Runtime XAML supports identifying resources either using the x:Key attribute or the x:Name attribute. Using x:Name attribute is more common for a Storyboard, because you'll want to reference it by variable name eventually, so that you can call its Begin method and run the animations. If you do use x:Key attribute, you'll need to use ResourceDictionary methods such as the Item indexer to retrieve it as a keyed resource and then cast the retrieved object to **Storyboard** to use the **Storyboard** methods.

Storyboards for visual states

You also put your animations within a **Storyboard** unit when you are declaring the visual state animations for a control's visual appearance. In that case the **Storyboard** elements you define go into a **VisualState** container that's nested more deeply in a **Style** (it's the **Style** that is the keyed resource). You don't need a key or name for your **Storyboard** in this case because it's the **VisualState** that has a target name that the **VisualStateManager** can invoke. The styles for controls are often factored into separate XAML **ResourceDictionary** files rather than placed in a page or app **Resources** collection. For more info, see Storyboarded animations for visual states.

Dependent and independent animations

At this point we need to introduce some important points about how the animation system works. In particular, animation interacts fundamentally with how a Windows Runtime app renders to the screen, and how that rendering uses processing threads. A Windows Runtime app always has a main UI thread, and this thread is responsible for updating the screen with current information. In addition, a Windows Runtime app has a composition thread, which is used for precalculating layouts immediately before they are shown. When you animate the UI, there's potential to cause a lot of work for the UI thread. The system must redraw large areas of the screen using fairly short time intervals between each refresh. This is necessary for capturing the latest property value of the animated property. If you're not careful, there's risk that an animation can make the UI less responsive, or will impact performance of other app features that are also on the same UI thread.

The variety of animation that is determined to have some risk of slowing down the UI thread is called a *dependent animation*. An animation not subject to this risk is an *independent animation*. The distinction between dependent and independent animations isn't just determined by animation types (**DoubleAnimation** and so on) as

we described earlier. Instead, it's determined by which specific properties you are animating, and other factors like inheritance and composition of controls. There are circumstances where even if an animation does change UI, the animation can have minimal impact to the UI thread, and can instead be handled by the composition thread as an independent animation.

An animation is independent if it has any of these characteristics:

- The **Duration** of the animation is 0 seconds (see Warning)
- The animation targets **UIElement.Opacity**
- The animation targets a sub-property value of these **UIElement** properties: **Transform3D**, **RenderTransform**, **Projection**, **Clip**
- The animation targets **Canvas.Left** or **Canvas.Top**
- The animation targets a **Brush** value and uses a **SolidColorBrush**, animating its **Color**
- The animation is an **ObjectAnimationUsingKeyFrames**

> ⚠ **Warning**
>
> In order for your animation to be treated as independent, you must explicitly set `Duration="0"`. For example, if you remove `Duration="0"` from this XAML, the animation is treated as dependent, even though the **KeyTime** of the frame is "0:0:0".

XAML

```
<Storyboard>
  <DoubleAnimationUsingKeyFrames
    Duration="0"
    Storyboard.TargetName="Button2"
    Storyboard.TargetProperty="Width">
    <DiscreteDoubleKeyFrame KeyTime="0:0:0" Value="200"/>
  </DoubleAnimationUsingKeyFrames>
</Storyboard>
```

If your animation doesn't meet these criteria, it's probably a dependent animation. By default, the animation system won't run a dependent animation. So during the process of developing and testing, you might not even be seeing your animation running. You can still use this animation, but you must specifically enable each such dependent animation. To enable your animation, set the **EnableDependentAnimation** property of the animation object to **true**. (Each **Timeline** subclass that represents an animation has a different implementation of the property but they're all named `EnableDependentAnimation`.)

The requirement of enabling dependent animations falling onto the app developer is a conscious design aspect of the animation system and the development experience. We want developers to be aware that animations do have a performance cost for the responsiveness of your UI. Poorly performing animations are difficult to isolate and debug in a full-scale app. So it's better to turn on only the dependent animations you really need for your app's UI experience. We didn't want to make it too easy to compromise your app's performance because of decorative animations that use a lot of cycles. For more info on performance tips for animation, see Optimize animations and media.

As an app developer, you can also choose to apply an app-wide setting that always disables dependent animations, even those where **EnableDependentAnimation** is **true**. See **Timeline.AllowDependentAnimations**.

> **Tip**
>
> If you're using the Animation Pane in Blend for Visual Studio 2019, whenever you attempt to apply a dependent animation to a visual state property, warnings will be displayed in the designer. Warnings will not show in the build output or Error List. If you're editing XAML by hand, the designer will not show a warning. At runtime when debugging, the Output pane's Debug output will show a warning that the animation is not independent and will be skipped.

Starting and controlling an animation

Everything we've shown you so far doesn't actually cause an animation to run or be applied! Until the animation is started and is running, the value changes that an animation is declaring in XAML are latent and won't happen yet. You must explicitly start an animation in some way that's related to the app lifetime or the user experience. At the simplest level, you start an animation by calling the **Begin** method on the **Storyboard** that's the parent for that animation. You can't call methods from XAML directly, so whatever you do to enable your animations, you'll be doing it from code. That will either be the code-behind for the pages or components of your app, or perhaps the logic of your control if you're defining a custom control class.

Typically, you'll call **Begin** and just let the animation run to its duration completion. However, you can also use **Pause**, **Resume** and **Stop** methods to control the **Storyboard** at run-time, as well as other APIs that are used for more advanced animation control scenarios.

When you call **Begin** on a storyboard that contains an animation that repeats infinitely (`RepeatBehavior="Forever"`), that animation runs until the page containing it is unloaded, or you specifically call **Pause** or **Stop**.

Starting an animation from app code

You can either start animations automatically, or in response to user actions. For the automatic case, you typically use an object lifetime event such as **Loaded** to act as the animation trigger. The **Loaded** event is a good event to use for this because at that point the UI is ready for interaction, and the animation won't be cut off at the beginning because another part of UI was still loading.

In this example, the **PointerPressed** event is attached to the rectangle so that when the user clicks the rectangle, the animation begins.

XAML

```
<Rectangle PointerPressed="Rectangle_Tapped"
  x:Name="MyAnimatedRectangle"
  Width="300" Height="200" Fill="Blue"/>
```

The event handler start the **Storyboard** (the animation) by using the **Begin** method of the **Storyboard**.

C#

```
myStoryboard.Begin();
```

C++/WinRT

```
myStoryboard().Begin();
```

C++

```
myStoryboard->Begin();
```

VB

```
myStoryBoard.Begin()
```

You can handle the **Completed** event if you want other logic to run after the animation has finished applying values. Also, for troubleshooting property system/animation

interactions, the **GetAnimationBaseValue** method can be useful.

> **Tip**
>
> Whenever you are coding for an app scenario where you are starting an animation from app code, you might want to review again whether an animation or transition already exists in the animation library for your UI scenario. The library animations enable a more consistent UI experience across all Windows Runtime apps, and are easier to use.

Animations for visual states

The run behavior for a **Storyboard** that's used to define a control's visual state is different from how an app might run a storyboard directly. As applied to a visual state definition in XAML, the **Storyboard** is an element of a containing **VisualState**, and the state as a whole is controlled by using the **VisualStateManager** API. Any animations within will run according to their animation values and **Timeline** properties when the containing **VisualState** is used by a control. For more info, see Storyboards for visual states. For visual states, the apparent **FillBehavior** is different. If a visual state is changed to another state, all the property changes applied by the previous visual state and its animations are canceled, even if the new visual state doesn't specifically apply a new animation to a property.

Storyboard and EventTrigger

There is one way to start an animation that can be declared entirely in XAML. However, this technique isn't widely used anymore. It's a legacy syntax from WPF and early versions of Silverlight prior to **VisualStateManager** support. This **EventTrigger** syntax still works in Windows Runtime XAML for import/compatibility reasons, but only works for a trigger behavior based on the **FrameworkElement.Loaded** event; attempting to trigger off other events will throw exceptions or fail to compile. For more info, see **EventTrigger** or **BeginStoryboard**.

Animating XAML attached properties

It's not a common scenario, but you can apply an animated value to a XAML attached property. For more info on what attached properties are and how they work, see Attached properties overview. Targeting an attached property requires a property-path

syntax that encloses the property name in parentheses. You can animate the built-in attached properties such as **Canvas.ZIndex** by using an **ObjectAnimationUsingKeyFrames** that applies discrete integer values. However, an existing limitation of the Windows Runtime XAML implementation is that you cannot animate a custom attached property.

More animation types, and next steps for learning about animating your UI

Up to now, we've shown the custom animations that are animating between two values, and then linearly interpolating the values as necessary while the animation runs. These are called **From/To/By** animations. But there's another animation type that enables you to declare intermediate values that fall between the start and end. These are called *key-frame animations*. There's also a way to alter the interpolation logic on either a **From/To/By** animation or a key-frame animation. This involves applying an easing function. For more info on these concepts, see Key-frame and easing function animations.

Related topics

- Property-path syntax
- Dependency properties overview
- Key-frame and easing function animations
- Storyboarded animations for visual states
- Control templates
- **Storyboard**
- **Storyboard.TargetProperty**

Key-frame animations and easing function animations

Article • 06/24/2021

Linear key-frame animations, key-frame animations with a **KeySpline** value, or easing functions are three different techniques for approximately the same scenario: creating a storyboarded animation that's a bit more complex, and that uses a nonlinear animation behavior from a starting state to an end state.

Prerequisites

Make sure you've read the Storyboarded animations topic. This topic builds on the animation concepts that were explained in Storyboarded animations and won't go over them again. For example, Storyboarded animations describes how to target animations, storyboards as resources, the **Timeline** property values such as **Duration**, **FillBehavior**, and so on.

Animating using key-frame animations

Key-frame animations permit more than one target value that is reached at a point along the animation timeline. In other words, each key frame can specify a different intermediate value, and the last key frame reached is the final animation value. By specifying multiple values to animate, you can make more complex animations. Key-frame animations also enable different interpolation logic, which are each implemented as a different **KeyFrame** subclass per animation type. Specifically, each key-frame animation type has a **Discrete**, **Linear**, **Spline** and **Easing** variation of its **KeyFrame** class for specifying its key frames. For example, to specify an animation that targets a **Double** and uses key frames, you could declare key frames with **DiscreteDoubleKeyFrame**, **LinearDoubleKeyFrame**, **SplineDoubleKeyFrame**, and **EasingDoubleKeyFrame**. You can use any and all of these types within a single **KeyFrames** collection, to change the interpolation each time a new key frame is reached.

For interpolation behavior, each key frame controls the interpolation until its **KeyTime** time is reached. Its **Value** is reached at that time also. If there are more key frames beyond, the value then becomes the starting value for the next key frame in a sequence.

At the start of the animation, if no key frame with **KeyTime** of "0:0:0" exists, the starting value is whatever the non-animated value of the property is. This is similar to how a **From/To/By** animation acts if there is no **From**.

The duration of a key-frame animation is implicitly the duration equal to the highest **KeyTime** value set in any of its key frames. You can set an explicit **Duration** if you want, but be careful it's not shorter than a **KeyTime** in your own key frames or you'll cut off part of the animation.

In addition to **Duration**, you can set all the **Timeline** based properties on a key-frame animation, like you can with a **From/To/By** animation, because the key-frame animation classes also derive from **Timeline**. These are:

- **AutoReverse**: once the last key frame is reached, the frames are repeated in reverse order from the end. This doubles the apparent duration of the animation.
- **BeginTime**: delays the start of the animation. The timeline for the **KeyTime** values in the frames doesn't start counting until **BeginTime** is reached, so there's no risk of cutting off frames
- **FillBehavior**: controls what happens when the last key frame is reached. **FillBehavior** has no effect on any intermediate key frames.
- **RepeatBehavior**:
  - If set to **Forever**, then the key frames and their timeline repeat infinitely.
  - If set to an iteration count, the timeline repeats that many times.
  - If set to a **Duration**, the timeline repeats until that time is reached. This might truncate the animation part way through the key frame sequence, if it's not an integer factor of the timeline's implicit duration.
- **SpeedRatio** (not commonly used)

Linear key frames

Linear key frames result in a simple linear interpolation of the value until the frame's **KeyTime** is reached. This interpolation behavior is the most similar to the simpler **From/To/By** animations described in the Storyboarded animations topic.

Here's how to use a key-frame animation to scale the render height of a rectangle, using linear key frames. This example runs an animation where the height of the rectangle increases slightly and linearly for the first 4 seconds, then scales rapidly for the last second until the rectangle is double the starting height.

XML

```
<StackPanel>
    <StackPanel.Resources>
        <Storyboard x:Name="myStoryboard">
            <DoubleAnimationUsingKeyFrames
              Storyboard.TargetName="myRectangle"
              Storyboard.TargetProperty="(UIElement.RenderTransform).
(ScaleTransform.ScaleY)">
```

```
                <LinearDoubleKeyFrame Value="1" KeyTime="0:0:0"/>
                <LinearDoubleKeyFrame Value="1.2" KeyTime="0:0:4"/>
                <LinearDoubleKeyFrame Value="2" KeyTime="0:0:5"/>
            </DoubleAnimationUsingKeyFrames>
        </Storyboard>
    </StackPanel.Resources>
</StackPanel>
```

Discrete key frames

Discrete key frames don't use any interpolation at all. When a **KeyTime** is reached, the new **Value** is simply applied. Depending on which UI property is being animated, this often produces an animation that appears to "jump". Be certain that this is the aesthetic behavior that you really want. You can minimize the apparent jumps by increasing the number of key frames you declare, but if a smooth animation is your goal, you might be better off using linear or spline key frames instead.

> **Note**
>
> Discrete key frames are the only way to animate a value that isn't of type **Double**, **Point**, and **Color**, with a **DiscreteObjectKeyFrame**. We'll discuss this in more detail later in this topic.

Spline key frames

A spline key frame creates a variable transition between values according to the value of the **KeySpline** property. This property specifies the first and second control points of a Bezier curve, which describes the acceleration of the animation. Basically, a **KeySpline** defines a function-over-time relationship where the function-time graph is the shape of that Bezier curve. You typically specify a **KeySpline** value in a XAML shorthand attribute string that has four **Double** values separated by spaces or commas. These values are "X,Y" pairs for two control points of the Bezier curve. "X" is time and "Y" is the function modifier to the value. Each value should always be between 0 and 1 inclusive. Without control point modification to a **KeySpline**, the straight line from 0,0 to 1,1 is the representation of a function over time for a linear interpolation. Your control points change the shape of that curve and thus the behavior of the function over time for the spline animation. It's probably best to see this visually as a graph. You can run the Silverlight key-spline visualizer sample in a browser to see how the control points modify the curve and how a sample animation runs when using it as a **KeySpline** value.

This next example shows three different key frames applied to an animation, with the last one being a key spline animation for a **Double** value (**SplineDoubleKeyFrame**). Note the string "0.6,0.0 0.9,0.00" applied for **KeySpline**. This produces a curve where the animation appears to run slowly at first but then rapidly reaches the value just before the **KeyTime** is reached.

XML

```
<Storyboard x:Name="myStoryboard">
    <!-- Animate the TranslateTransform's X property
        from 0 to 350, then 50,
        then 200 over 10 seconds. -->
    <DoubleAnimationUsingKeyFrames
        Storyboard.TargetName="MyAnimatedTranslateTransform"
        Storyboard.TargetProperty="X"
        Duration="0:0:10" EnableDependentAnimation="True">

        <!-- Using a LinearDoubleKeyFrame, the rectangle moves
            steadily from its starting position to 500 over
            the first 3 seconds.  -->
        <LinearDoubleKeyFrame Value="500" KeyTime="0:0:3"/>

        <!-- Using a DiscreteDoubleKeyFrame, the rectangle suddenly
            appears at 400 after the fourth second of the animation. -->
        <DiscreteDoubleKeyFrame Value="400" KeyTime="0:0:4"/>

        <!-- Using a SplineDoubleKeyFrame, the rectangle moves
            back to its starting point. The
            animation starts out slowly at first and then speeds up.
            This KeyFrame ends after the 6th second. -->
        <SplineDoubleKeyFrame KeySpline="0.6,0.0 0.9,0.00" Value="0"
KeyTime="0:0:6"/>
    </DoubleAnimationUsingKeyFrames>
</Storyboard>
```

Easing key frames

An easing key frame is a key frame where interpolation is being applied, and the function over time of the interpolation is controlled by several pre-defined mathematical formulas. You can actually produce much the same result with a spline key frame as you can with some of the easing function types, but there are also some easing functions, such as **BackEase**, that you can't reproduce with a spline.

To apply an easing function to an easing key frame, you set the **EasingFunction** property as a property element in XAML for that key frame. For the value, specify an object element for one of the easing function types.

This example applies a CubicEase and then a BounceEase as successive key frames to a DoubleAnimation to create a bouncing effect.

XML

```
<Storyboard x:Name="myStoryboard">
    <DoubleAnimationUsingKeyFrames Duration="0:0:10"
        Storyboard.TargetProperty="Height"
        Storyboard.TargetName="myEllipse">

        <!-- This keyframe animates the ellipse up to the crest
            where it slows down and stops. -->
        <EasingDoubleKeyFrame Value="-300" KeyTime="00:00:02">
            <EasingDoubleKeyFrame.EasingFunction>
                <CubicEase/>
            </EasingDoubleKeyFrame.EasingFunction>
        </EasingDoubleKeyFrame>

        <!-- This keyframe animates the ellipse back down and makes
            it bounce. -->
        <EasingDoubleKeyFrame Value="0" KeyTime="00:00:06">
            <EasingDoubleKeyFrame.EasingFunction>
                <BounceEase Bounces="5"/>
            </EasingDoubleKeyFrame.EasingFunction>
        </EasingDoubleKeyFrame>
    </DoubleAnimationUsingKeyFrames>
</Storyboard>
```

This is just one easing function example. We'll cover more in the next section.

Easing functions

Easing functions allow you to apply custom mathematical formulas to your animations. Mathematical operations are often useful to produce animations that simulate real-world physics in a 2-D coordinate system. For example, you may want an object to realistically bounce or behave as though it were on a spring. You could use key frame or even **From/To/By** animations to approximate these effects but it would take a significant amount of work and the animation would be less accurate than using a mathematical formula.

Easing functions can be applied to animations in three ways:

- By using an easing keyframe in a keyframe animation, as described in the previous section. Use EasingColorKeyFrame.EasingFunction, EasingDoubleKeyFrame.EasingFunction, or EasingPointKeyFrame.EasingFunction.
- By setting the **EasingFunction** property on one of the **From/To/By** animation types. Use ColorAnimation.EasingFunction, DoubleAnimation.EasingFunction or

PointAnimation.EasingFunction.

- By setting GeneratedEasingFunction as part of a VisualTransition. This is specific to defining visual states for controls; for more info, see GeneratedEasingFunction or Storyboards for visual states.

Here is a list of the easing functions:

- BackEase: Retracts the motion of an animation slightly before it begins to animate in the path indicated.
- BounceEase: Creates a bouncing effect.
- CircleEase: Creates an animation that accelerates or decelerates using a circular function.
- CubicEase: Creates an animation that accelerates or decelerates using the formula f(t) = t3.
- ElasticEase: Creates an animation that resembles a spring oscillating back and forth until it comes to rest.
- ExponentialEase: Creates an animation that accelerates or decelerates using an exponential formula.
- PowerEase: Creates an animation that accelerates or decelerates using the formula f(t) = tp where p is equal to the Power property.
- QuadraticEase: Creates an animation that accelerates or decelerates using the formula f(t) = t2.
- QuarticEase: Creates an animation that accelerates or decelerates using the formula f(t) = t4.
- QuinticEase: Create an animation that accelerates or decelerates using the formula f(t) = t5.
- SineEase: Creates an animation that accelerates or decelerates using a sine formula.

Some of the easing functions have their own properties. For example, BounceEase has two properties Bounces and Bounciness that modify the function-over-time behavior of that particular **BounceEase**. Other easing functions such as CubicEase don't have properties other than the EasingMode property that all easing functions share, and always produce the same function-over-time behavior.

Some of these easing functions have a bit of overlap, depending on how you set properties on the easing functions that have properties. For example, QuadraticEase is exactly the same as a PowerEase with Power equal to 2. And CircleEase is basically a default-value ExponentialEase.

The BackEase easing function is unique because it can change the value outside of the normal range as set by **From**/**To** or values of key frames. It starts the animation by

changing the value in the opposite direction as would be expected from a normal **From/To** behavior, goes back to the **From** or starting value again, and then runs the animation as normal.

In an earlier example, we showed how to declare an easing function for a key-frame animation. This next sample applies an easing function to a **From/To/By** animation.

XML

```
<StackPanel x:Name="LayoutRoot" Background="White">
    <StackPanel.Resources>
        <Storyboard x:Name="myStoryboard">
            <DoubleAnimation From="30" To="200" Duration="00:00:3"
                Storyboard.TargetName="myRectangle"
                Storyboard.TargetProperty="(UIElement.RenderTransform).
(ScaleTransform.ScaleY)">
                <DoubleAnimation.EasingFunction>
                    <BounceEase Bounces="2" EasingMode="EaseOut"
                                Bounciness="2"/>
                </DoubleAnimation.EasingFunction>
            </DoubleAnimation>
        </Storyboard>
    </StackPanel.Resources>
    <Rectangle x:Name="myRectangle" Fill="Blue" Width="200" Height="30"/>
</StackPanel>
```

When an easing function is applied to a **From/To/By** animation, it's changing the function- over-time characteristics of how the value interpolates between the **From** and **To** values over the **Duration** of the animation. Without an easing function, that would be a linear interpolation.

Discrete object value animations

One type of animation deserves special mention because it's the only way you can apply an animated value to properties that aren't of type **Double**, **Point**, or **Color**. This is the key-frame animation **ObjectAnimationUsingKeyFrames**. Animating using **Object** values is different because there's no possibility of interpolating the values between the frames. When the frame's **KeyTime** is reached, the animated value is immediately set to the value specified in the key frame's **Value**. Because there's no interpolation, there's only one key frame you use in the **ObjectAnimationUsingKeyFrames** key frames collection: **DiscreteObjectKeyFrame**.

The **Value** of a **DiscreteObjectKeyFrame** is often set using property element syntax, because the object value you are trying to set often is not expressible as a string to fill

Value in attribute syntax. You can still use attribute syntax if you use a reference such as StaticResource.

One place you'll see an **ObjectAnimationUsingKeyFrames** used in the default templates is when a template property references a **Brush** resource. These resources are **SolidColorBrush** objects, not just a **Color** value, and they use resources that are defined as system themes (**ThemeDictionaries**). They can be assigned directly to a **Brush**-type value such as **TextBlock.Foreground** and don't need to use indirect targeting. But because a **SolidColorBrush** is not **Double**, **Point**, or **Color**, you have to use a **ObjectAnimationUsingKeyFrames** to use the resource.

XML

```
<Style x:Key="TextButtonStyle" TargetType="Button">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="Button">
                <Grid Background="Transparent">
                    <TextBlock x:Name="Text"
                        Text="{TemplateBinding Content}"/>
                    <VisualStateManager.VisualStateGroups>
                        <VisualStateGroup x:Name="CommonStates">
                            <VisualState x:Name="Normal"/>
                            <VisualState x:Name="PointerOver">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="Text" Storyboard.TargetProperty="Foreground">
                                        <DiscreteObjectKeyFrame KeyTime="0"
Value="{StaticResource ApplicationPointerOverForegroundThemeBrush}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
                            <VisualState x:Name="Pressed">
                                <Storyboard>
                                    <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="Text" Storyboard.TargetProperty="Foreground">
                                        <DiscreteObjectKeyFrame KeyTime="0"
Value="{StaticResource ApplicationPressedForegroundThemeBrush}"/>
                                    </ObjectAnimationUsingKeyFrames>
                                </Storyboard>
                            </VisualState>
...
                        </VisualStateGroup>
                    </VisualStateManager.VisualStateGroups>
                </Grid>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

You also might use **ObjectAnimationUsingKeyFrames** to animate properties that use an enumeration value. Here's another example from a named style that comes from the Windows Runtime default templates. Note how it sets the **Visibility** property that takes a **Visibility** enumeration constant. In this case you can set the value using attribute syntax. You only need the unqualified constant name from an enumeration for setting a property with an enumeration value, for example "Collapsed".

XML

```
<Style x:Key="BackButtonStyle" TargetType="Button">
    <Setter Property="Template">
      <Setter.Value>
        <ControlTemplate TargetType="Button">
          <Grid x:Name="RootGrid">
            <VisualStateManager.VisualStateGroups>
              <VisualStateGroup x:Name="CommonStates">
              <VisualState x:Name="Normal"/>
...           <VisualState x:Name="Disabled">
                <Storyboard>
                  <ObjectAnimationUsingKeyFrames
Storyboard.TargetName="RootGrid" Storyboard.TargetProperty="Visibility">
                    <DiscreteObjectKeyFrame Value="Collapsed" KeyTime="0"/>
                  </ObjectAnimationUsingKeyFrames>
                </Storyboard>
              </VisualState>
            </VisualStateGroup>
...
          </VisualStateManager.VisualStateGroups>
        </Grid>
      </ControlTemplate>
    </Setter.Value>
  </Setter>
</Style>
```

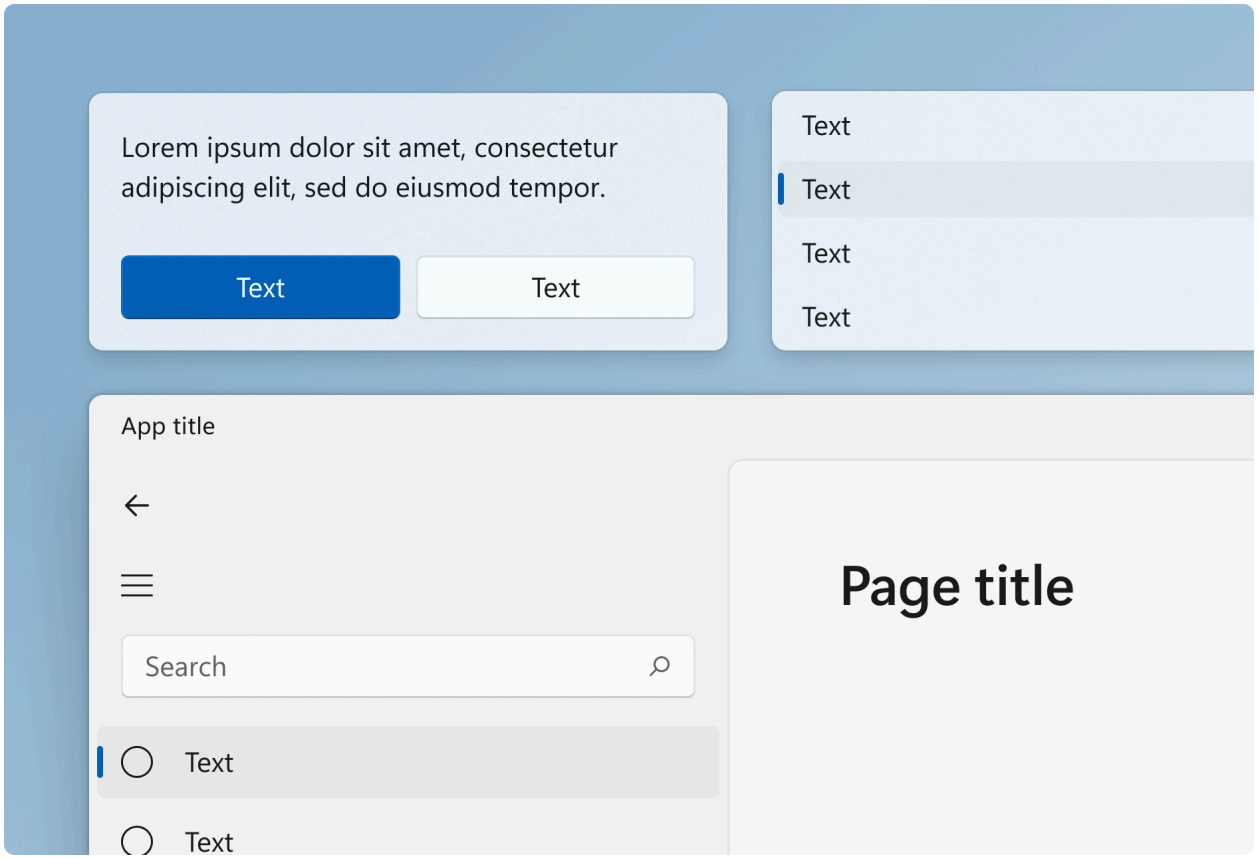
You can use more than one **DiscreteObjectKeyFrame** for an **ObjectAnimationUsingKeyFrames** frame set. This might be an interesting way to create a "slide show" animation by animating the value of **Image.Source**, as an example scenario for where multiple object values might be useful.

Related topics

- Property-path syntax
- Dependency properties overview
- **Storyboard**
- **Storyboard.TargetProperty**

Geometry in Windows

Article • 11/05/2024

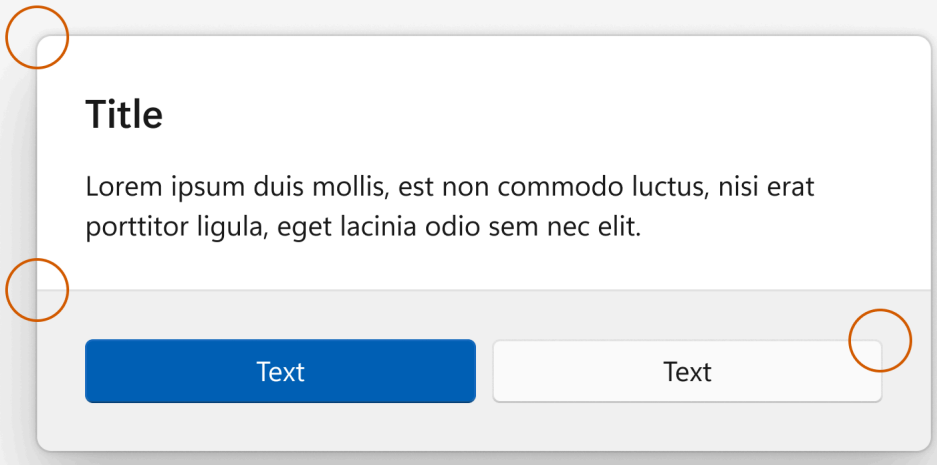


Geometry describes the shape, size and position of UI elements on screen. These fundamental design elements help experiences feel coherent across the entire design system.

Windows geometry has been crafted to support modern app experiences. Progressively rounded corners, nested elements, and consistent gutters combine to create a soft, calm, and approachable effect that emphasizes unity of purpose and ease of use.

> **Tip**
>
> This article describes how the Fluent Design language is applied to Windows apps. For more information, see Fluent Design - Shapes.

Rounded corners



Windows 11 applies rounded corners to all top-level app windows. The same applies to most common controls such as Button and ListView. (For more information, see Use the latest common controls.)

Windows 11 uses three levels of rounding depending on what UI component is being rounded and how that component is arranged relative to neighboring elements.

Expand table

Corner radius	Usage
8px	Top-level containers such as app windows, flyouts and dialogs are rounded using an 8px corner radius.
4px	In-page elements such as buttons and list backplates are rounded using a 4px corner radius.
0px	Straight edges that intersect with other straight edges are not rounded.
0px	Window corners are not rounded when windows are snapped or maximized.

Corner radius

Article • 03/23/2022

Starting with version 2.2 of WinUI, the default style for many controls has been updated to use rounded corners. These new styles are intended to evoke warmth and trust, and make the UI easier for users to visually process.

Here are two Button controls, the first without rounded corners and the second using a rounded corner style.

WinUI provides you with the updated styles for both WinUI and platform controls. See Customization options, for details on how to customize rounded corners.

> **ⓘ Important**
>
> Some controls are available both in the platform (Windows.UI.Xaml.Controls) and in WinUI (Microsoft.UI.Xaml.Controls); for example, **TreeView** or **ColorPicker**. When you use WinUI in your app, you should use the WinUI version of the control. Corner rounding might be applied inconsistently in the platform version when used with WinUI.

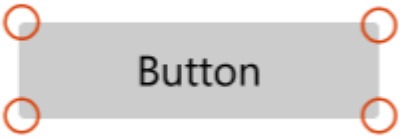
> **Important APIs**: Control.CornerRadius property

Default control designs

There are three areas of the controls where the rounded corner styles are used: rectangular elements, flyout elements, and bar elements.

Corners of rectangle UI elements

- These UI elements include basic controls like buttons that users see on screen at all times.
- The default radius value we use for these UI elements is **4px**.



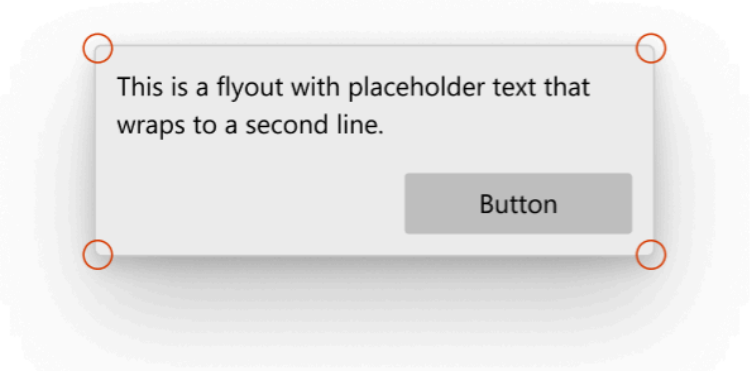
Controls

- AutoSuggestBox
- Buttons
  - ContentDialog buttons
- CalendarDatePicker
- CheckBox
  - TreeView, GridView, and ListView multi-select check boxes
- Color picker
- CommandBar
- ComboBox
- DatePicker
- DropDownButton
- Expander
- FlipView
- GridView and ListView
  - AutoSuggestBox, ComboBox, DatePicker, MenuBar, NavigationView, TimePicker, TreeView list
- InfoBar
- Inking controls
- Media playback
- MenuBar
- NumberBox
- PasswordBox
- RichEditBox
- SplitButton
- TextBox
- TimePicker
- ToggleButton
- ToggleSplitButton

Corners of flyout and overlay UI elements

- These can be transient UI elements that appear on screen temporarily, like MenuFlyout, or elements that overlay other UI, like TabView tabs.

- The default radius value we use for these UI elements is **8px**.



Controls

- CommandBarFlyout
- ContentDialog
- Flyout
- MenuFlyout
- TabView tabs
- TeachingTip
- ToolTip (uses **4px** radius due to small size)
- Flyout part (when open)
  - AutoSuggestBox
  - CalendarDatePicker
  - ComboBox
  - DatePicker
  - DropDownButton
  - Inking control
  - MenuBar
  - NumberBox
  - SplitButton
  - TimePicker
  - ToggleSplitButton

Bar elements

- These UI elements are shaped like bars or lines; for example, ProgressBar.
- The default radius values we use here are **4px**.

Controls

- NavigationView selection indicator
- ProgressBar
- ScrollBar
- Slider
  - ColorPicker color slider
  - MediaTransportControls seek bar slider

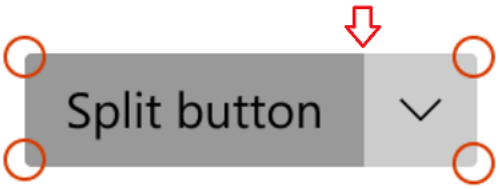
Customization options

The default corner radii values that we provide are not set in stone and there are a few ways you can easily modify the amount of rounding on the corners. This can be done through two global resources, or through the CornerRadius property directly on the control, depending on the level of customization granularity you want.

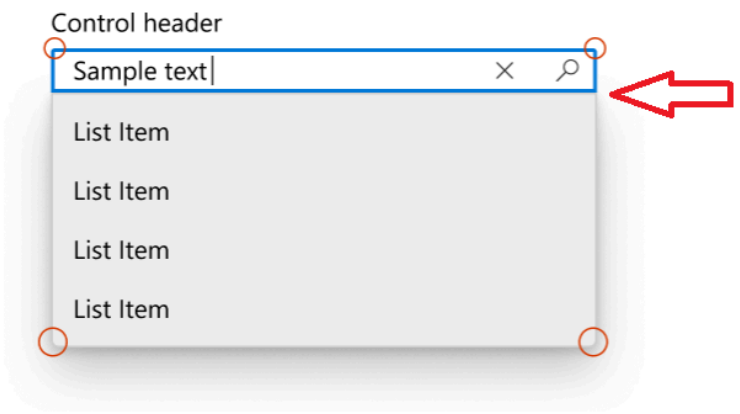
When not to round

There are instances where the corner of a control should not be rounded, and we don't round these by default.

- When multiple UI elements that are housed inside a container touch each other, such as the two parts of a SplitButton. There should be no space when they contact.



- When a flyout UI element is connected to a UI that invokes the flyout on one side.



Page or app-wide CornerRadius changes

There are 2 app resources that control the corner radii of all the controls:

- `ControlCornerRadius` - default is 4px.
- `OverlayCornerRadius` - default is 8px.

If you override the value of these resources at any scope, it will affect all controls within that scope accordingly.

This means if you want to change the roundness of all controls where roundness could be applied, you can define both resources at the app level with the new CornerRadius values like this:

XAML

```
<Application
    xmlns="http://schemas.microsoft.com/winfx/2006/xamlpresentation"
    xmlns:x="http://schemas.micosoft.com/winfx/2006/xaml"
    xmlns:control="using:Microsoft.UI.Xaml.Controls">
    <Application.Resources>
      <controls:XamlControlsResources>
        <controls:XamlControlsResources.MergedDictionaries>
          <ResourceDictionary>
            <CornerRadius x:Key="OverlayCornerRadius">0</CornerRadius>
            <CornerRadius x:Key="ControlCornerRadius">0</CornerRadius>
          </ResourceDictionary>
        </controls:XamlControlsResources.MergedDictionaries>
      </controls:XamlControlsResources>
    </Application.Resources>
</Application>
```

Alternatively, if you want to change all controls' roundness within a particular scope, like at a page or container level, you can follow a similar pattern:

XAML

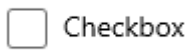
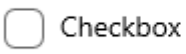
```
<Grid>
    <Grid.Resources>
        <CornerRadius x:Key="ControlCornerRadius">8</CornerRadius>
    </Grid.Resources>
    <Button Content="Button"/>
</Grid>
```

> **Note**
>
> The `OverlayCornerRadius` resource must be defined at the app level in order to take effect.
>
> This is because popup and flyouts are not in to the page's visual tree, they are added to the Popup Root. The resource resolution system does not properly traverse the Popup Root visual tree into the Page's visual tree.

Per-control CornerRadius changes

You can modify the CornerRadius property on controls directly if you want to change the roundness of only a select number of controls.

Expand table

Default	Property modified
	
`<CheckBox Content="Checkbox"/>`	`<CheckBox Content="Checkbox" CornerRadius="5"/>`

Not all controls' corners will respond to their `CornerRadius` property being modified. To ensure that the control whose corners you wish to round will indeed respond to their `CornerRadius` property the way you expect, first check that the `ControlCornerRadius` or `OverlayCornerRadius` global resources affect the control in question. If they do not, check that the control you wish to round has corners at all. Many of our controls do not render actual edges and therefore cannot make proper use of the `CornerRadius` property.

Basing custom styles on WinUI

You can base your custom styles on the WinUI rounded corner styles by specifying the correct `BasedOn` attribute in your style. For example to create a custom button style based on WinUI button style, do the following:

XAML

```
<Style x:Key="MyCustomButtonStyle" BasedOn="{StaticResource
DefaultButtonStyle}">
    ...
</Style>
```

In general, WinUI control styles follow a consistent naming convention: "DefaultXYZStyle" where "XYZ" is the name of the control. For full reference, you can browse the XAML files in the WinUI repository.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

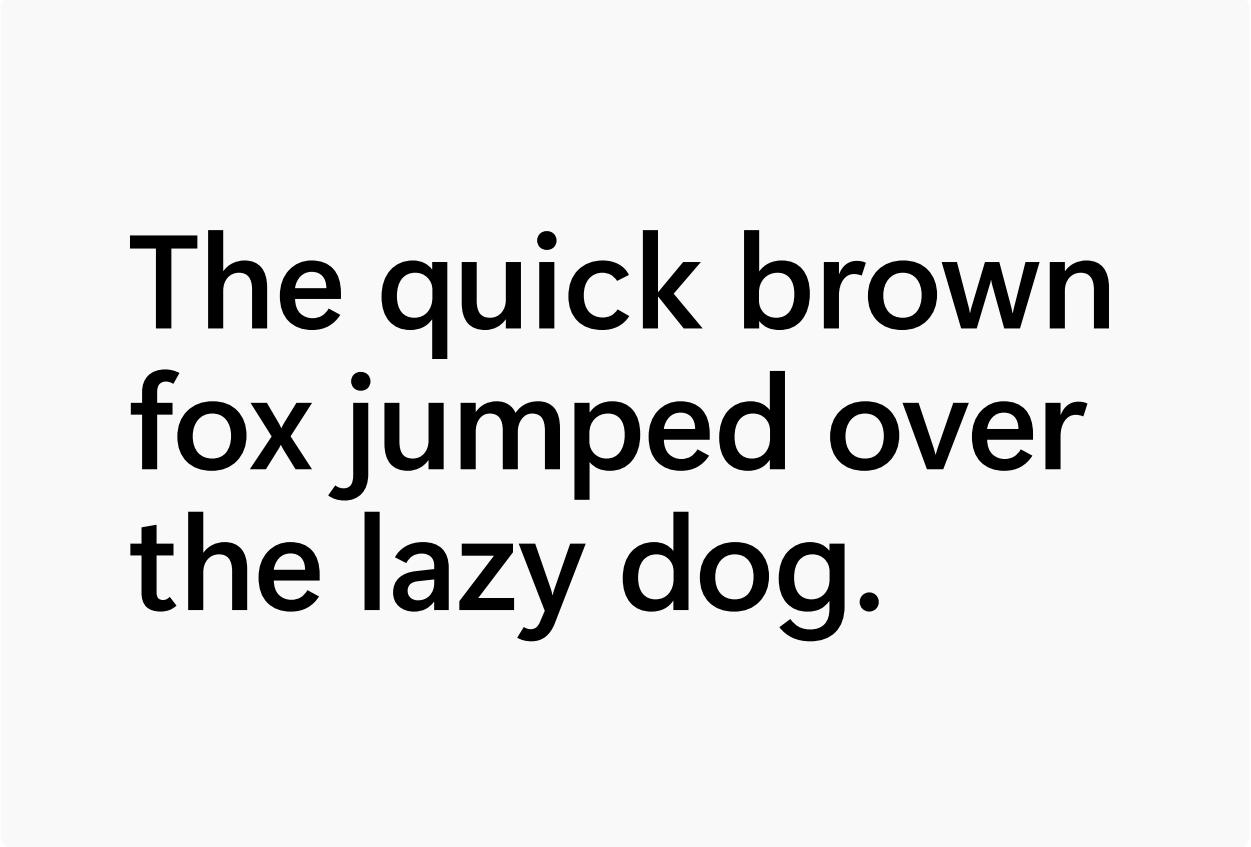
Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Typography in Windows

Article • 11/05/2024



As the visual representation of language, typography's main task is to communicate information. The Windows type system helps you create structure and hierarchy in your content in order to maximize legibility and readability in your UI.

Segoe UI Variable is the new system font for Windows. It is a refreshed take on the classic Segoe and uses variable font technology to dynamically provide great legibility at very small sizes, and improved outlines at display sizes.

> **Tip**
>
> This article describes how the **Fluent Design language** is applied to Windows apps. For more information, see **Fluent Design - Typography**.

Using Segoe Fluent Variable

Segoe UI Variable supports two axes for finer control of text: **weight** and **optical size**.

- The weight axis (`wght`) is incremental with weights from Thin (100) to Bold (700).

- The optical size axis (`opsz`) is automatic and on by default. It controls the shape and size of the counters in the font, to prioritize legibility at the small sizes and personality at the large sizes (for optical scaling from 8pt to 36pt).

When using XAML common controls, the Segoe UI Variable font will be selected by default for supported languages. When this font or another variable font with an optical axis is used, the optical size will automatically match the requested font-size. When using HTML, optical scaling is also automatic, but you will need to specify the Segoe UI Variable font in CSS.

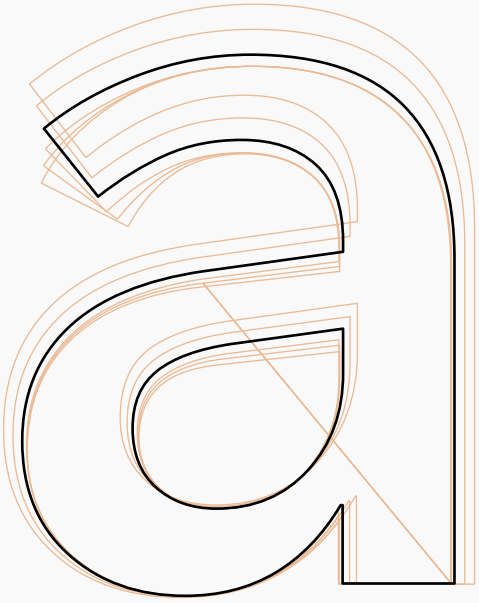
Weights

Expand table

Weight name	Weight axis value	Visual
Light	300	
Semilight	350	
Regular	400	
Semibold	600	

Weight name	Weight axis value	Visual
Bold	700	Segoe

Optical axis



Typography best practices in Windows 11

Windows 11 uses Segoe UI Variable with the following attributes based on the context in which the text is being displayed.

Expand table

Attribute	Value	Notes
Weight	Regular, Semibold	Use regular weight for most text, use Semibold for titles
Alignment	Left, Center	Align left by default, Align center only in rare cases such as text below icons
Minimum values	14px Semibold, 12px Regular	Text smaller than these sizes and weights are illegible in some languages
Casing	Sentence case	Use sentence casing for all UI text, including titles
Truncation	Ellipses and clipping	Use ellipses in most cases; clipping is only used in rare cases

Examples

Open the WinUI 3 Gallery app and see Typography principles in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Typography in Windows Apps

As the visual representation of language, typography's main task is to communicate information. Its style should never get in the way of that goal. In this article, we'll discuss how to style typography in your Windows app to help users understand content easily and efficiently.

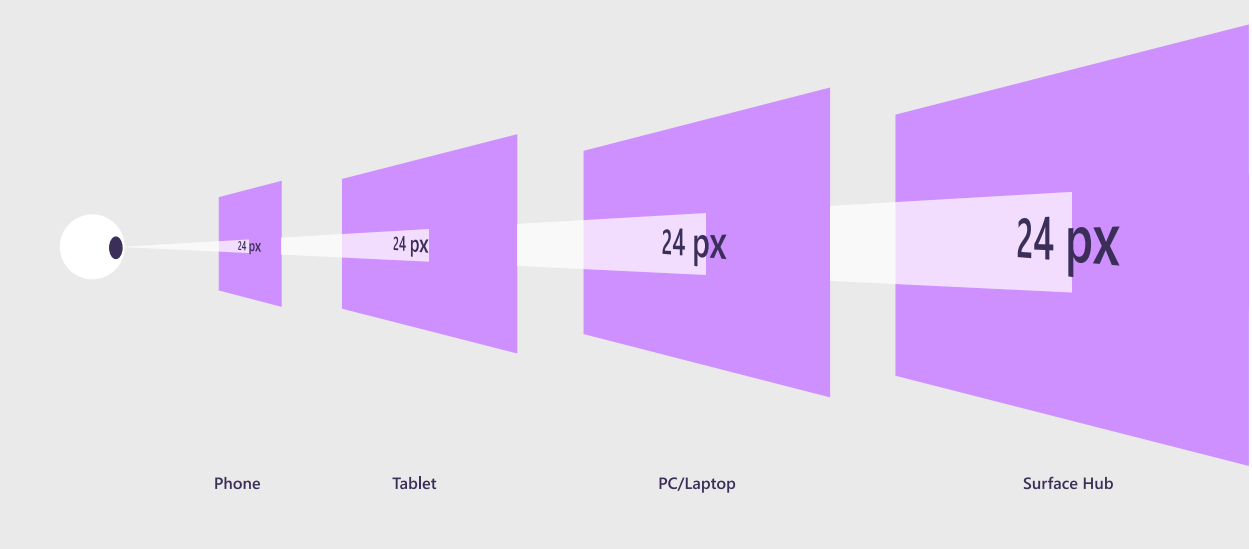
Font

You should use one font throughout your app's UI, and we recommend sticking with the default font for Windows apps, **Segoe UI Variable**. It's designed to maintain optimal legibility across sizes and pixel densities and offers a clean, light, and open aesthetic that complements the content of the system.

To display non-English languages or to select a different font for your app, please see Languages and Fonts for our recommended fonts for Windows apps.

Size and scaling

Font sizes in XAML apps automatically scale on all devices. The scaling algorithm ensures that a 24 px font on a large screen 10 feet away is just as legible as a 24 px font on a small screen that's a few inches away.



Because of how the scaling system works, you're designing in effective pixels, not actual physical pixels, and you shouldn't have to alter font sizes for different screens sizes or resolutions.

Hierarchy

Users rely on visual hierarchy when scanning a page: headers summarize content, and body text provides more detail. To create a clear visual hierarchy in your app, follow the Windows type ramp.

Type Hierarchy

Hierarchy

lets the reader know where to look first

Hierarchy can be created by using size, weight, color and spacing

Type ramp

The Windows type ramp establishes crucial relationships between the type styles on a page, helping users read content easily. All sizes are in effective pixels and are optimized for Windows apps running on all screen sizes.

Windows 11 uses the following values for various types of text in the UI.

Expand table

Example	Weight	Size/line height
Caption	Small	12/16 epx
Body	Text	14/20 epx
Body Strong	Text semibold	14/20 epx
Body Large	Text	18/24 epx
Subtitle	Display semibold	20/28 epx
Title	Display semibold	28/36 epx
Title Large	Display semibold	40/52 epx

Example	Weight	Size/line height
Display	Display semibold	68/92 epx

Check out the guidance on using the XAML type ramp for more details.

Alignment

The default TextAlignment is Left, and in most instances, flush-left and ragged right provides consistent anchoring of the content and a uniform layout. For RTL languages, see Adjusting layout and fonts to support globalization.

Flush-left text is aligned along the left margin or gutter, also known as left-aligned or ragged right. Ensure even line endings or rag when setting flush-left text.

XAML

```
<TextBlock TextAlignment="Left">
```

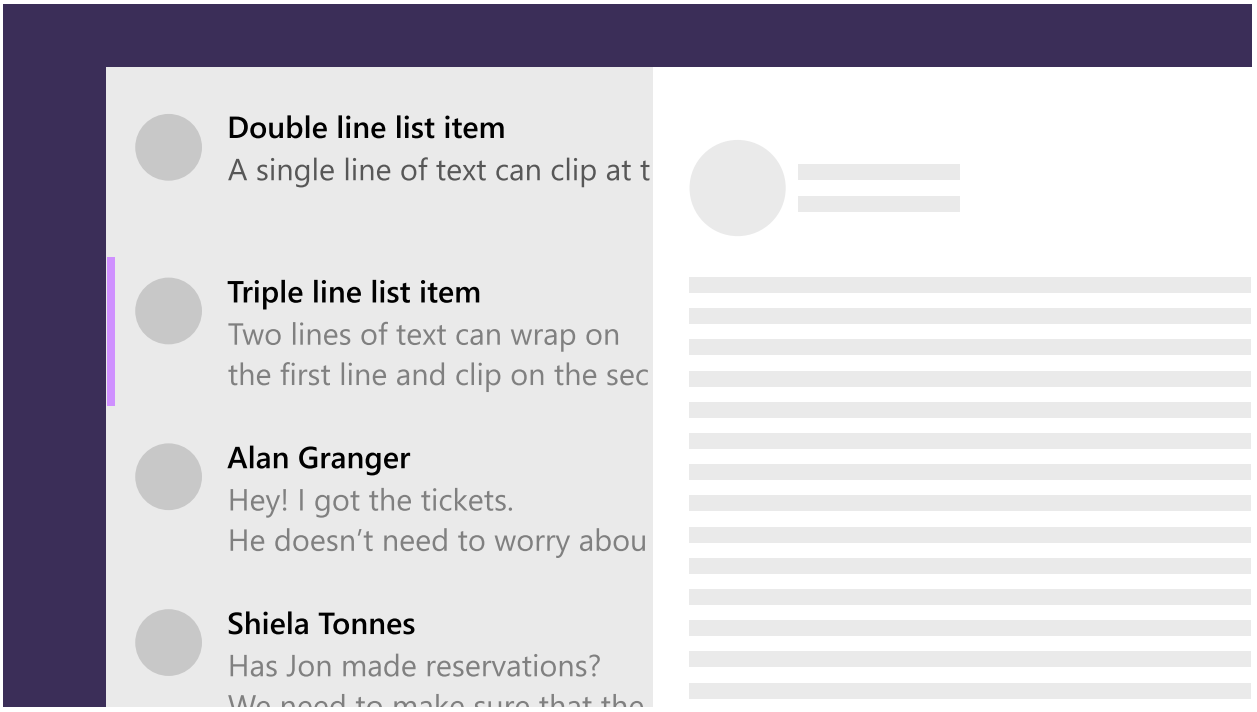
Character count

✓ Do Keep to 50–60 letters per line for ease of reading.

✗ Don't Don't use fewer than 20 characters or more than 60 characters per line as this is difficult to read.

Clipping and ellipses

When the amount of text extends beyond the space available, we recommend clipping the text and inserting ellipses [...], which is the default behavior of most UWP text controls.



XAML

```
<TextBlock TextWrapping="WrapWholeWords" TextTrimming="Clip"/>
```

✓ Do Clip text, and wrap if multiple lines are enabled.

✗ Don't Don't use ellipses to avoid visual clutter.

> **Note**
>
> If containers are not well-defined (for example, no differentiating background color), or when there is a link to see more text, then use ellipses.

Languages

Segoe UI Variable is our font for English, European languages, Greek, and Russian. For other languages, see the following recommendations.

Globalizing/localizing fonts

Use the LanguageFont font-mapping APIs for programmatic access to the recommended font family, size, weight, and style for a particular language. The LanguageFont object provides access to the correct font info for various categories of content including UI headers, notifications, body text, and user-editable document body fonts. For more info, see Adjusting layout and fonts to support globalization.

Fonts for non-Latin languages

Expand table

Font-family	Styles	Notes
Ebrima	Regular, Bold	User-interface font for African scripts (ADLaM, Ethiopic, N'Ko, Osmanya, Tifinagh, Vai).
Gadugi	Regular, Bold	User-interface font for North American scripts (Canadian Syllabics, Cherokee, Osage).
Leelawadee UI	Regular, Semilight, Bold	User-interface font for Southeast Asian scripts (Buginese, Khmer, Lao, Thai).
Malgun Gothic	Regular	User-interface font for Korean.
Microsoft JhengHei UI	Regular, Bold, Light	User-interface font for Traditional Chinese.
Microsoft YaHei UI	Regular, Bold, Light	User-interface font for Simplified Chinese.
Myanmar Text	Regular	Fallback font for Myanmar script.
Nirmala UI	Regular, Semilight, Bold	User-interface font for South Asian scripts (Bangla, Chakma, Devanagari, Gujarati, Gurmukhi, Kannada, Malayalam, Meetei Mayek, Odia, Ol Chiki, Sinhala, Sora Sompeng, Tamil, Telugu).
Segoe UI	Regular, Italic, Light Italic, Black Italic, Bold, Bold Italic, Light, Semilight, Semibold, Black	User-interface font for Arabic, Armenian, Georgian, and Hebrew.

Font-family	Styles	Notes
SimSun	Regular	A legacy Chinese UI font.
Yu Gothic UI	Light, Semilight, Regular, Semibold, Bold	User-interface font for Japanese.

Fonts

Sans-serif fonts

Sans-serif fonts are a great choice for headings and UI elements.

Expand table

Font-family	Styles	Notes
Arial	Regular, Italic, Bold, Bold Italic, Black	Supports European and Middle Eastern scripts (Latin, Greek, Cyrillic, Arabic, Armenian, and Hebrew). Black weight supports European scripts only.
Calibri	Regular, Italic, Bold, Bold Italic, Light, Light Italic	Supports European and Middle Eastern scripts (Latin, Greek, Cyrillic, Arabic and Hebrew). Arabic available in the uprights only.
Consolas	Regular, Italic, Bold, Bold Italic	Fixed width font that supports European scripts (Latin, Greek and Cyrillic).
Segoe UI	Regular, Italic, Light Italic, Black Italic, Bold, Bold Italic, Light, Semilight, Semibold, Black	User-interface font for European and Middle East scripts (Arabic, Armenian, Cyrillic, Georgian, Greek, Hebrew, Latin), and also Lisu script.
Selawik	Regular, Semilight, Light, Bold, Semibold	An open-source font that's metrically compatible with Segoe UI, intended for apps on other platforms that don't want to bundle Segoe UI. Get Selawik on GitHub.

Serif fonts

Serif fonts are good for presenting large amounts of text.

Expand table

Font-family	Styles	Notes
Cambria	Regular	Serif font that supports European scripts (Latin, Greek, Cyrillic).
Courier New	Regular, Italic, Bold, Bold Italic	Serif fixed width font that supports European and Middle Eastern scripts (Latin, Greek, Cyrillic, Arabic, Armenian, and Hebrew).
Georgia	Regular, Italic, Bold, Bold Italic	Supports European scripts (Latin, Greek and Cyrillic).
Times New Roman	Regular, Italic, Bold, Bold Italic	Legacy font that supports European scripts (Latin, Greek, Cyrillic, Arabic, Armenian, Hebrew).

Variable fonts

Variable fonts are good for precisely controlling the appearance of text.

Expand table

Font-family	Axes	Notes
Bahnschrift	Weight, Width	Variable font that supports Latin, Greek, and Cyrillic.
Segoe UI Variable	Weight, Optical Size	Variable font that supports Latin, Greek, and Cyrillic.

Symbols and icons

Expand table

Font-family	Styles	Notes
Segoe MDL2 Assets	Regular	User-interface font for app icons. For more info, see the Segoe Fluent Icons font article.
Segoe UI Emoji	Regular	User-interface font for Emoji.
Segoe UI Symbol	Regular	Fallback font for symbols.

Related articles

- Text controls
- XAML theme resources

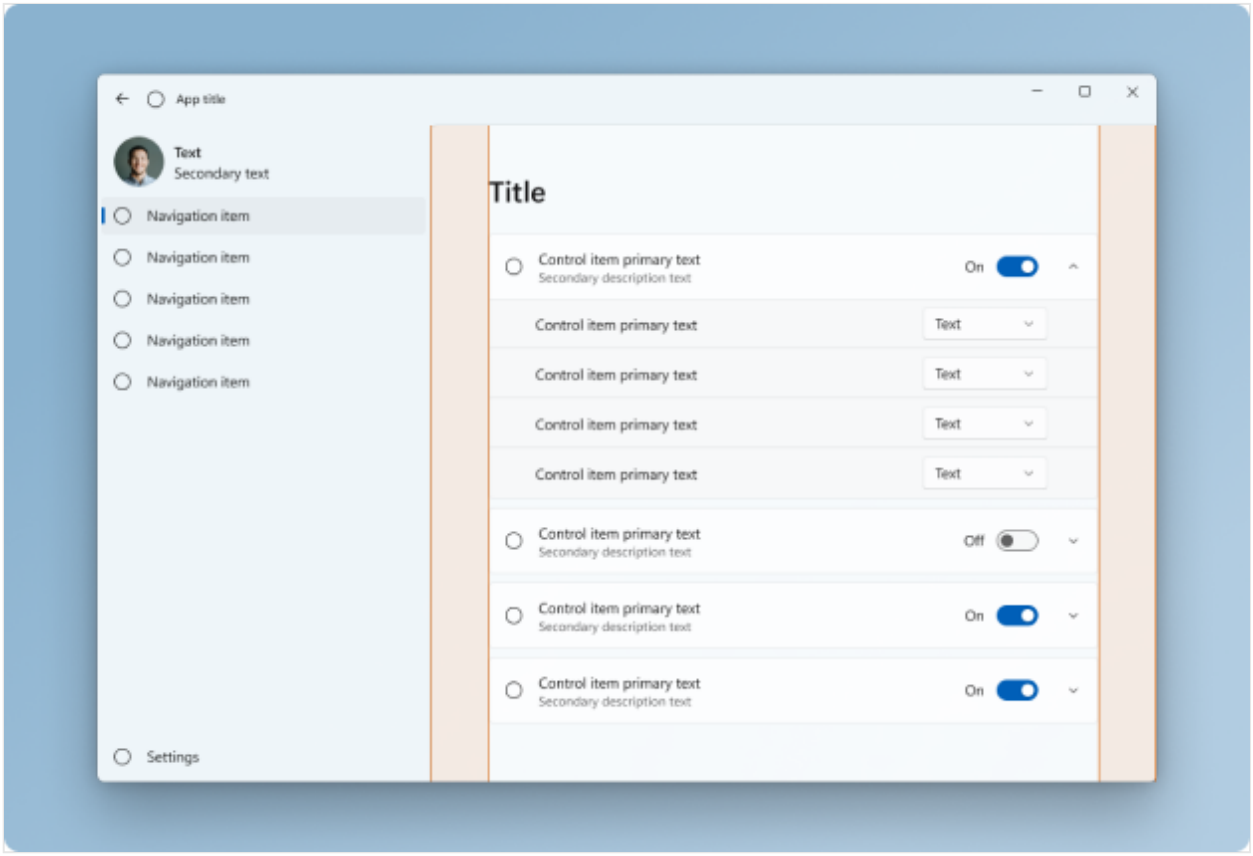
- XAML styles
- Microsoft Typography
- Variable Fonts

Design basics for Windows apps

Article • 10/04/2021

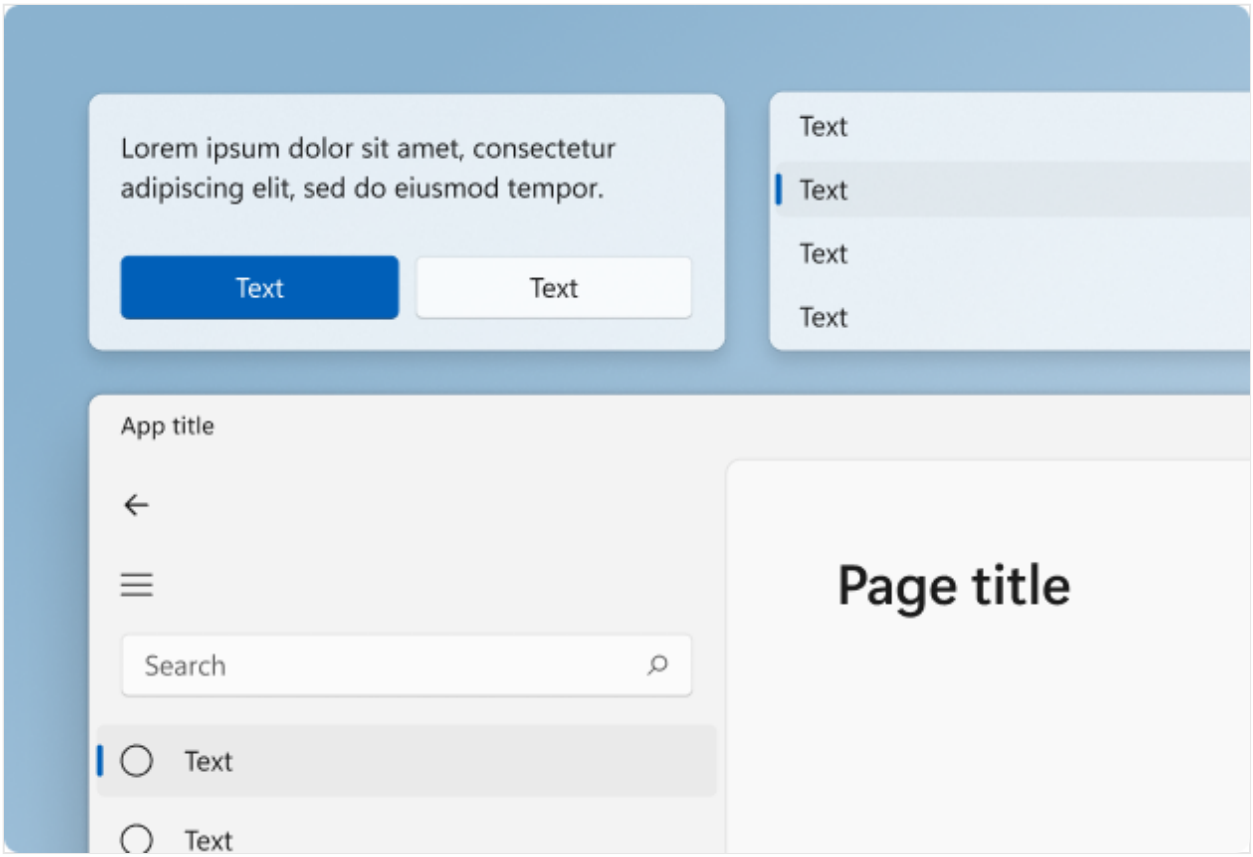
Windows design guidance is a resource to help you design and build beautiful, polished apps. It's not a list of prescriptive rules - it's a living document, designed to adapt to our evolving design language as well as the needs of our app-building community.

Overview



App Silhouette

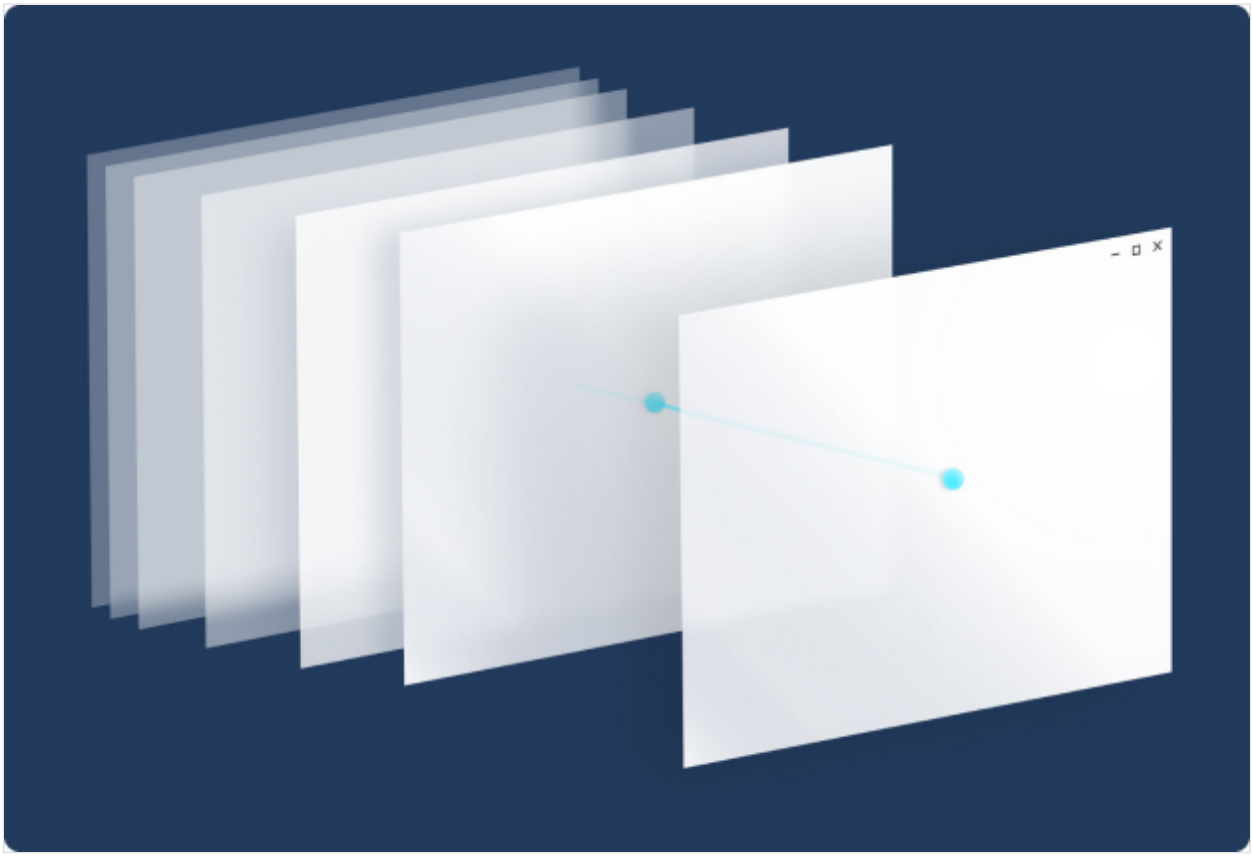
A high-level look at app patterns commonly used across our in-box applications.



Windows 11 Signature Experiences

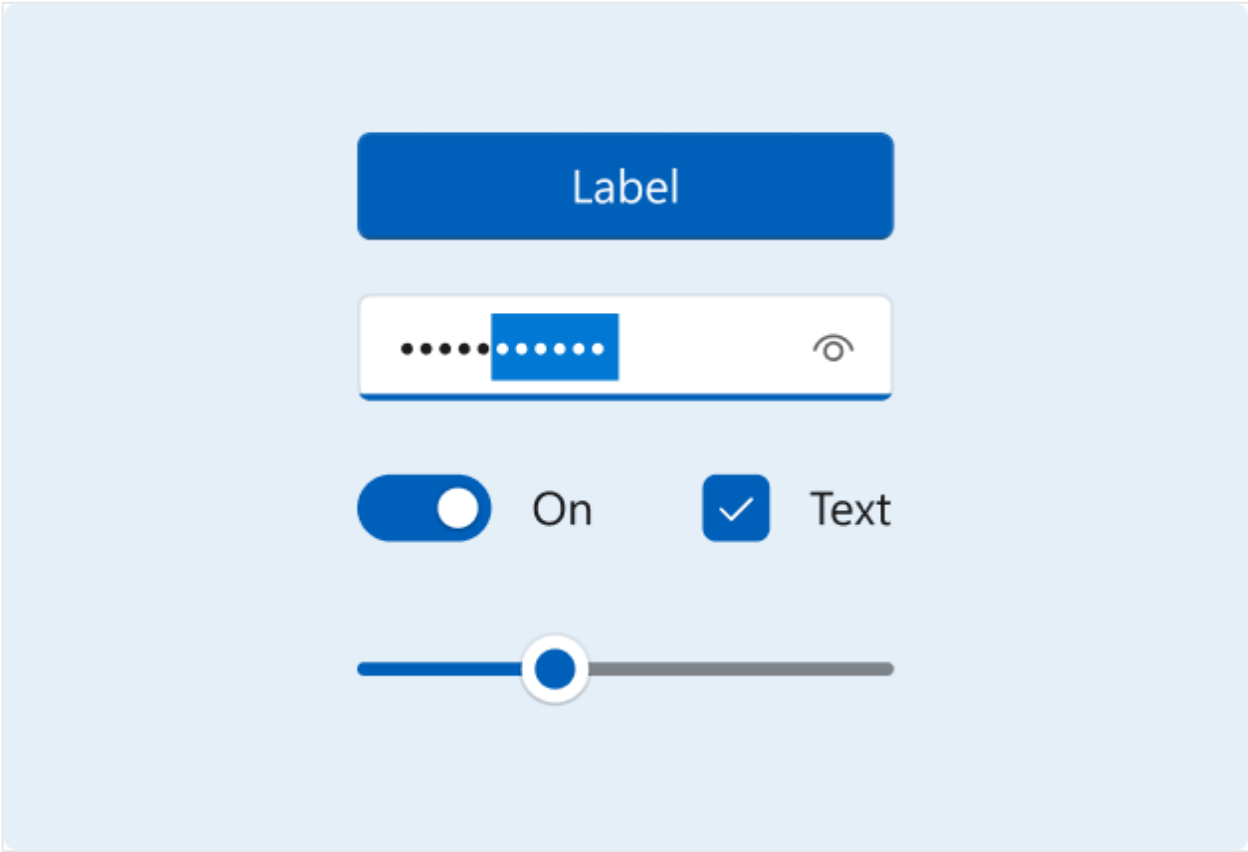
See what's new for Windows 11 design elements.

Basics



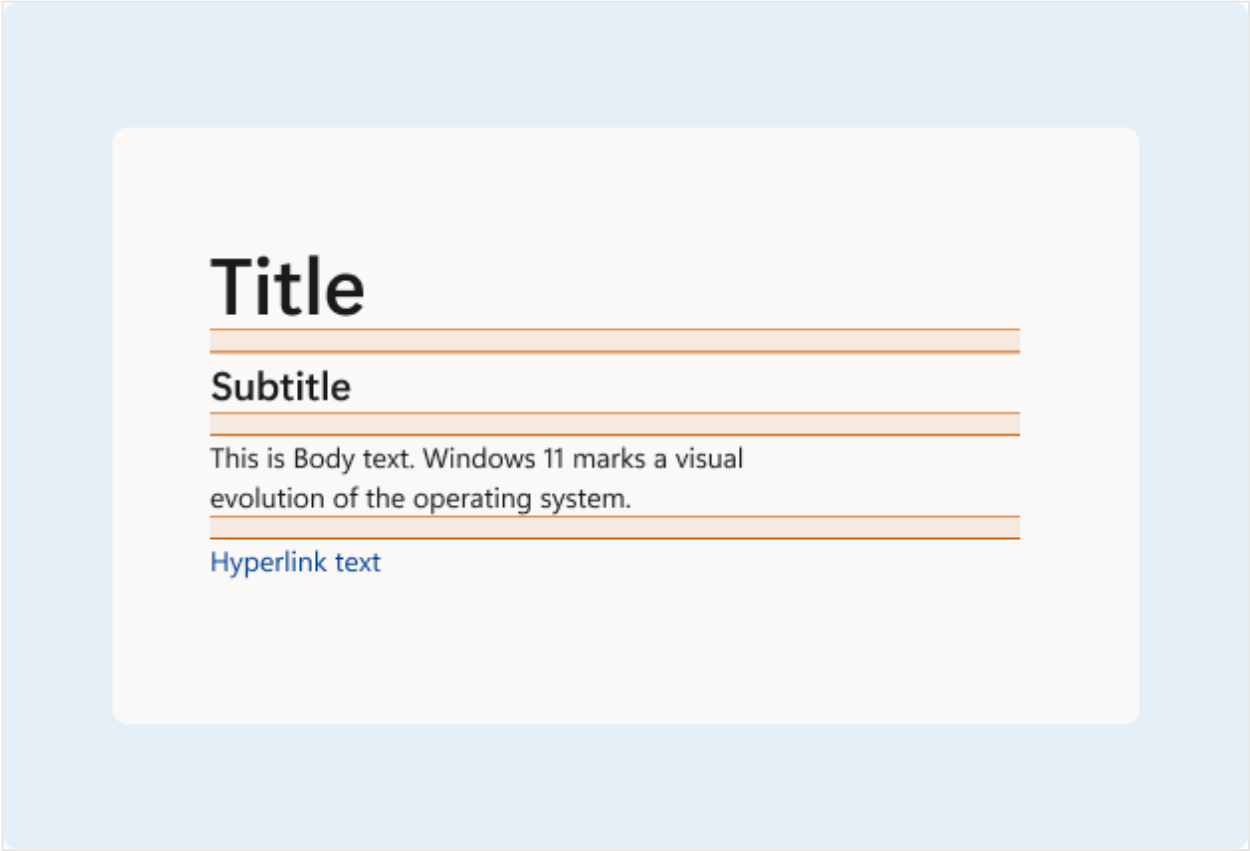
Navigation basics

Navigation in Windows apps is based on a flexible model of navigation structures, navigation elements, and system-level features. This article introduces you to these components and shows you how to use them together to create a good navigation experience.



Command basics

Command elements are the interactive UI elements that enable the user to perform actions, such as sending an email, deleting an item, or submitting a form. This article describes the command elements, such as buttons and check boxes, the interactions they support, and the command surfaces (such as command bars and context menus) for hosting them.



Content basics

This article provides some practical tips and examples to help you design the content of your app: Windows spacing rationale, using the type ramp to demonstrate hierarchy, lists and grids, and how to group controls.

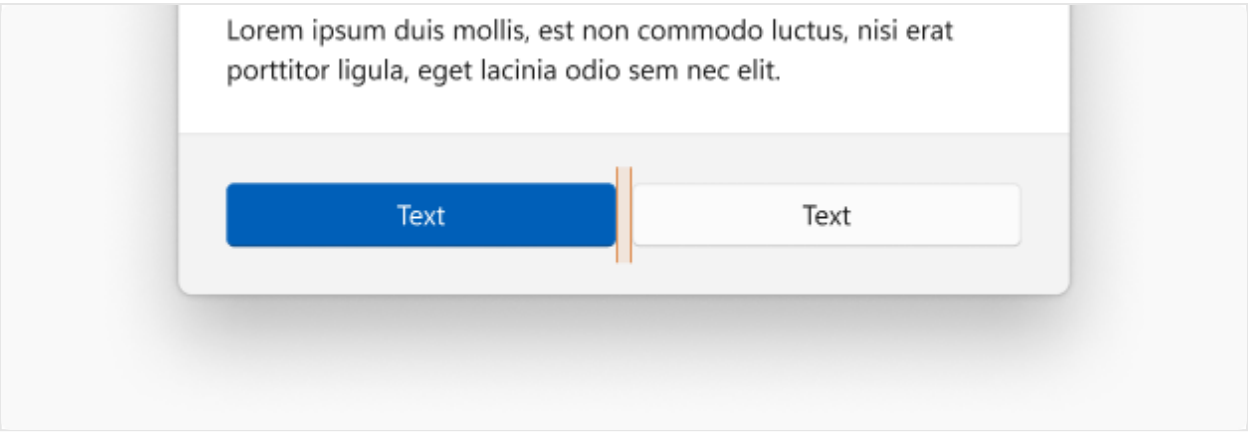
Content design basics for Windows apps

Article • 10/04/2021

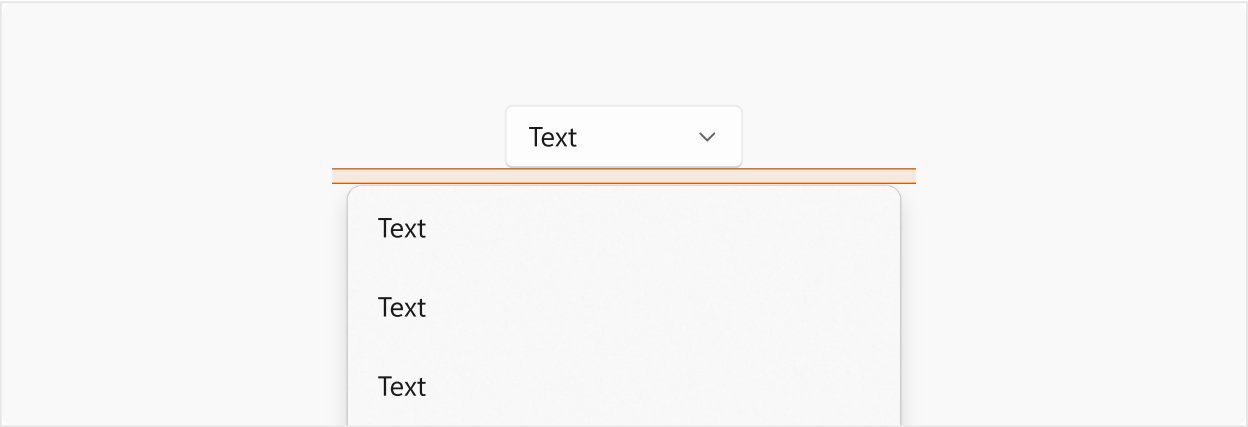
This article provides some practical tips and examples to help you design the content of your app: Windows spacing rationale, using the type ramp to demonstrate hierarchy, lists and grids, and how to group controls.

Spacing and gutters

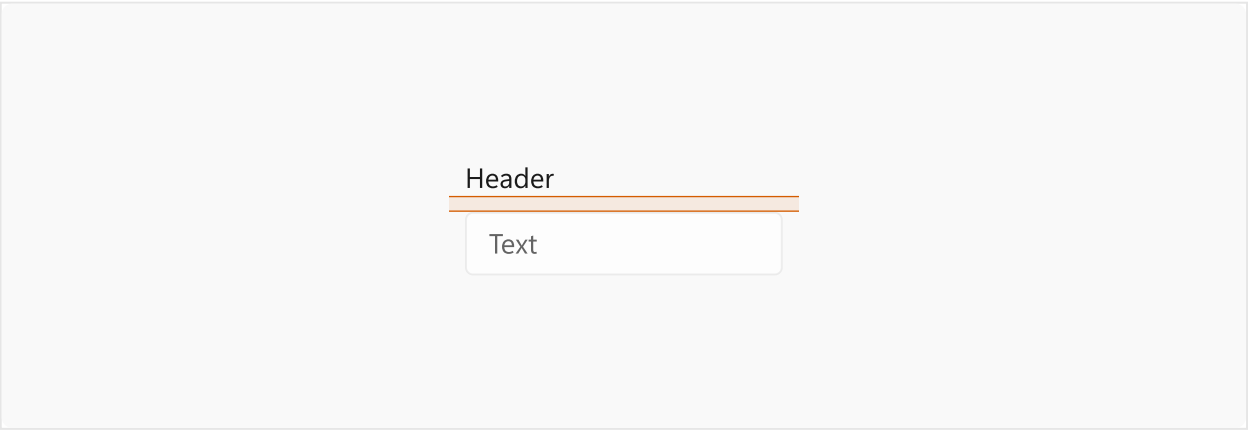
The use of consistently sized spacing and gutters semantically groups an experience into separate components. These values map to our rounded corner logic and together help create a cohesive and usable layout.



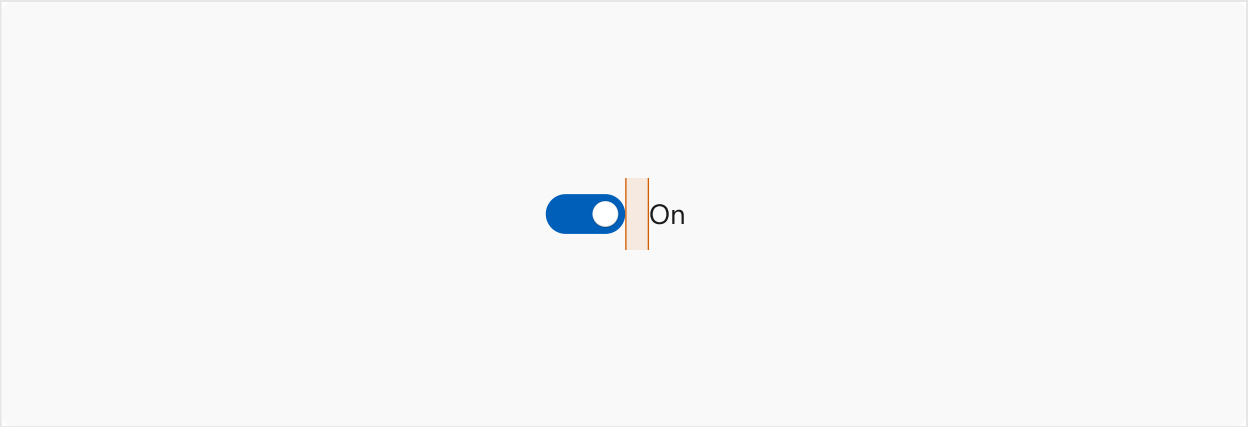
8epx between buttons



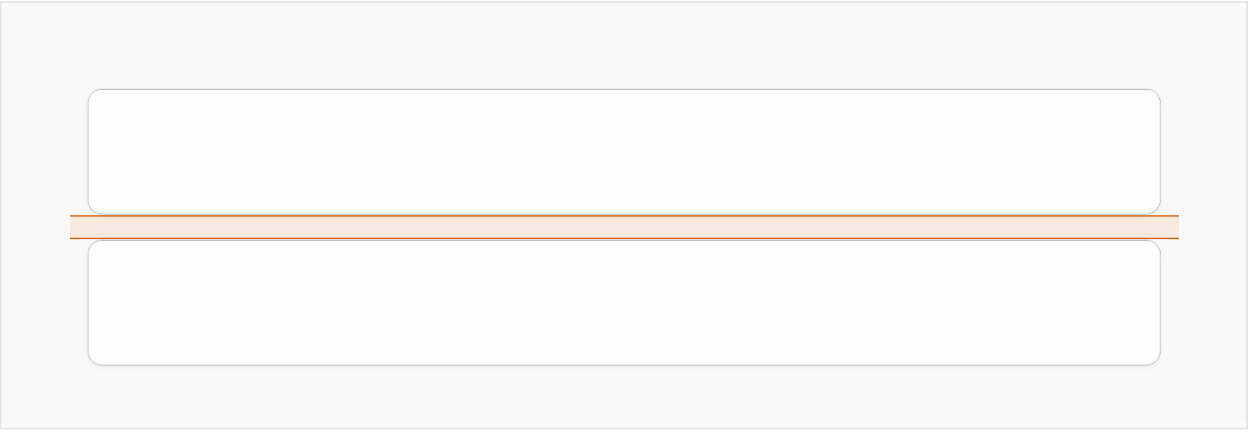
8epx between buttons and flyouts



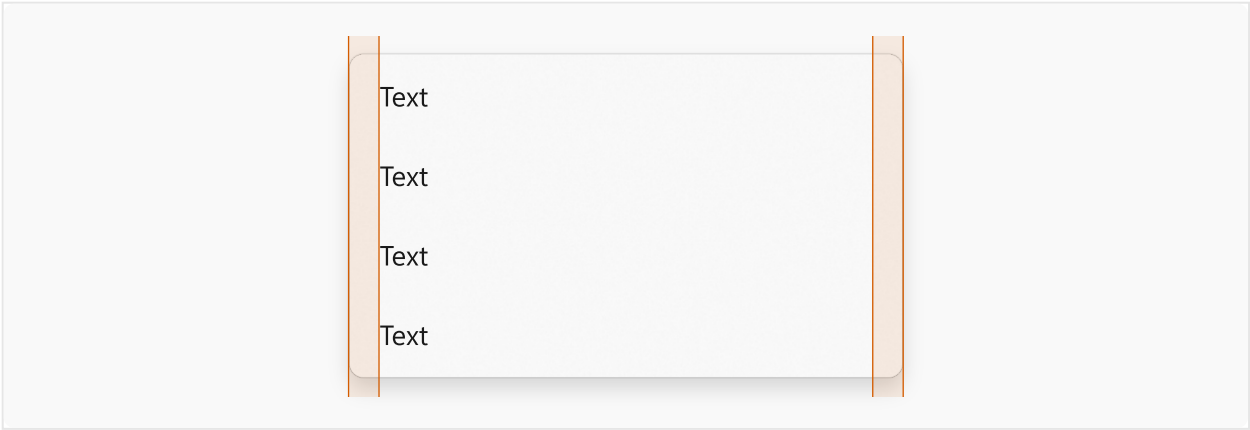
8epx between control and header



12epx between control and label



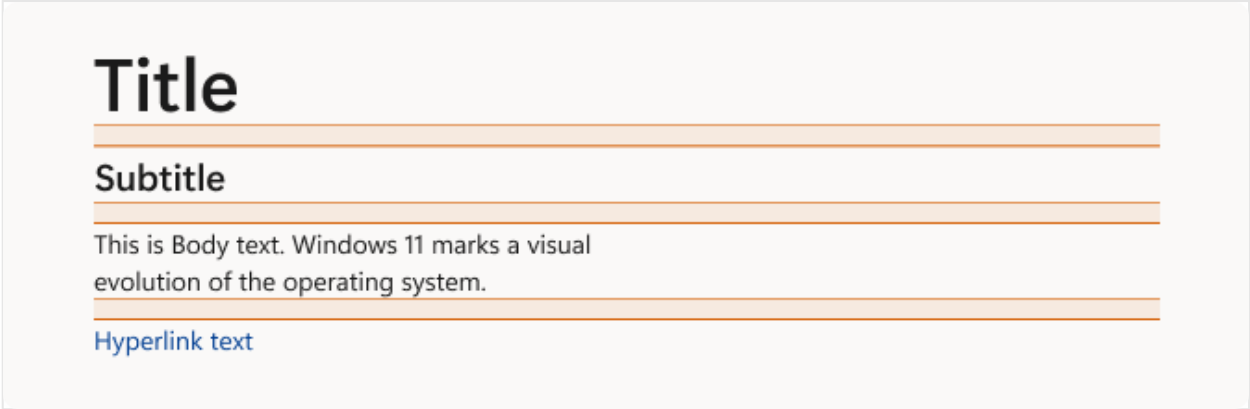
12epx between content areas



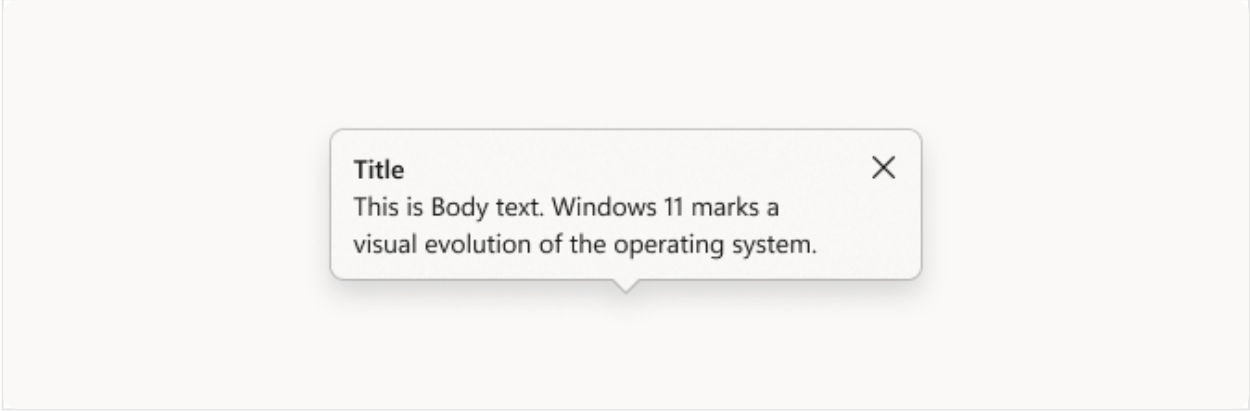
16epx between surface and edge text

Text + hierarchy

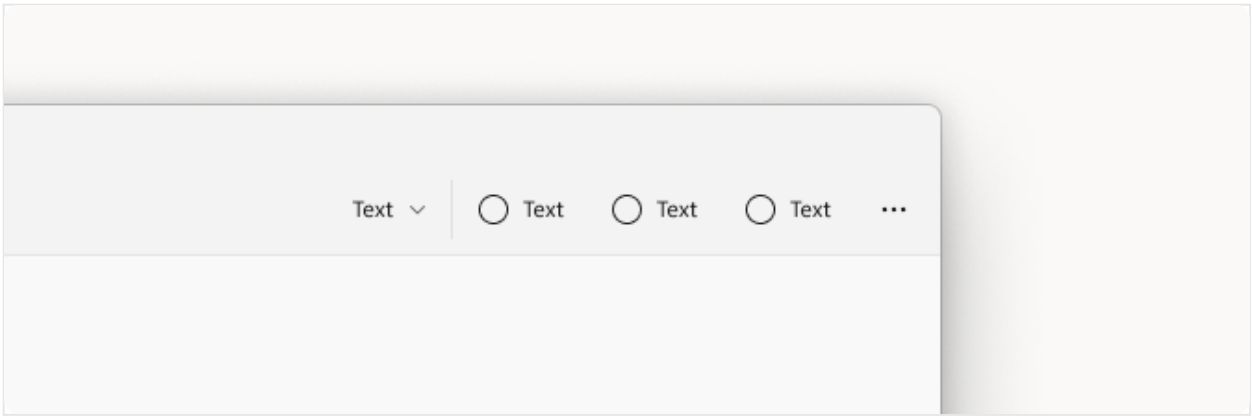
Our type ramp (link) is designed to provide an array of sizes that can help communicate hierarchy within an app.



Using Title, Subtitle and Body with 12epx spacing.



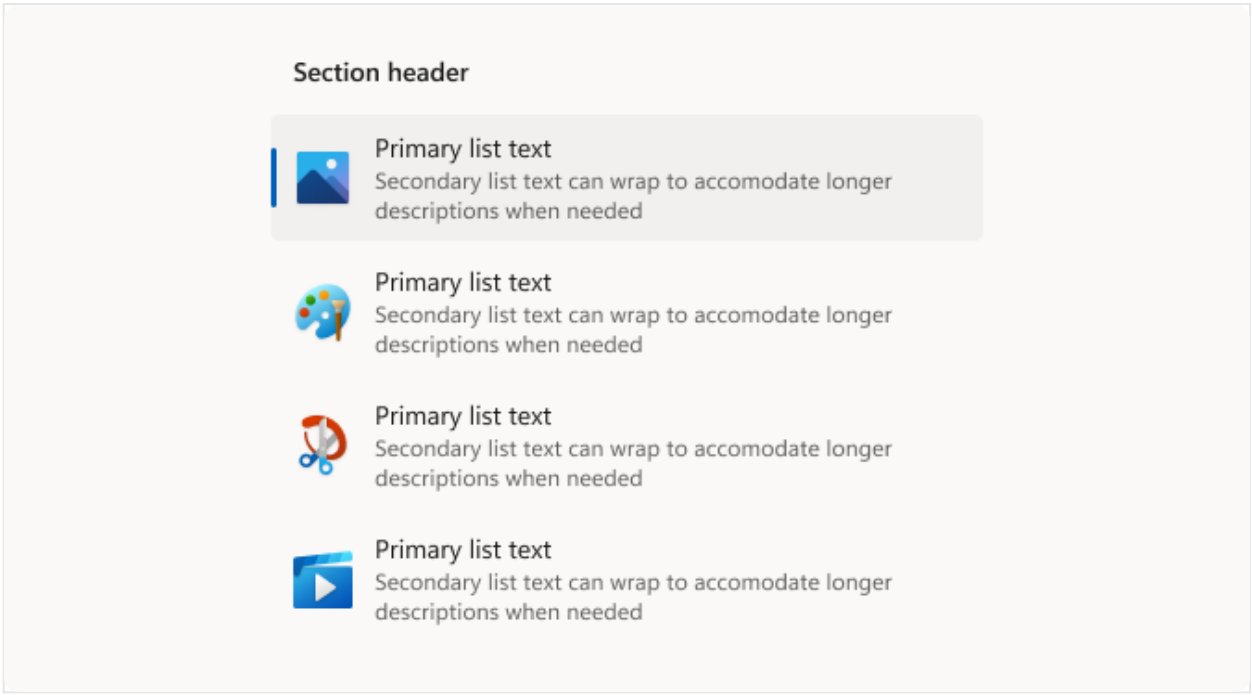
When differentiating a title in a confined UI space, use Body Strong for the title without any additional spacing between text blocks.



Use caption size for very confined spaces where text is needed, such as command buttons.

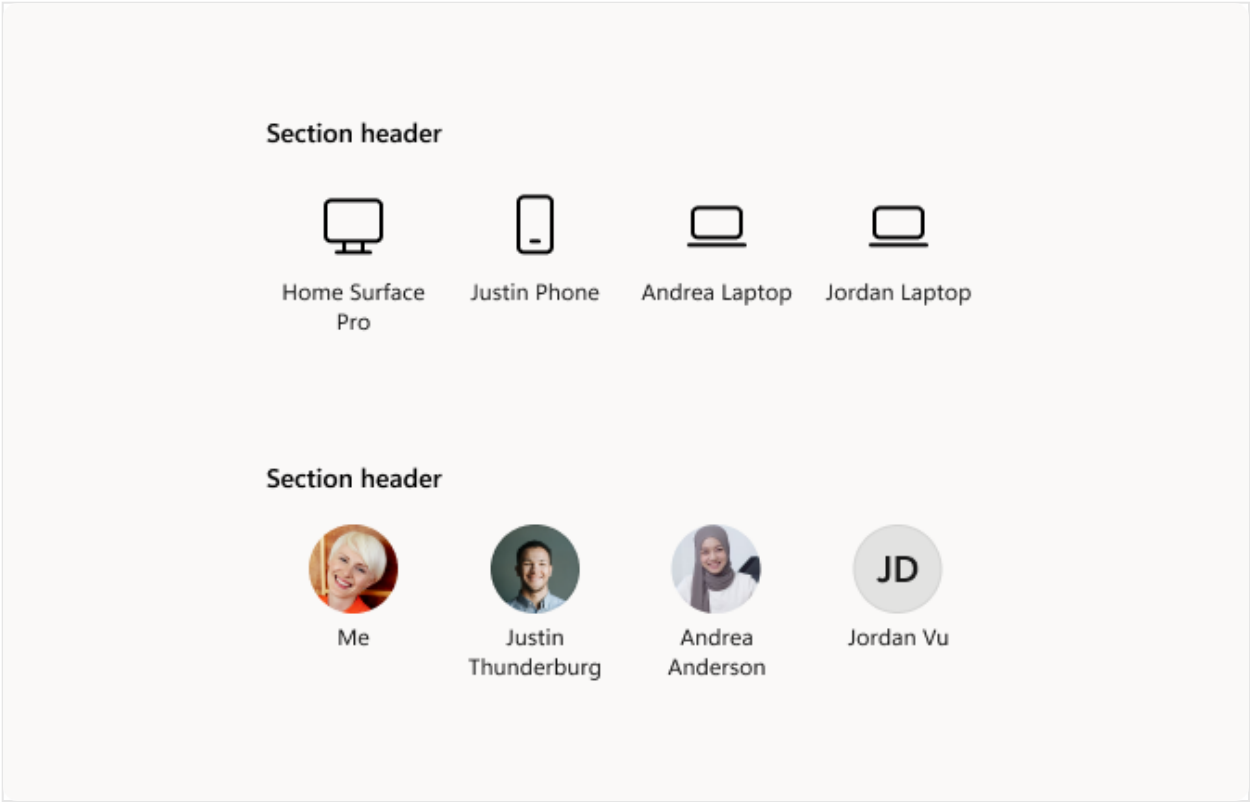
Lists and grids

There are a variety of list and grid styles that can be created. Below are a variety of compositions used in Windows.

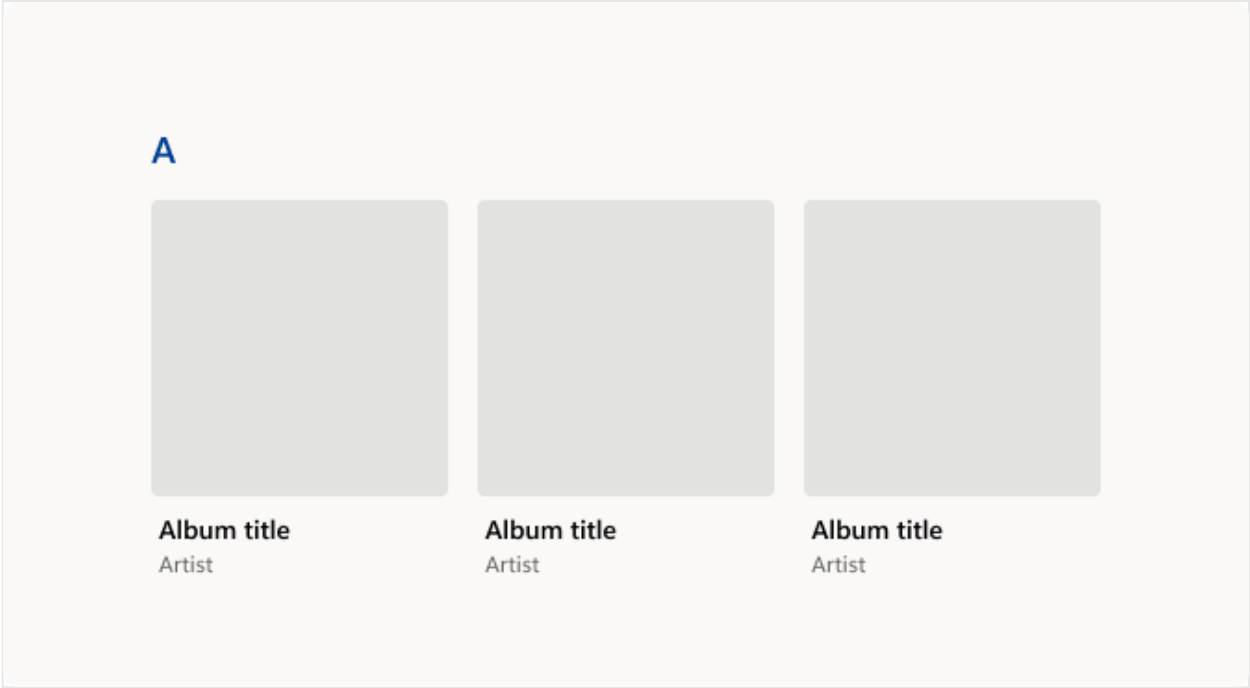


For multi-line lists, use Body and Caption from the type ramp, and 32epx icons.

Use Body Strong for section headers.



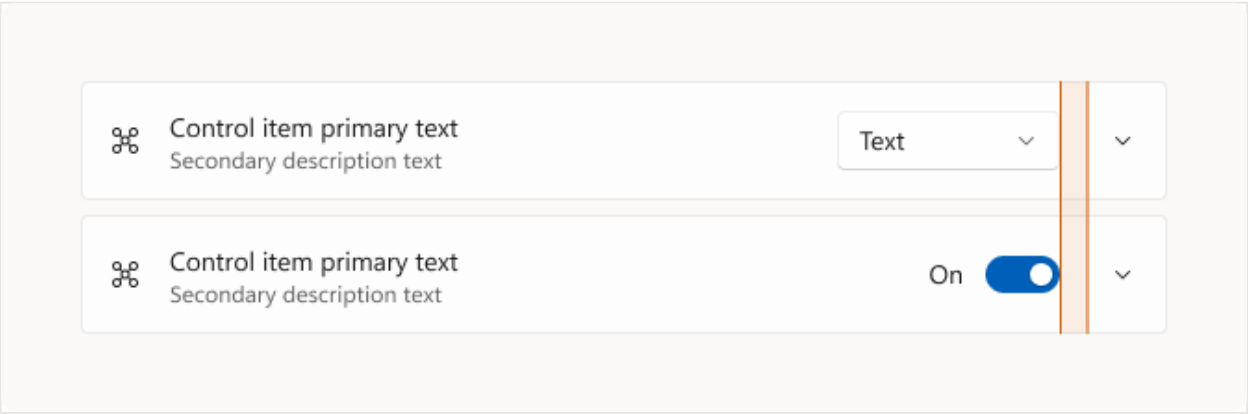
When using icons or person picture elements for grid items, use Caption text that is center-aligned.



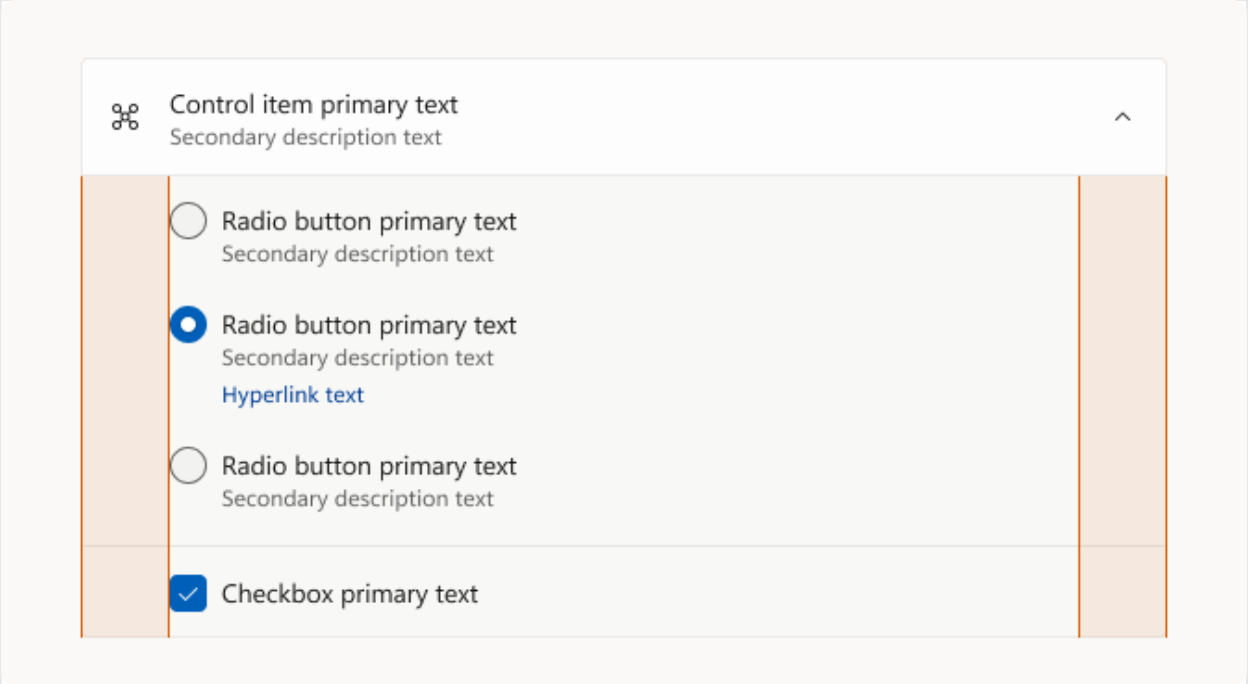
Use Body style for primary text and left-align to the image if your list contains large graphical elements with text.

Using controls

Some examples of how controls can relate to each other in common configurations.



Examples showing how to use an expander control (link) with list styles and common controls. Controls should be right-aligned with 16epx between the control and expander button.



This example shows controls alignment when placed inside the expander. Indent the controls 48epx.

Writing style

Article • 06/24/2021

The way you phrase an error message, the way you write help documentation, and even the text you choose for a button have big impacts on the usability of your app. Writing style can make a big difference between an awful user experience and a better one.

Voice and tone principles

Research shows that people respond best to a writing style that's friendly, helpful, and concise. As a part of this research, Microsoft developed three voice and tone principles that we apply across all our content, and are an integral part of Fluent design.

Be warm and relaxed

Above all else, you don't want to scare off the user. Be informal, be casual, and don't use terms they won't understand. Even when things break, don't blame the user for any problems. Your app should take responsibility instead, and offer welcoming guidance that puts the user's actions first.

Be ready to lend a hand

Always show empathy in your writing. Be focused on explaining what's going on and providing the information that the user needs, without overloading them with unnecessary info. And if possible, always provide a solution when there's a problem.

Be crisp and clear

Most of the time, text isn't the focus of an app. It's there to guide people, to teach them what's going on and what they should do next. Don't lose sight of this when writing the text in your app, and don't assume that users will read every word. Use language that is familiar to your audience, make sure it's easy to understand at a glance.

Lead with what's important

Users need to be able to read and understand your text at a glance. Don't pad your words with unnecessary introductions. Give your key points the most visibility, and always present the core of an idea before you add onto it.

✓ Do Select **filters** to add effects to your image.

✕ Don't If you want to add visual effects or alterations to your image, select **filters.**

Emphasize action

Apps are defined by actions. Users take action when they use the app, and the app takes action when it responds to the user. Make sure your text uses the *active voice* throughout your app. People and functions should be described as doing things, instead of having things done to them.

✓ Do Restart the app to see your changes.

✕ Don't The changes will be applied when the app is restarted.

Short and sweet

Users scan text, and will often skip over larger blocks of words entirely. Don't sacrifice necessary information and presentation, but don't use more words than you have to.

Sometimes, this will mean relying on many shorter sentences or fragments. Other times, this will mean being extra choosy about the words and structure of longer sentences.

✔ **Do** We couldn't upload the picture. If this happens again, try restarting the app. But don't worry — your picture will be waiting when you come back.

✘ **Don't** An error occurred, and we weren't able to upload the picture. Please try again, and if you encounter this problem again, you may need to restart the app. But don't worry — we've saved your work locally, and it'll be waiting for you when you come back.

Style conventions

If you don't consider yourself to be a writer, it can be intimidating to try to implement these principles and recommendations. But don't worry — using simple and straightforward language is a great way to provide a good user experience. And if you're still unsure how to structure your words, here are some helpful guidelines. And if you want more information, check out the Microsoft Style Guide.

Addressing the user

Speak directly to the user.

- Always address the user as "you."
- Use "we" to refer to your own perspective. It's welcoming and helps the user feel like part of the experience.
- Don't use "I" or "me" to refer to the app's perspective, even if you're the only one creating it.

✔ **Do** We couldn't save your file to that location.

Abbreviations

Abbreviations can be useful when you need to refer to products, places, or technical concepts multiple times throughout your app. They can save space and feel more natural, as long as the user understands them.

- Don't assume that users are already familiar with any abbreviations, even if you think they're common.
- Always define what a new abbreviation means the first time the user will see it.
- Don't use abbreviations that are too similar to one another.
- Don't use abbreviations if you're localizing your app, or if your users speak English as a second language.

✓ Do

The Windows app design guidance is a resource to help you design and build beautiful, polished apps. With the design features that are included in every Windows app, you can build user interfaces (UI) that scale across a range of devices.

Contractions

People are used to contractions, and expect to see them. Avoiding them can make your app seem too formal or even stilted.

- Use contractions when they're a natural fit for the text.
- Don't use unnatural contractions just to save space, or when they would make your words sound less conversational.

✓ Do

When you're happy with your image, select **save** to add it to your gallery. From there, you'll be able to share it with friends.

Periods

Ending text with a period implies that that text is a full sentence. Use a period for larger blocks of text, and avoid them for text that's shorter than a complete sentence.

- Use periods to end full sentences in tooltips, error messages, and dialogs.
- Don't end text for buttons, radio buttons, labels, or checkboxes with a period.

✓ Do

You're not connected.

- Check that your network cables are plugged in.
- Make sure you're not in airplane mode.
- See if your wireless switch is turned on.

- Restart your router.

Capitalization

While capital letters are important, they're easy to overuse.

- Capitalize proper nouns.
- Capitalize the start of every string of text in your app: the start of every sentence, label, and title.

✓ Do **Which part is giving you trouble?**

- I forgot your password.
- It won't accept password.
- Someone else might be using my account.

Error messages

When something goes wrong in your app, users pay attention. Because users might be confused or frustrated when they encounter an error message, they're an area where good voice and tone can have a particularly significant impact.

More than anything else, it's important that your error message doesn't blame the user. But it's also important not to overwhelm them with information that they don't understand. Most of the time a user who encounters an error just wants to get back to what they were doing as quickly and as easily as they can. Therefore, any error message you write should:

- **Be warm and relaxed** by using a conversational tone and avoiding unfamiliar terms and technical jargon.
- **Be ready to lend a hand** by telling the user what went wrong to the best of your ability, by telling them what will happen next, and by providing a realistic solution they can accomplish.
- **Be crisp and clear** by eliminating extraneous information.

✓ Do **You're not connected.**

- Check that your network cables are plugged in.

- Make sure you're not in airplane mode.
- See if your wireless switch is turned on.
- Restart your router.

Dialogs

Many of the same advice for writing error messages also applies when creating the text for any dialogs in your app. While dialogs are expected by the user, they still interrupt the normal flow of the app, and need to be helpful and concise so the user can get back to what they were doing.

But most important is the "call and response" between the title of a dialog and its buttons. Make sure that your buttons are clear answers to the question posed by the title, and that their format is consistent across your app.

✓ Do **Which part is giving you trouble?**

1. I forgot my password
2. It won't accept my password
3. Someone else might be using my account

Buttons

Text on buttons needs to be concise enough that users can read it all at a glance and clear enough that the button's function is immediately obvious. The longest the text on a button should ever be is a couple short words, and many should be shorter than that. When writing the text for buttons, remember that every button represents an action. Be sure to use the *active voice* in button text, to use words that represent actions rather than reactions.

✓ Do

- Install now
- Share

Spoken experiences

The same general principles and recommendations apply when writing text for spoken experiences, such as Cortana. In those features, the principles of good writing are even

more important, because you are unable to provide users with other visual design elements to supplement the spoken words.

- **Be warm and relaxed** by engaging your users with a conversational tone. More than in any other area, it's vital that a spoken experience sound warm and approachable, and be something that users aren't afraid to talk to.
- **Be ready to lend a hand** by providing alternative suggestions when the user asks the impossible. Much like in an error message, if something went wrong and your app isn't able to fulfill the request, it should give the user a realistic alternative that they can try asking, instead.
- **Be crisp and clear** by keeping your language simple. Spoken experiences aren't suitable for long sentences or complicated words.

Accessibility and localization

Your app can reach a wider audience if it's written with accessibility and localization in mind. This is something that can't only be accomplished through text, though straightforward and friendly language is a great start. For more information, see our accessibility overview and localization guidelines.

- **Be ready to lend a hand** by taking different experiences into account. Avoid phrases that might not make sense to an international audience, and don't use words that make assumptions about what the user can and can't do.
- **Be crisp and clear** by avoiding unusual and specialized words when they aren't necessary. The more straightforward your text is, the easier it is to localize.

Techniques for non-writers

You don't need to be a trained or experienced writer to provide your users with a good experience. Pick words that sound comfortable to you — they'll feel comfortable to others, too. But sometimes, that's not as easy as it sounds. If you get stuck, these techniques can help you out.

- Imagine that you're talking to a friend about your app. How would you explain the app to them? How would you talk about its features or give them instructions? Better yet, explain the app to an actual person who hasn't used it yet.
- Imagine how you would describe a completely different app. For instance, if you're making a game, think of what you'd say or write to describe a financial or a news

app. The contrast in the language and structure you use can give you more insight into the right words for what you're actually writing about.

- Take a look at similar apps for inspiration.

Finding the right words is a problem that many people struggle with, so don't feel bad if it's not easy to settle on something that feels natural.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Commanding basics

Article • 08/01/2022

In a Windows app, *command elements* are interactive UI elements that let users perform actions such as sending an email, deleting an item, or submitting a form. *Command interfaces* are composed of common command elements, the command surfaces that host them, the interactions they support, and the experiences they provide.

Provide the best command experience

The most important aspect of a command interface is what you're trying to let a user accomplish. As you plan the functionality of your app, consider the steps required to accomplish those tasks and the user experiences you want to enable. Once you've completed an initial draft of these experiences, then you can make decisions on the tools and interactions to implement them.

Here are some common command experiences:

- Sending or submitting information
- Selecting settings and choices
- Searching and filtering content
- Opening, saving, and deleting files
- Editing or creating content

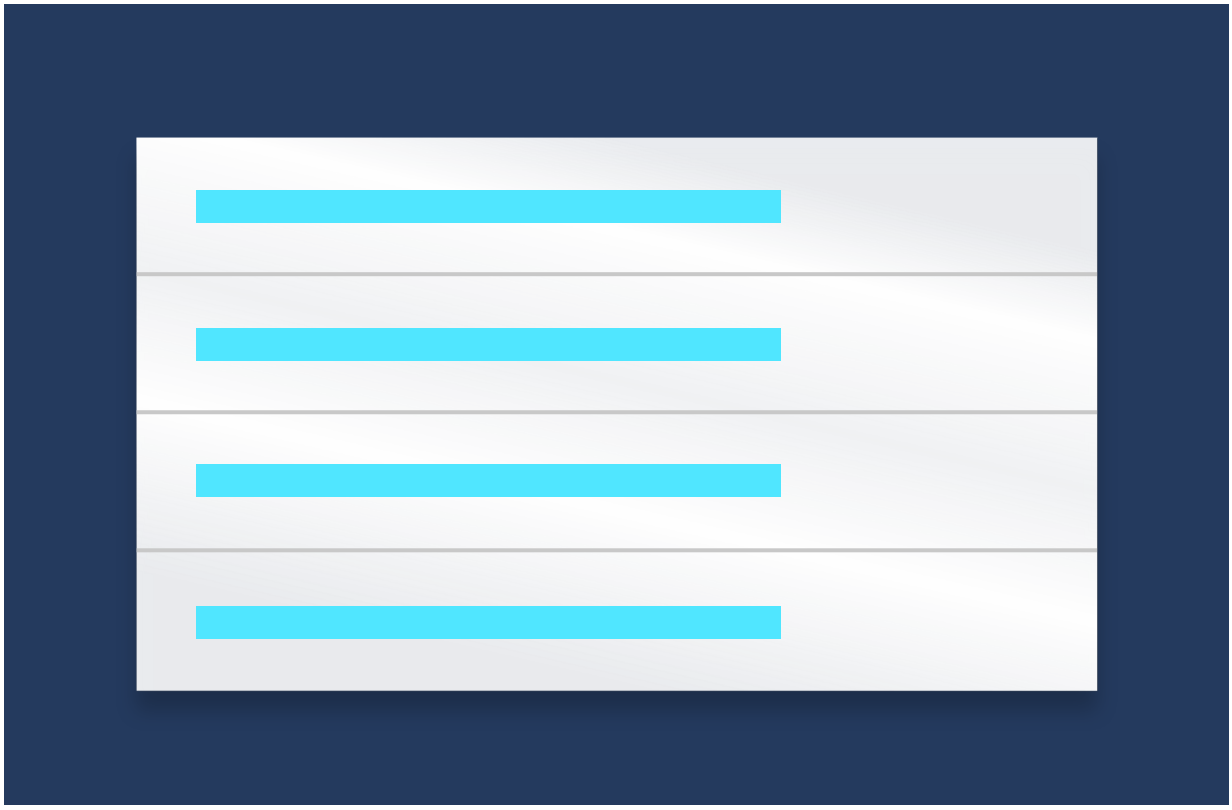
Be creative with the design of your command experiences. Choose which input devices your app supports, and how your app responds to each device. By supporting the broadest range of capabilities and preferences you make your app as usable, portable, and accessible as possible (see Commanding design for Windows apps for more detail).

Choose the right command elements

Using the right elements in a command interface can make the difference between an intuitive, easy-to-use app and a difficult, confusing app. A comprehensive set of command elements are available in the Windows app. Here's a list of some of the most common UWP command elements.

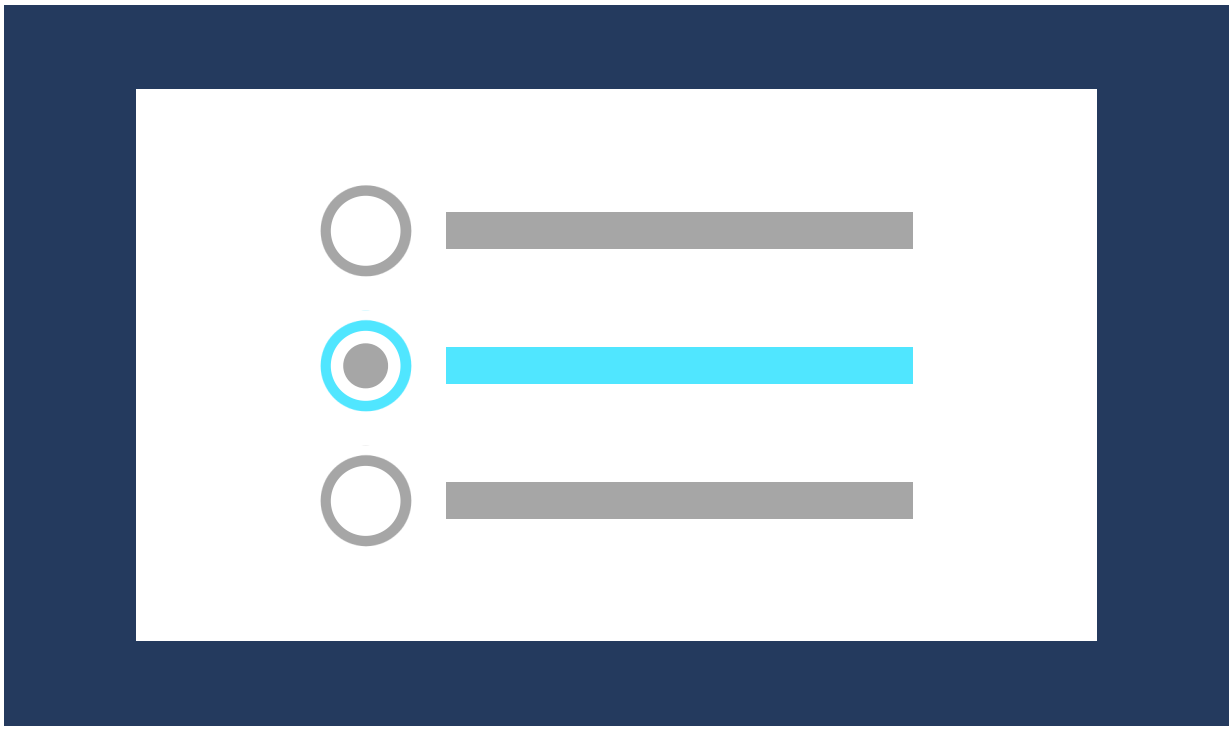
Buttons

Buttons trigger an immediate action. Examples include sending an email, submitting form data, or confirming an action in a dialog.



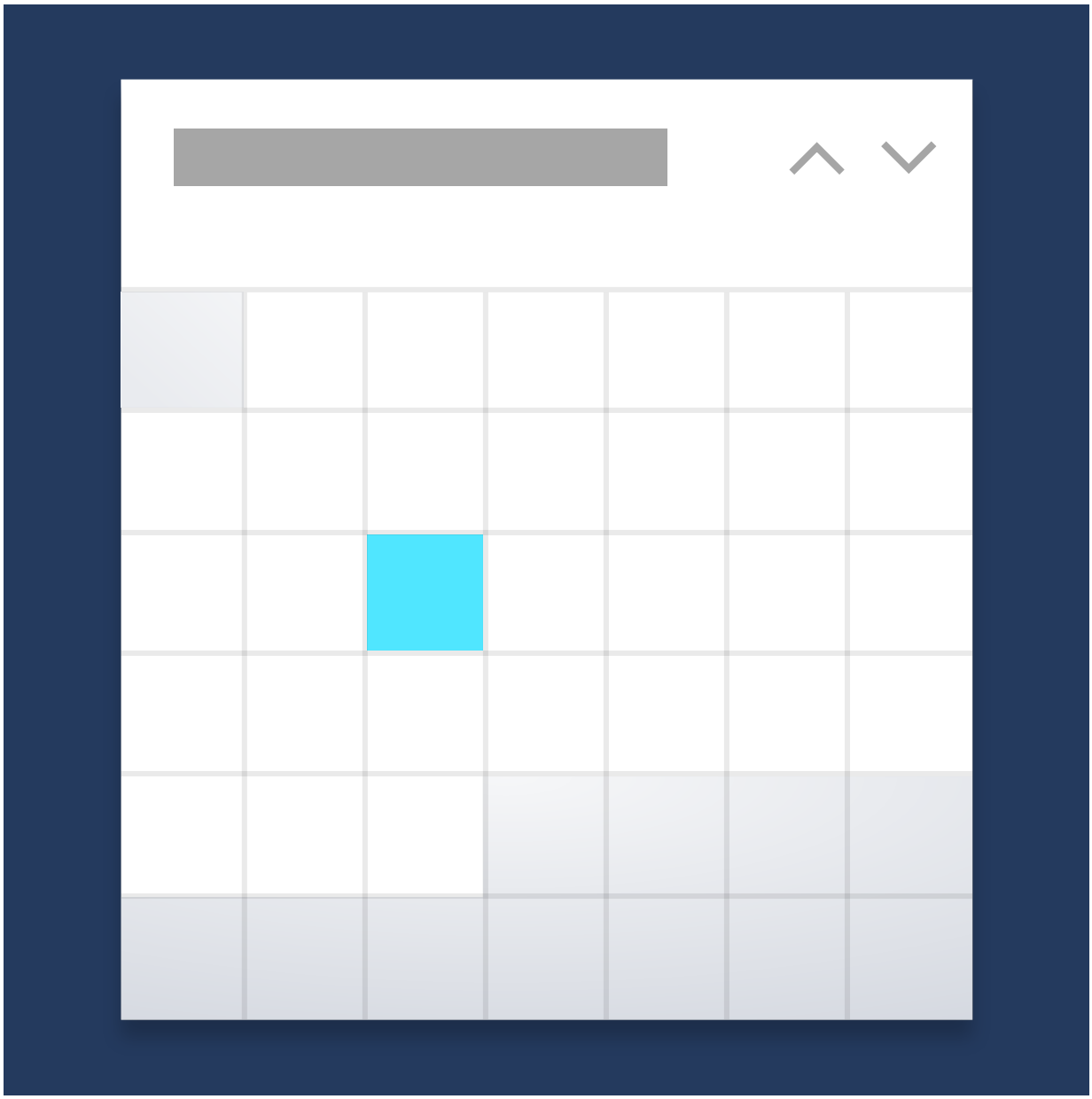
Lists

Lists present items in an interactive list or a grid. Usually used for many options or display items. Examples include drop-down list, list box, list view and grid view.



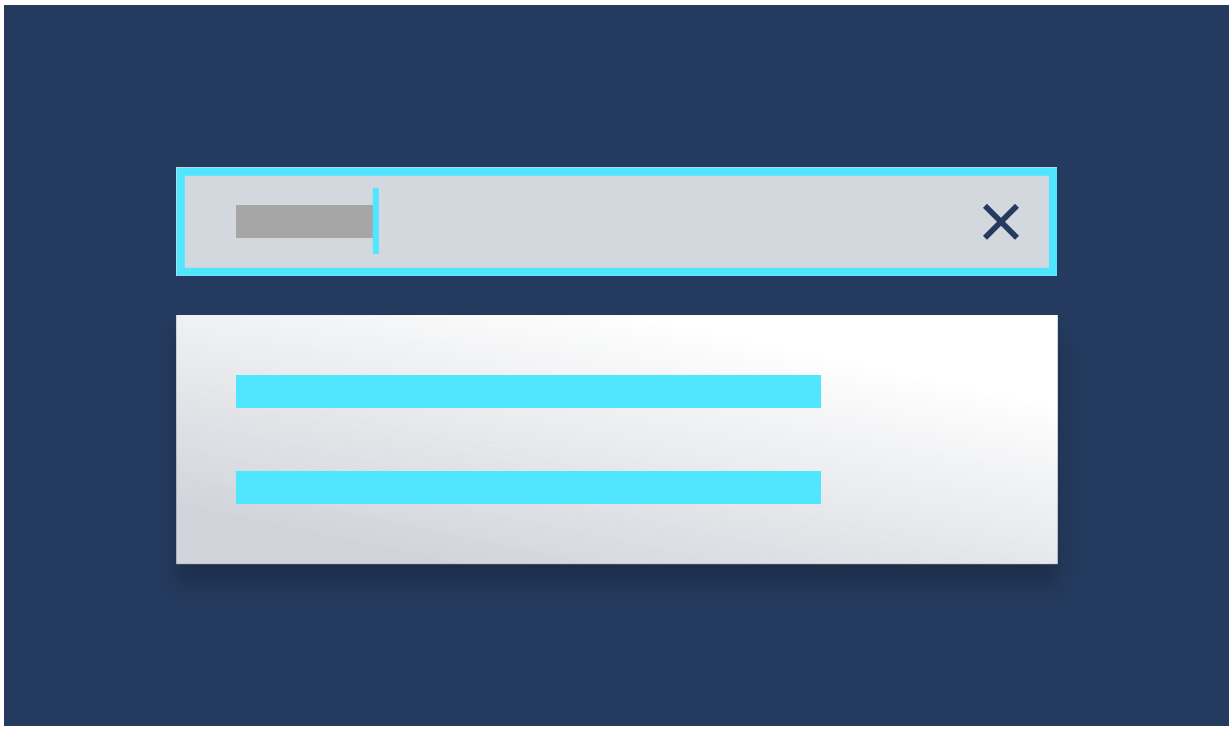
Selection controls

Lets users choose from a few options, such as when completing a survey or configuring app settings. Examples include CheckBox, RadioButton, and Toggle switch.



Calendar, date and time pickers

Calendar, date and time pickers enable users to view and modify date and time info, such as when creating an event or setting an alarm. Examples include calendar date picker, calendar view, date picker, time picker.



Predictive text entry

Provides suggestions as users type, such as when entering data or performing queries. Examples include AutoSuggestBox.

For a complete list, see Controls and UI elements

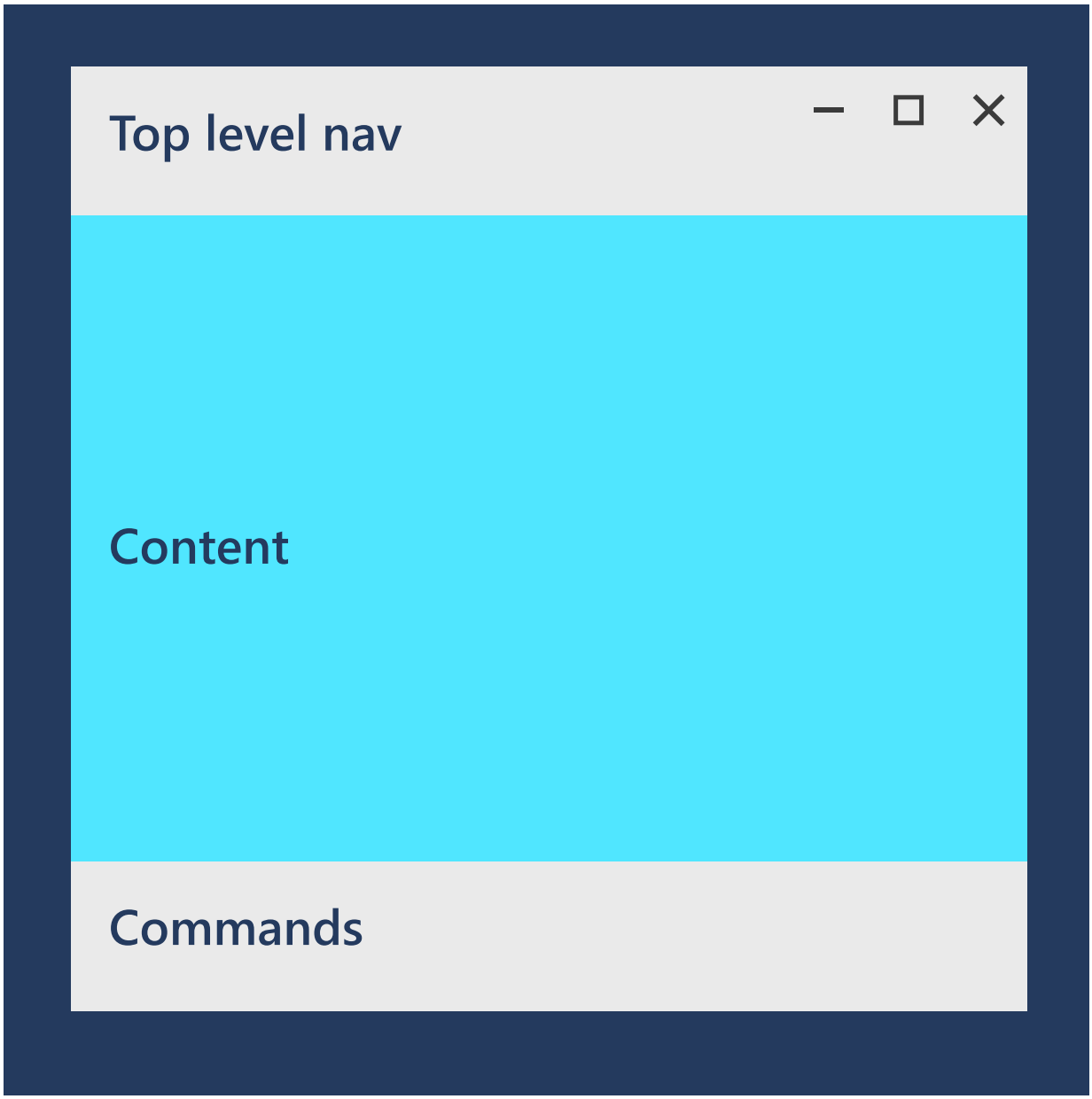
Place commands on the right surface

You can place command elements on a number of surfaces in your app, including the app canvas or special command containers, such as a command bar, command bar flyout, menu bar, or dialog.

Always try to let users manipulate content directly rather than through commands that act on the content, such as dragging and dropping to rearrange list items rather than up and down command buttons.

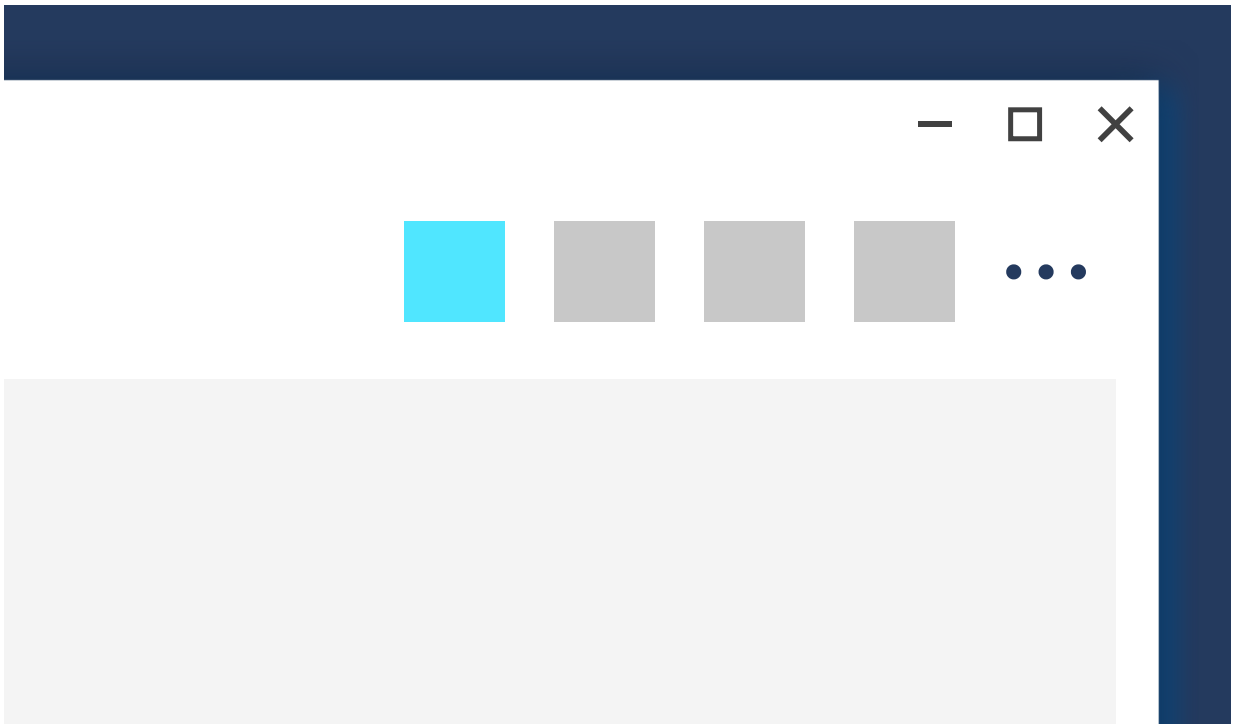
However, this might not be possible with certain input devices, or when accommodating specific user abilities and preferences. In these cases, provide as many commanding affordances as possible, and place these command elements on a command surface in your app.

Here's a list of some of the most common command surfaces.



App canvas (content area)

If a command is constantly needed for users to complete core scenarios, put it on the canvas. Because you can put commands near (or on) the objects they affect, putting commands on the canvas makes them easy and obvious to use. However, choose the commands you put on the canvas carefully. Too many commands on the app canvas take up valuable screen space and can overwhelm the user. If the command won't be frequently used, consider putting it in another command surface.



Command bars and menu bars

Command bars help organize commands and make them easy to access. Command bars can be placed at the top of the screen, at the bottom of the screen, or at both the top and bottom of the screen (a MenuBar can also be used when the functionality in your app is too complex for a command bar).

Menus and context menus

Menus and context menus save space by organizing commands and hiding them until the user needs them. Users typically access a menu or context menu by clicking a button or right-clicking a control.

The CommandBarFlyout is a type of contextual menu that combines the benefits of a command bar and a context menu into a single control. It can provide shortcuts to commonly-used actions and provide access to secondary commands that are only relevant in certain contexts, such as clipboard or custom commands.

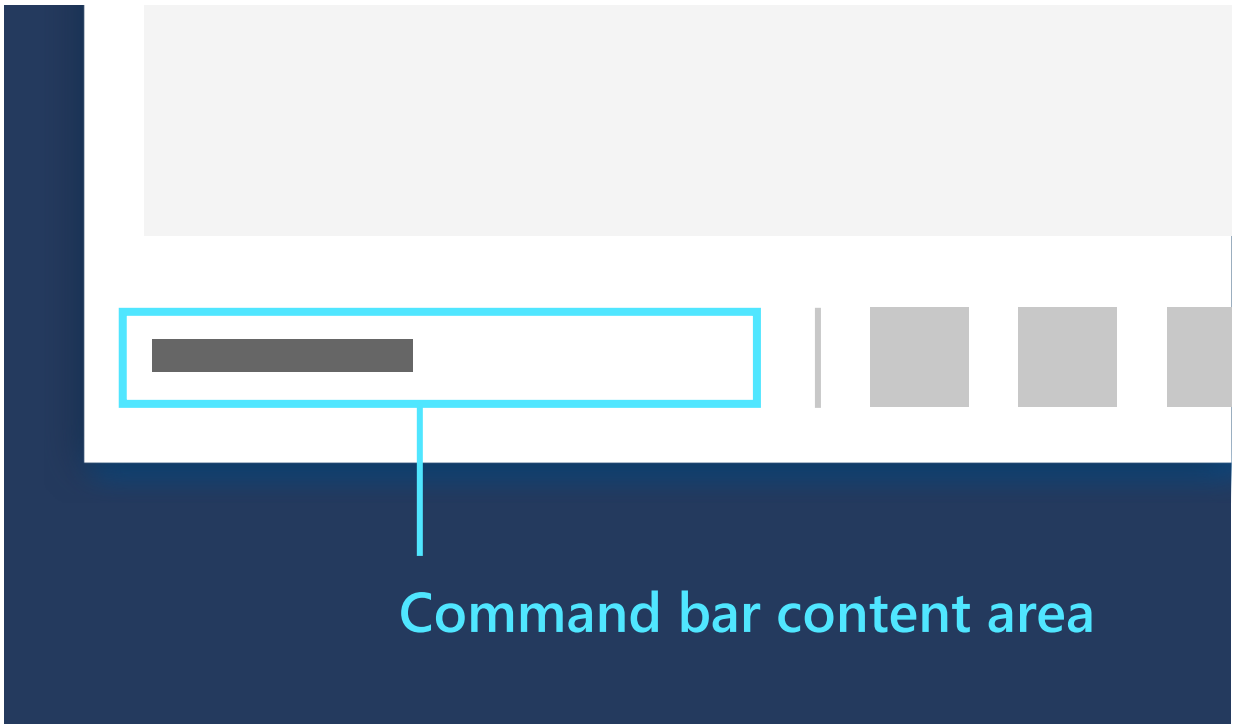
UWP also provides a set of traditional menus and context menus; for more info, see Menus and context menus.

Provide command feedback

Command feedback communicates to users that an interaction or command was detected, how the command was interpreted and handled, and whether the command was successful or not. This helps users understand what they've done, and what they can do next. Ideally, feedback should be integrated naturally in your UI, so users don't have to be interrupted, or take additional action unless absolutely necessary.

> **Note**
>
> Provide feedback only when necessary and only if it's not available elsewhere. Keep your application UI clean and uncluttered unless you are adding value.

Here are some ways to provide feedback in your app.



Command bar

The content area of the CommandBar is an intuitive place to communicate status to users if they'd like to see feedback.

Flyouts

Flyouts are lightweight contextual popups that can be dismissed by tapping or clicking somewhere outside the flyout.

Dialog controls

Dialog controls are modal UI overlays that provide contextual app information. In most cases, dialogs block interactions with the app window until being explicitly dismissed, and often request some kind of action from the user. Dialogs can be disruptive and should only be used in certain situations. For more info, see the When to confirm or undo actions section.

> **Tip**
>
> Be careful of how much your app uses confirmation dialogs; they can be very helpful when the user makes a mistake, but they are a hindrance whenever the user is trying to perform an action intentionally.

When to confirm or undo actions

No matter how well-designed your application's UI is, all users perform an action they wish they hadn't. Your app can help in these situations by requiring confirmation of an action, or by providing a way to undo recent actions.

For actions that can't be undone and have major consequences, we recommend using a confirmation dialog. Examples of such actions include:

- Overwriting a file

- Not saving a file before closing
- Confirming permanent deletion of a file or data
- Making a purchase (unless the user opts out of requiring a confirmation)
- Submitting a form, such as signing up for something

For actions that can be undone, offering a simple undo command is usually enough. Examples of such actions include:

- Deleting a file
- Deleting an email (not permanently)
- Modifying content or editing text
- Renaming a file

Optimize for specific input types

See the Interaction primer for more detail on optimizing user experiences around a specific input type or device.

Sound

Article • 05/17/2022

There are many ways to use sound to enhance your app. You can use to sound to supplement other UI elements, enabling users to recognize events audibly. Sound can be an effective user interface element for people with visual disabilities. You can use sound to create an atmosphere that immerses the user; for example, you might play a whimsical soundtrack in the background of puzzle game, or use ominous sound effects for a horror/survival game.

Examples

WinUI 2 Gallery

If you have the **WinUI 2 Gallery** app installed, click here to open the app and see Sound in action.

- Get the WinUI 2 Gallery app (Microsoft Store)
- Get the source code (GitHub)

Sound Global API

UWP provides an easily accessible sound system that allows you to simply "flip a switch" and get an immersive audio experience across your entire app.

The **ElementSoundPlayer** is an integrated sound system within XAML, and when turned on all default controls will play sounds automatically.

C#

```
ElementSoundPlayer.State = ElementSoundPlayerState.On;
```

The **ElementSoundPlayer** has three different states: **On Off** and **Auto**.

If set to **Off**, no matter where your app is run, sound will never play. If set to **On** sounds for your app will play on every platform.

Enabling ElementSoundPlayer will automatically enable spatial audio (3D sound) as well. To disable 3D sound (while still keeping the sound on), disable the **SpatialAudioMode** of the ElementSoundPlayer:

C#

```
ElementSoundPlayer.SpatialAudioMode = ElementSpatialAudioMode.Off
```

The **SpatialAudioMode** property can takes these values:

- **Auto**: Spatial audio will turn on when sound is on.
- **Off**: Spatial audio is always off, even if sound is on.
- **On**: Spatial audio will always play.

To learn more about spatial audio and how XAML handles it see AudioGraph - Spatial Audio.

Sound for TV and Xbox

Sound is a key part of the 10-foot experience, and by default, the **ElementSoundPlayer**'s state is **Auto**, meaning that you will only get sound when your app is running on Xbox. Please see Designing for Xbox and TV for more details.

Sound Volume Override

All sounds within the app can be dimmed with the **Volume** control. However, sounds within the app cannot get *louder than the system volume*.

To set the app volume level, call:

C#

```
ElementSoundPlayer.Volume = 0.5;
```

Where maximum volume (relative to system volume) is 1.0, and minimum is 0.0 (essentially silent).

Control Level State

If a control's default sound is not desired, it can be disabled. This is done through the **ElementSoundMode** on the control.

The **ElementSoundMode** has two states: **Off** and **Default**. When not set, it is **Default**. If set to **Off**, every sound that control plays will be muted *except for focus*.

XAML

```
<Button Name="ButtonName" Content="More Info" ElementSoundMode="Off"/>
```

C#

```
ButtonName.ElementSoundState = ElementSoundMode.Off;
```

Is This The Right Sound?

When creating a custom control, or changing an existing control's sound, it is important to understand the usages of all the sounds the system provides.

Each sound relates to a certain basic user interaction, and although sounds can be customized to play on any interaction, this section serves to illustrate the scenarios where sounds should be used to maintain a consistent experience across all UWP apps.

Invoking an Element

The most common control-triggered sound in our system today is the **Invoke** sound. This sound plays when a user invokes a control through a tap/click/enter/space or press of the 'A' button on a gamepad.

Typically, this sound is only played when a user explicitly targets a simple control or control part through an input device.

To play this sound from any control event, simply call the Play method from **ElementSoundPlayer** and pass in **ElementSound.Invoke**:

C#

```
ElementSoundPlayer.Play(ElementSoundKind.Invoke);
```

Showing & Hiding Content

There are many flyouts, dialogs and dismissible UIs in XAML, and any action that triggers one of these overlays should call a **Show** or **Hide** sound.

When an overlay content window is brought into view, the **Show** sound should be called:

```
C#

ElementSoundPlayer.Play(ElementSoundKind.Show);
```

Conversely when an overlay content window is closed (or is light dismissed), the **Hide** sound should be called:

```
C#

ElementSoundPlayer.Play(ElementSoundKind.Hide);
```

Navigation Within a Page

When navigating between panels or views within an app's page (see NavigationView), there is typically bidirectional movement. Meaning you can move to the next view/panel or the previous one, without leaving the current app page you're on.

The audio experience around this navigation concept is encompassed by the **MovePrevious** and **MoveNext** sounds.

When moving to a view/panel that is considered the *next item* in a list, call:

```
C#

ElementSoundPlayer.Play(ElementSoundKind.MoveNext);
```

And when moving to a previous view/panel in a list considered the *previous item*, call:

```
C#

ElementSoundPlayer.Play(ElementSoundKind.MovePrevious);
```

Back Navigation

When navigating from the current page to the previous page within an app the **GoBack** sound should be called:

```
C#

ElementSoundPlayer.Play(ElementSoundKind.GoBack);
```

Focusing on an Element

The **Focus** sound is the only implicit sound in our system. Meaning a user isn't directly interacting with anything, but is still hearing a sound.

Focusing happens when a user navigates through an app, this can be with the gamepad/keyboard/remote or kinect. Typically the **Focus** sound *does not play on PointerEntered or mouse hover events*.

To set up a control to play the **Focus** sound when your control receives focus, call:

```
C#

ElementSoundPlayer.Play(ElementSoundKind.Focus);
```

Cycling Focus Sounds

As an added feature to calling **ElementSound.Focus**, the sound system will, by default, cycle through 4 different sounds on each navigation trigger. Meaning that no two exact focus sounds will play right after the other.

The purpose behind this cycling feature is to keep the focus sounds from becoming monotonous and from being noticeable by the user; focus sounds will be played most often and therefore should be the most subtle.

Get the sample code

- WinUI 2 Gallery sample - See all the XAML controls in an interactive format.

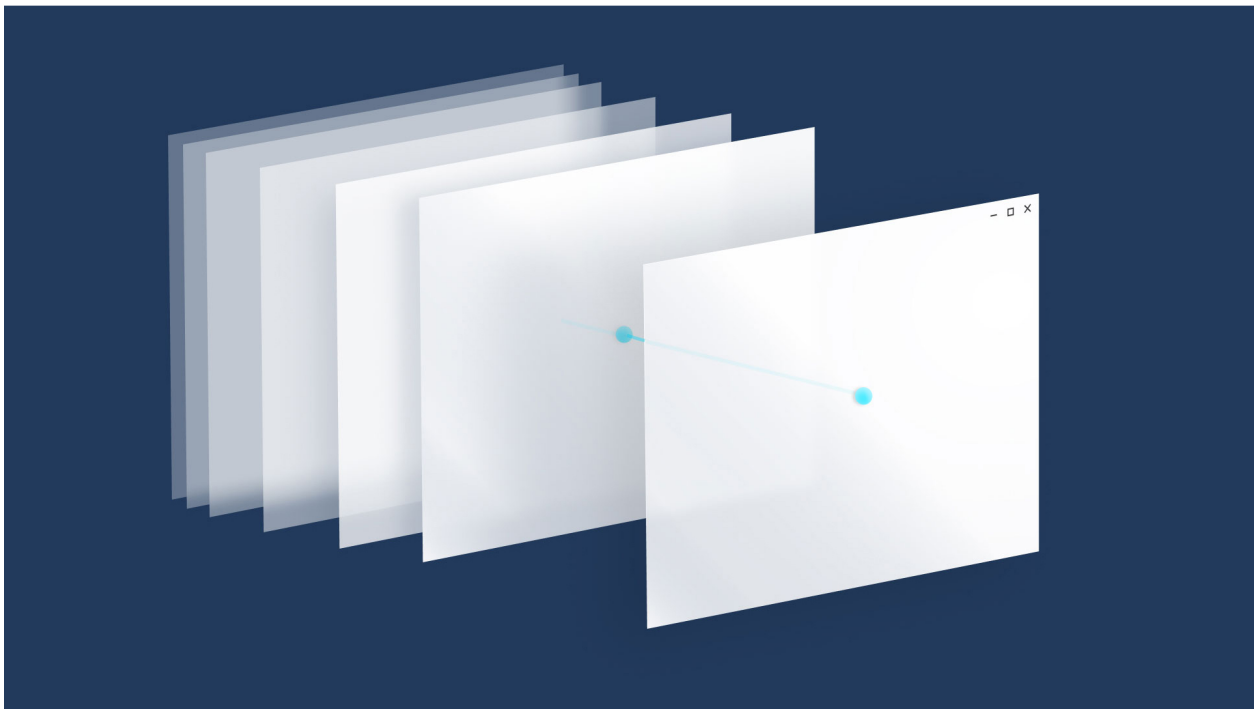
Related articles

- Designing for Xbox and TV

- ElementSoundPlayer class documentation

Navigation design basics for Windows apps

Article • 04/12/2023

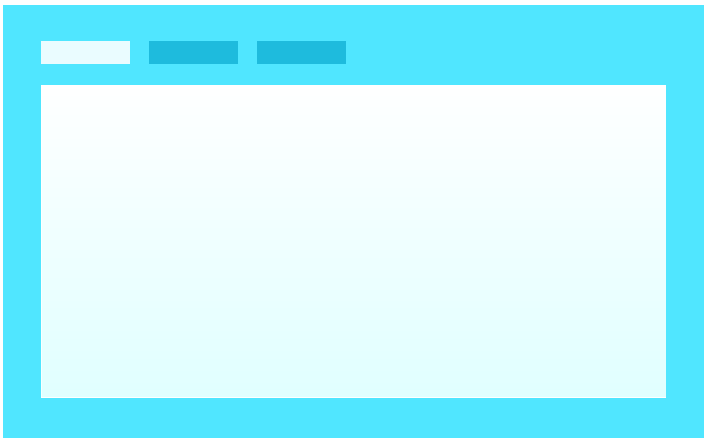


If you think of an app as a collection of pages, *navigation* describes the act of moving between pages and within a page. It's the starting point of the user experience, and it's how users find the content and features they're interested in. It's very important, and it can be difficult to get right.

You have a huge number of choices to make for navigation. You could:

Require users to go through a series of pages in order.

Provide a menu that allows users to jump directly to any page.



Place everything on a single page and provide filtering mechanisms for viewing content.

While there's no single navigation design that works for every app, there are principles and guidelines to help you decide the right design for your app.

Principles of good navigation

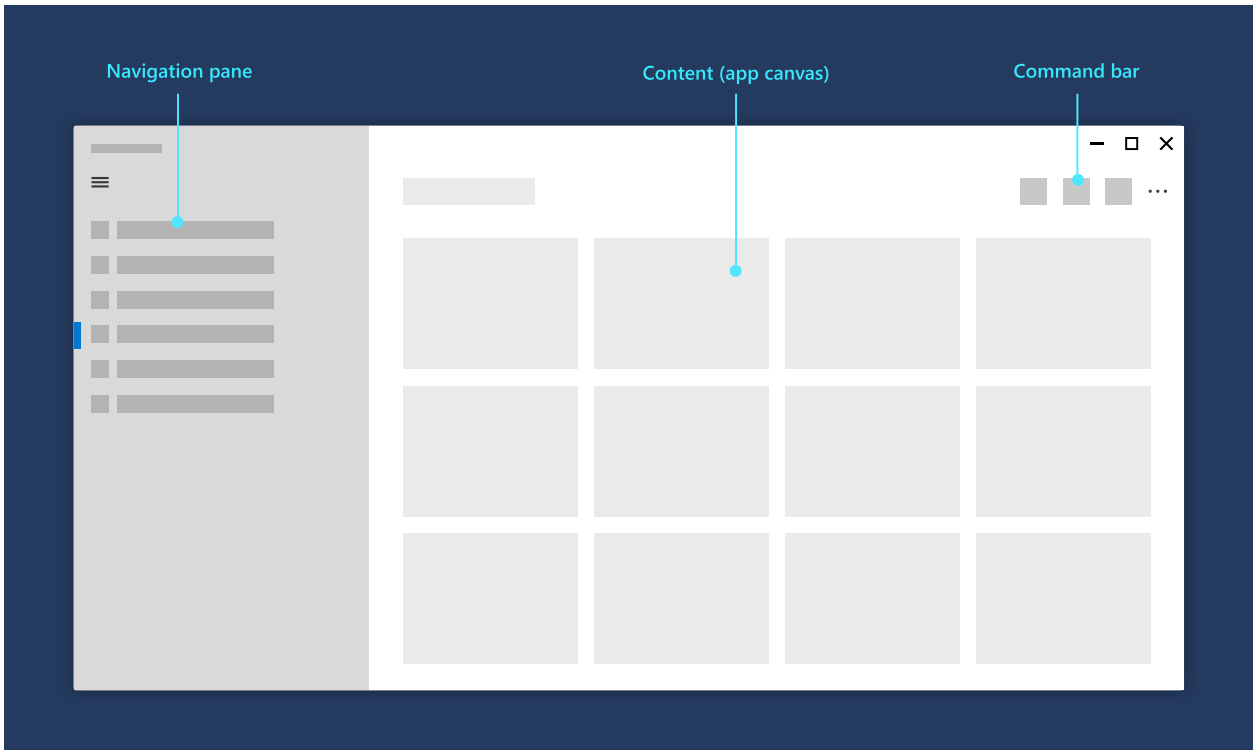
Let's start with the basic principles of good navigation design:

- **Consistency:** Meet user expectations.

- **Simplicity:** Don't do more than you need to.
- **Clarity:** Provide clear paths and options.

Consistency

Navigation should be consistent with user expectations. Using standard controls that users are familiar with and following standard conventions for icons, location, and styling will make navigation predictable and intuitive for users.

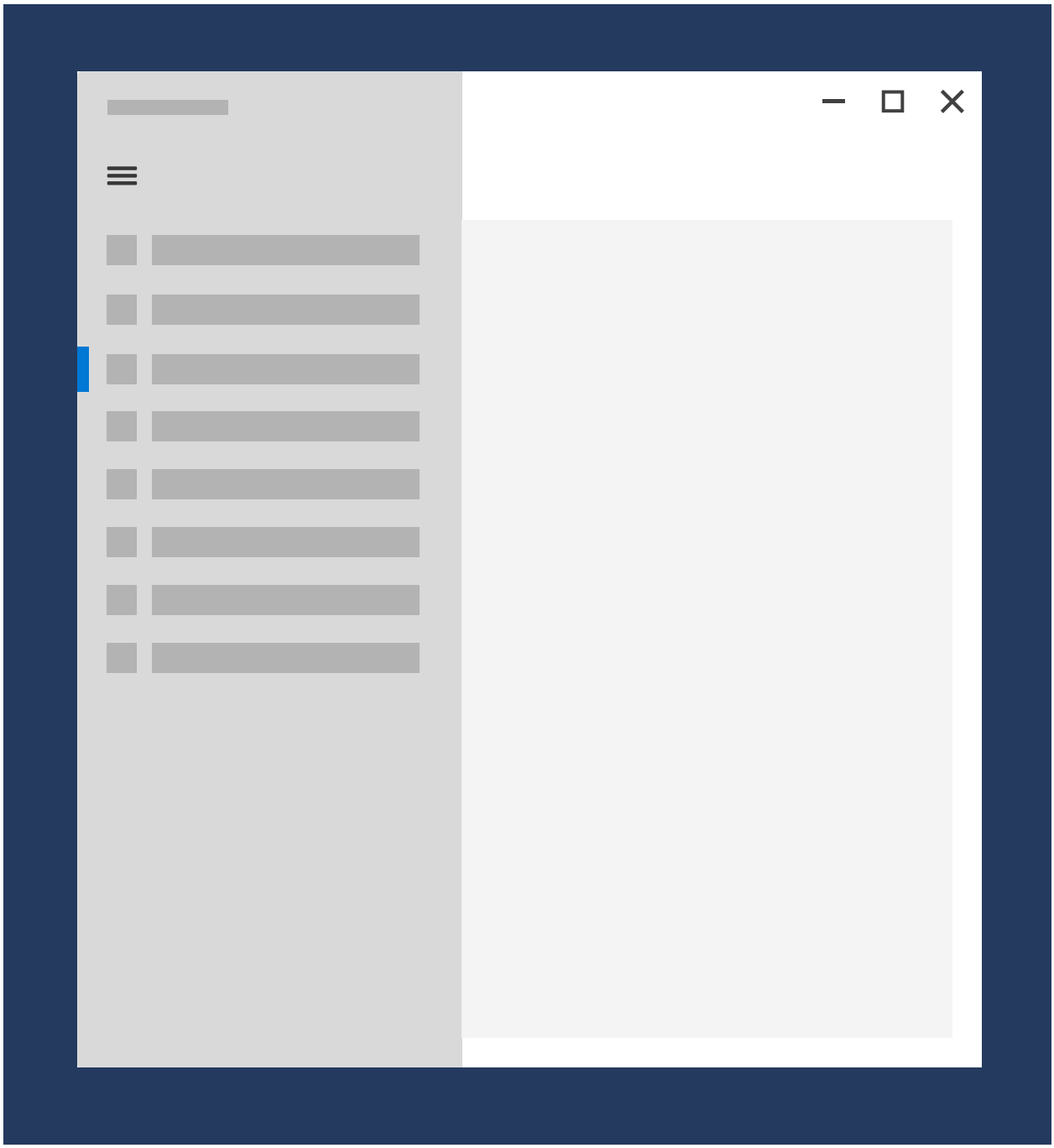


> *Users expect to find certain UI elements in standard locations.*

Simplicity

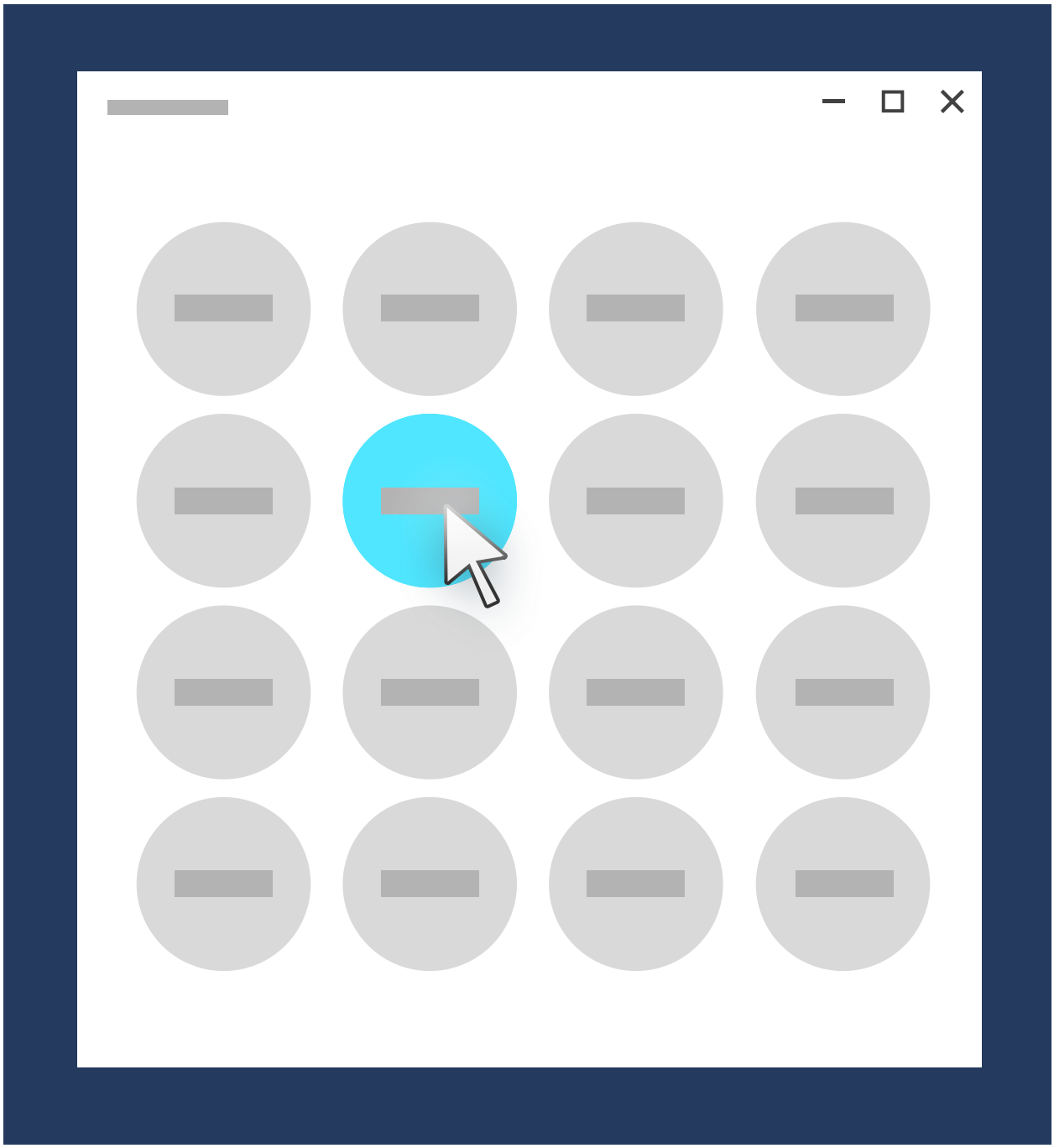
Fewer navigation items simplify decision making for users. Providing easy access to important destinations and hiding less important items will help users get where they want, faster.

Do



Present navigation items in a familiar navigation menu.

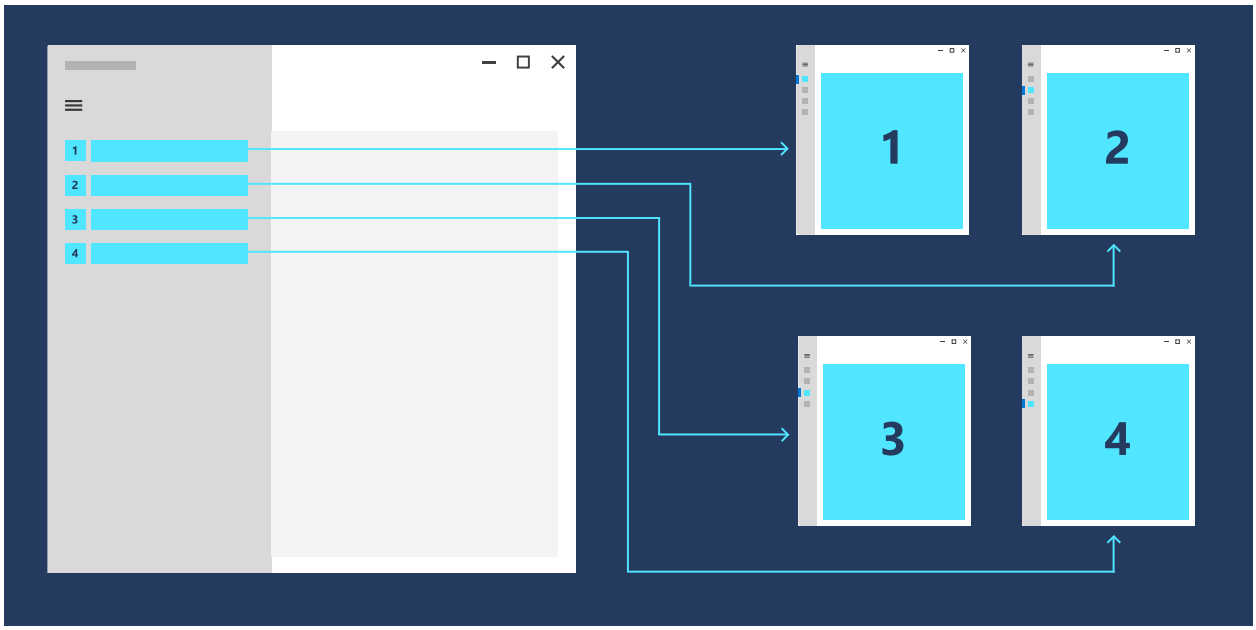
✕ Don’t



Don't overwhelm users with many navigation options.

Clarity

Clear paths allow for logical navigation for users. Making navigation options obvious and clarifying relationships between pages should prevent users from getting lost.



> *Destinations are clearly labeled so users know where they are.*

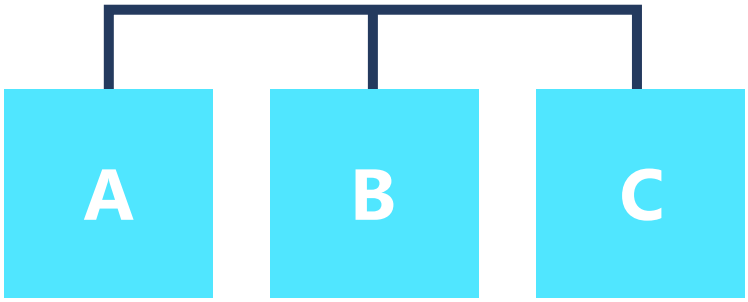
General recommendations

Now, let's take our design principles--consistency, simplicity, and clarity--and use them to come up with some general recommendations.

- Think about your users. Trace out typical paths they might take through your app, and for each page, think about why the user is there and where they might want to go.
- Avoid deep navigation hierarchies. If you go beyond two levels of navigation, provide a breadcrumb bar that shows the user where they are and let's them quickly get back out. Otherwise, you risk stranding your user in a deep hierarchy that they will have difficulty leaving.
- Avoid "pogo-sticking." Pogo-sticking occurs when there is related content, but navigating to it requires the user to go up a level and then down again.

Use the right structure

Now that you're familiar with general navigation principles, how should you structure your app? There are two general structures: flat and hierarchal.

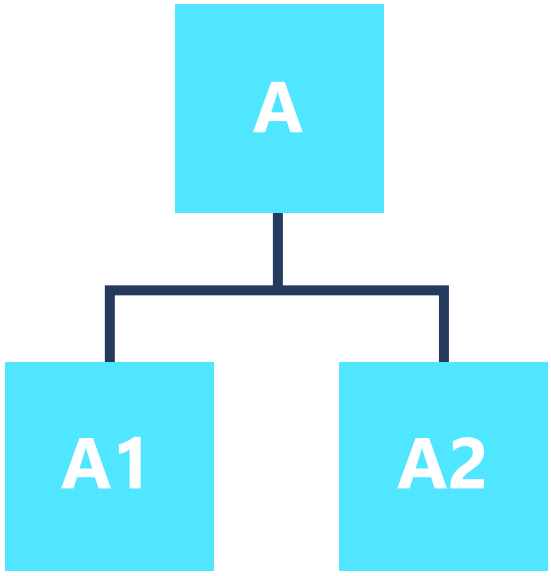


Flat/lateral

In a flat/lateral structure, pages exist side-by-side. You can go from one page to another in any order.

We recommend using a flat structure when:

- The pages can be viewed in any order.
- The pages are clearly distinct from each other and don't have an obvious parent/child relationship.
- There are less than 8 pages in the group.
  (When there are more pages, it might be difficult for users to understand how the pages are unique or to understand their current location within the group. If you don't think that's an issue for your app, go ahead and make the pages peers. Otherwise, consider using a hierarchical structure to break the pages into two or more smaller groups.)



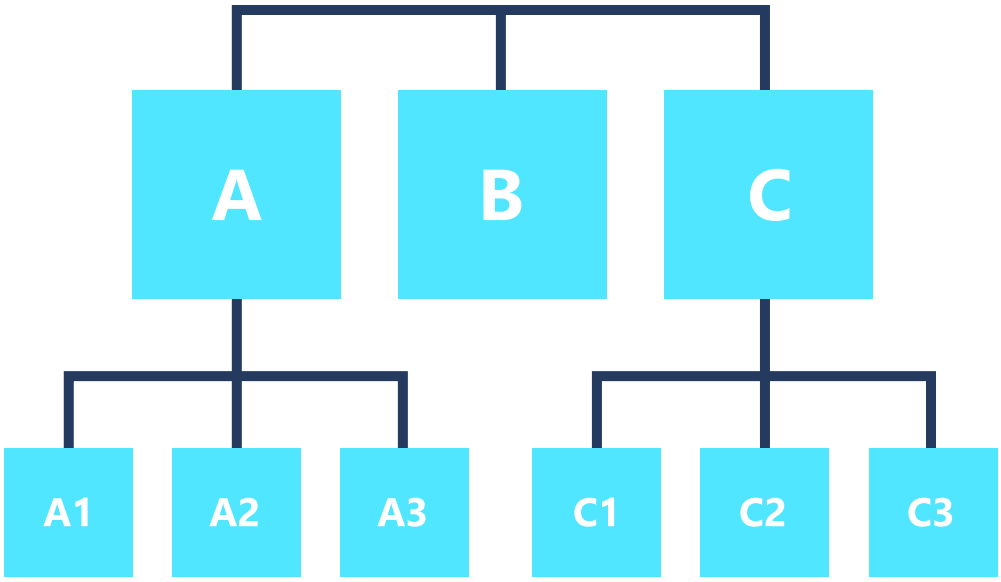
Hierarchical

In a hierarchical structure, pages are organized into a tree-like structure. Each child page has one parent, but a parent can have one or more child pages. To reach a child page, you travel through the parent.

Hierarchical structures are good for organizing complex content that spans lots of pages. The downside is some navigation overhead: the deeper the structure, the more clicks it takes to get from page to page.

We recommend a hierarchical structure when:

- Pages should be traversed in a specific order.
- There is a clear parent-child relationship between pages.
- There are more than 7 pages in the group.



Combining structures

You don't have to choose one structure or the other; many well-designed apps use both. An app can use flat structures for top-level pages that can be viewed in any order, and hierarchical structures for pages that have more complex relationships.

If your navigation structure has multiple levels, we recommend that peer-to-peer navigation elements only link to the peers within their current subtree. Consider the adjacent illustration, which shows a navigation structure that has two levels:

- At level 1, the peer-to-peer navigation element should provide access to pages A, B, and C.

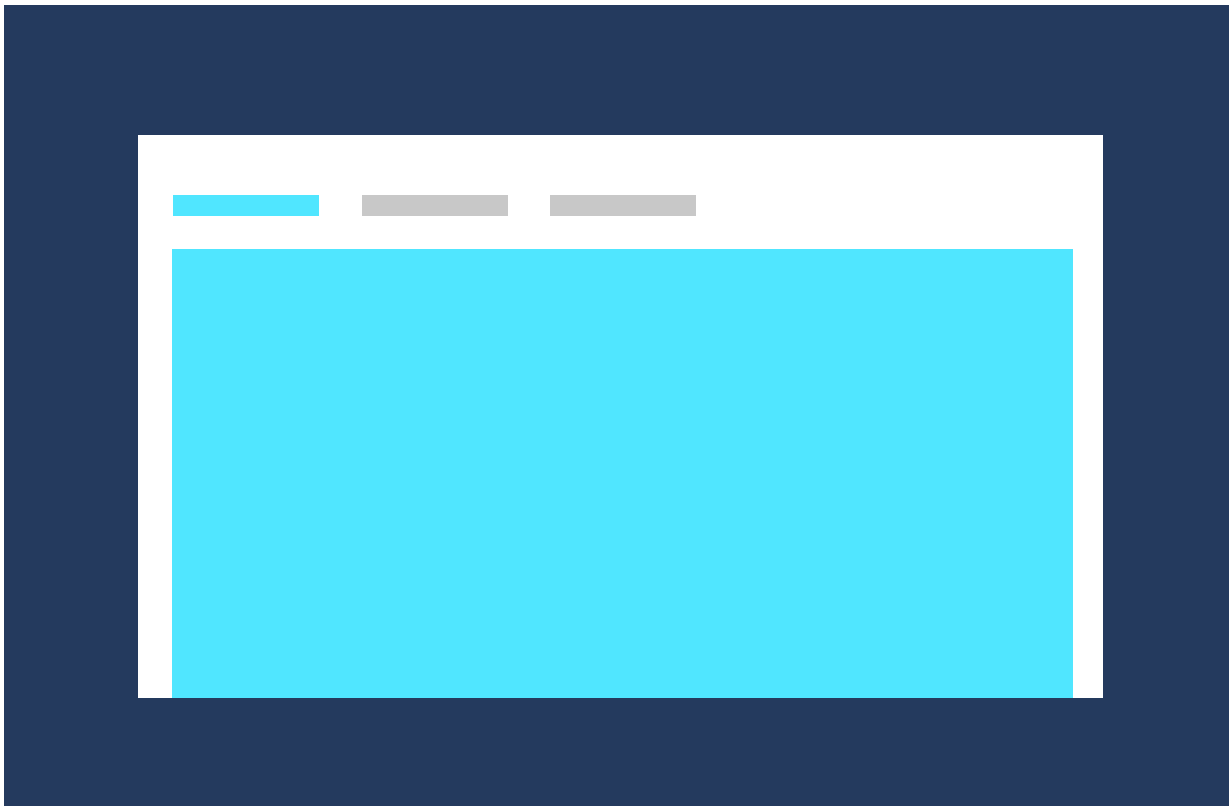
- At level 2, the peer-to-peer navigation elements for the A2 pages should only link to the other A2 pages. They should not link to level 2 pages in the C subtree.

Use the right controls

Once you've decided on a page structure, you need to decide how users navigate through those pages. XAML provides a variety of navigation controls to help ensure a consistent, reliable navigation experience in your app.

Frame

With few exceptions, any app that has multiple pages uses a frame. Typically, an app has a main page that contains the frame and a primary navigation element, such as a navigation view control. When the user selects a page, the frame loads and displays it.

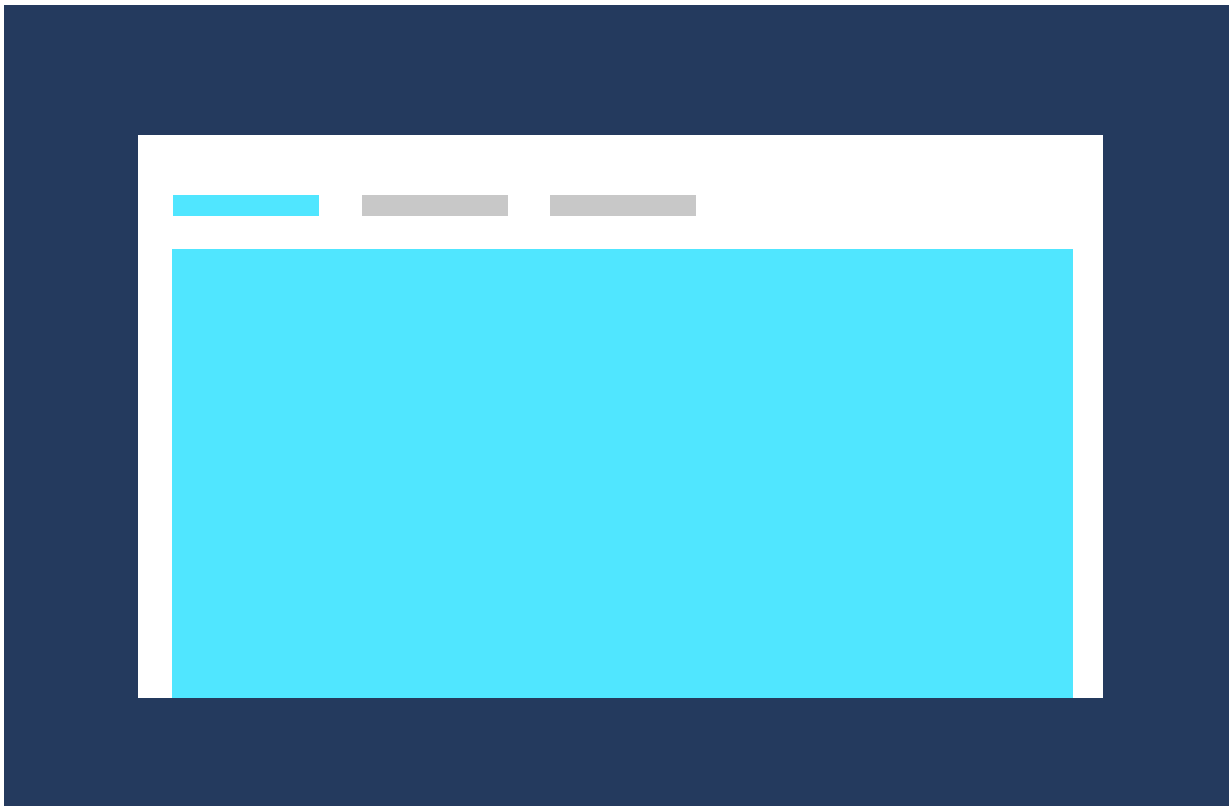


Top navigation

Displays a horizontal list of links to pages at the same level. The NavigationView control implements the top navigation pattern.

Use top navigation when:

- You want to show all navigation options on the screen.
- You desire more space for your app's content.
- Icons cannot clearly describe your navigation categories.

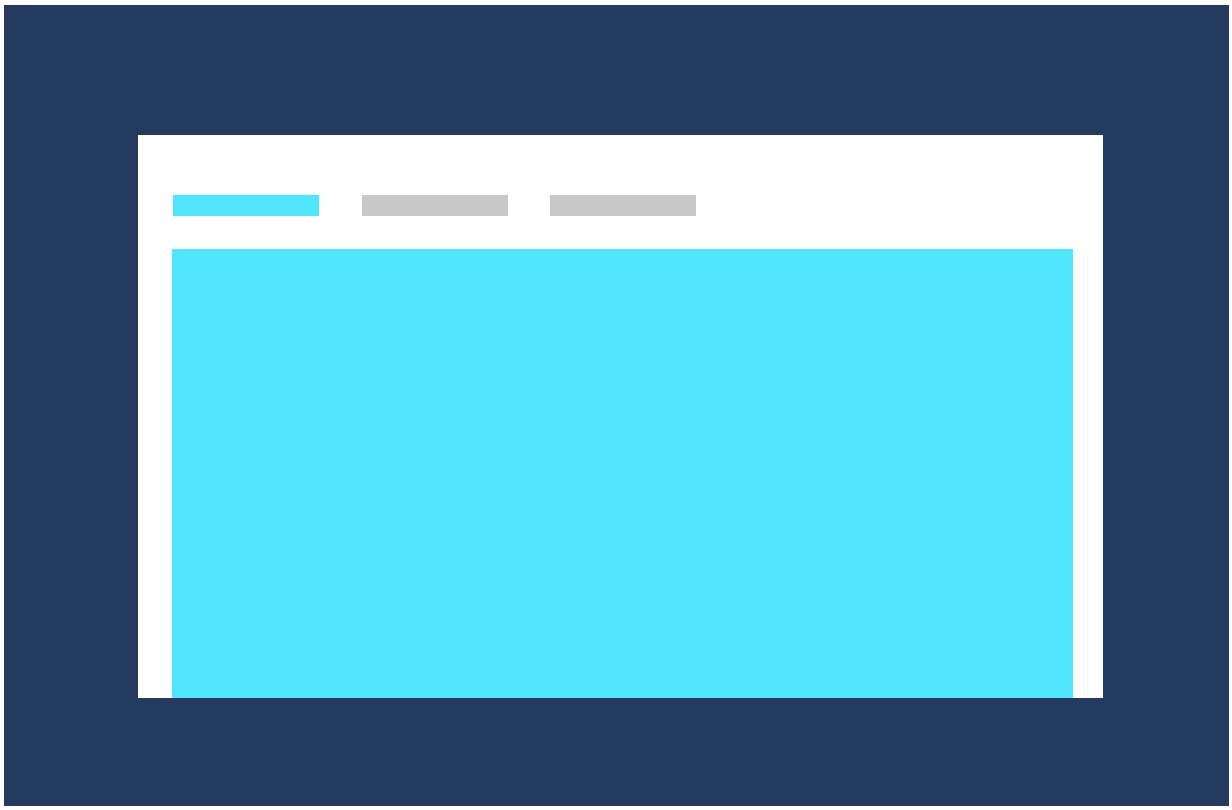


Tabs

Displays a horizontal set of tabs and their respective content. The TabView control is useful for displaying several pages (or documents) while giving the user the capability to rearrange, open, or close tabs.

Use tabs when:

- You want users to be able to dynamically open, close, or rearrange tabs.
- You expect that there might be a large number of tabs open at once.
- You expect users to be able to easily move tabs between windows in your application that use tabs, similar to web browsers like Microsoft Edge.

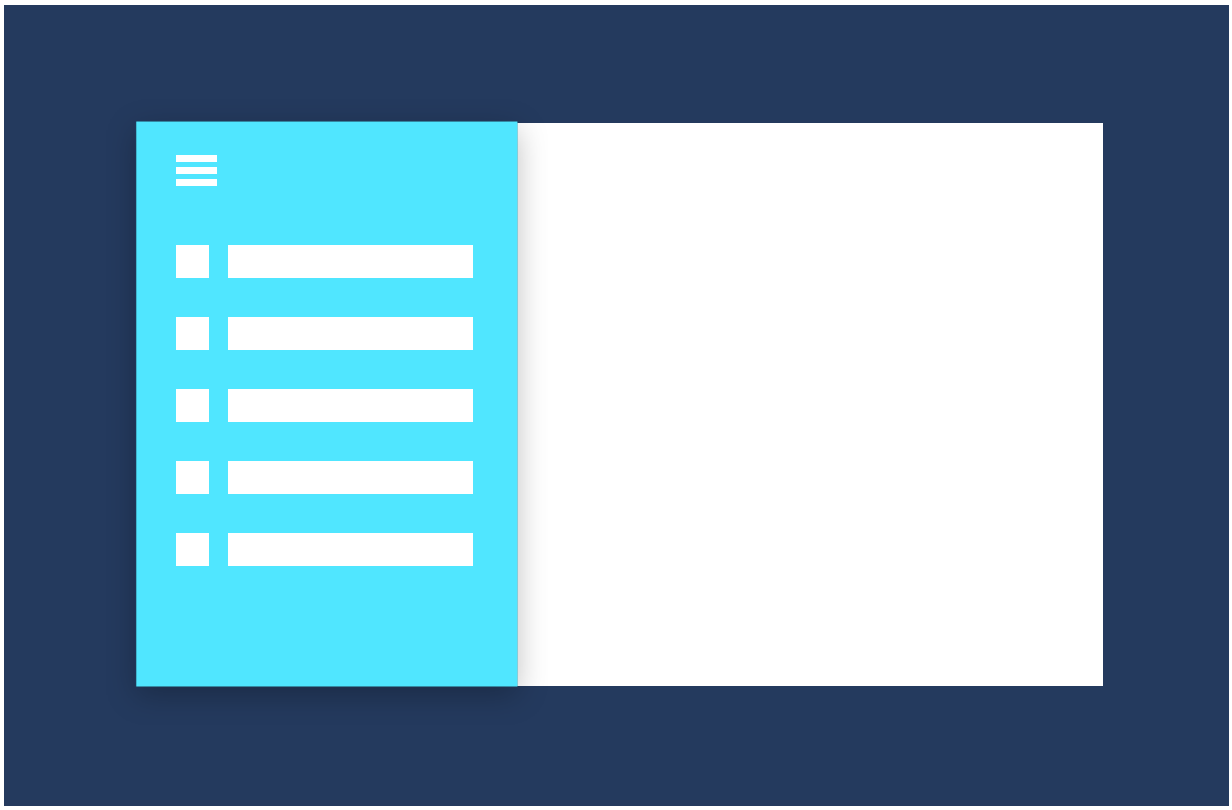


Breadcrumb

Displays a horizontal list of links to pages at each of the higher levels. The BreadcrumbBar control implements the top navigation pattern.

Use a breadcrumb when:

- You want to show the path to the current location
- You have many levels of navigation
- You expect users to be able to return to any previous level



Left navigation

Displays a vertical list of links to top-level pages. Use when:

- The pages exist at the top level.
- There are many navigation items (more than 5)
- You don't expect users to switch between pages frequently.

List/details

Displays a list of items. Selecting an item displays its corresponding page in the details section. Use when:

- You expect users to switch between child items frequently.
- You want to enable the user to perform high-level operations, such as deleting or sorting, on individual items or groups of items, and also want to enable the user to view or update the details for each item.

List/details is well suited for email inboxes, contact lists, and data entry.

Hyperlinks

Embedded navigation elements can appear in a page's content. Unlike other navigation elements, which should be consistent across the pages, content-embedded navigation elements are unique from page to page.

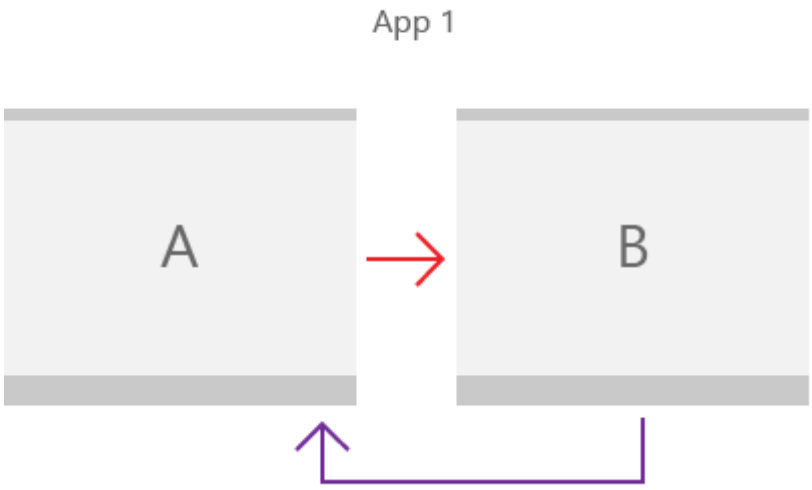
Next: Add navigation code to your app

The next article, Implement basic navigation, shows the code required to use a `Frame` control to enable basic navigation between two pages in your app.

Implement navigation between two pages

Article • 04/12/2023

Learn how to use a frame and pages to enable basic peer-to-peer navigation in your app.



Almost every app requires navigation between pages. Even a simple app with a single content page will typically have a settings page that requires navigation. In this article, we walk through the basics of adding a XAML `Page` to your app, and using a `Frame` to navigate between pages.

> **ⓘ Important**
>
> We use the **Blank App** template from Microsoft Visual Studio for this example. There are differences in the templates for Windows App SDK/WinUI 3 apps and UWP apps, so be sure to select the correct tab for your app type.

Windows App SDK

- ✓ **Applies to**: Windows App SDK/WinUI3
- ✓ **Important APIs**: Microsoft.UI.Xaml.Controls.Frame class, Microsoft.UI.Xaml.Controls.Page class, Microsoft.UI.Xaml.Navigation namespace

1. Create a blank app

Windows App SDK

To create a blank app in Visual Studio:

1. To set up your development computer, see Install tools for the Windows App SDK.
2. From the Microsoft Visual Studio start window, select **Create a new project**, OR, on the Visual Studio menu, choose **File** > **New** > **Project**.
3. In the **Create a new project** dialog's drop-down filters, select **C#** or **C++**, **Windows**, and **WinUI**, respectively.
4. Select the **Blank App, Packaged (WinUI 3 in Desktop)** project template, and click **Next**. That template creates a desktop app with a WinUI 3-based user interface.
5. In the **Project name** box, enter `BasicNavigation`, and click **Create**.
6. To run the program, choose **Debug** > **Start Debugging** from the menu, or press F5. Build and run your solution on your development computer to confirm that the app runs without errors. A blank page is displayed.
7. To stop debugging and return to Visual Studio, exit the app, or click **Stop Debugging** from the menu.
8. Remove any example code that's included in the template from the `MainWindow.xaml` and `MainWindow` code-behind files.

> **Tip**
>
> For more info, see **Create your first WinUI 3 (Windows App SDK) project**.

2. Use a Frame to navigate between pages

When your app has multiple pages, you use a Frame to navigate between them. The `Frame` class supports various navigation methods such as Navigate, GoBack, and GoForward, and properties such as BackStack, ForwardStack, and BackStackDepth.

Windows App SDK

When you create a new Windows App SDK project in Visual Studio, the project template creates a `MainWindow` class (of type Microsoft.UI.Xaml.Window). However, it doesn't create a Frame or Page and doesn't provide any navigation code.

To enable navigation between pages, add a `Frame` as the root element of `MainWindow`. You can do that in the Application.OnLaunched method override in the `App.xaml` code-behind file. Open the `App` code-behind file, update the `OnLaunched` override, and handle the NavigationFailed event as shown here.

C#

```
// App.xaml.cs

protected override void
OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
{
    m_window = new MainWindow();

    // Create a Frame to act as the navigation context and navigate to
the first page
    Frame rootFrame = new Frame();
    rootFrame.NavigationFailed += OnNavigationFailed;
    // Navigate to the first page, configuring the new page
    // by passing required information as a navigation parameter
    rootFrame.Navigate(typeof(MainPage), args.Arguments);

    // Place the frame in the current Window
    m_window.Content = rootFrame;
    // Ensure the MainWindow is active
    m_window.Activate();
}

void OnNavigationFailed(object sender, NavigationFailedEventArgs e)
{
    throw new Exception("Failed to load Page " +
e.SourcePageType.FullName);
}
```

> **Note**
>
> For apps with more complex navigation, you will typically use a **NavigationView** as the root of MainWindow, and place a `Frame` as the content of the navigation view. For more info, see **Navigation view**.

The Navigate method is used to display content in this `Frame`. Here, `MainPage.xaml` is passed to the `Navigate` method, so the method loads `MainPage` in the `Frame`.

If the navigation to the app's initial window fails, a `NavigationFailed` event occurs, and this code throws an exception in the event handler.

3. Add basic pages

The **Blank App** template doesn't create multiple app pages for you. Before you can navigate between pages, you need to add some pages to your app.

Windows App SDK

To add a new item to your app:

1. In **Solution Explorer**, right-click the `BasicNavigation` project node to open the context menu.
2. Choose **Add** > **New Item** from the context menu.
3. In the **Add New Item** dialog box, select the **WinUI** node in the left pane, then choose **Blank Page (WinUI 3)** in the middle pane.
4. In the **Name** box, enter `MainPage` and press the **Add** button.
5. Repeat steps 1-4 to add the second page, but in the **Name** box, enter `Page2`.

Now, these files should be listed as part of your `BasicNavigation` project.

C#	C++
• MainPage.xaml • MainPage.xaml.cs • Page2.xaml • Page2.xaml.cs	• MainPage.xaml • MainPage.xaml.cpp • MainPage.xaml.h • Page2.xaml • Page2.xaml.cpp • Page2.xaml.h

> **ⓘ Important**
>
> **For C++ projects**, you must add a `#include` directive in the header file of each page that references another page. For the inter-page navigation example presented here, mainpage.xaml.h file contains `#include "Page2.xaml.h"`, in turn, page2.xaml.h contains `#include "MainPage.xaml.h"`.
>
> C++ page templates also include an example `Button` and click handler code that you will need to remove from the XAML and code-behind files for the page.

Add content to the pages

In `MainPage.xaml`, replace the existing page content with the following content:

XAML

```
<Grid>
    <TextBlock x:Name="pageTitle" Text="Main Page"
               Margin="16" Style="{StaticResource TitleTextBlockStyle}"/>
    <HyperlinkButton Content="Click to go to page 2"
                     Click="HyperlinkButton_Click"
                     HorizontalAlignment="Center"/>
</Grid>
```

This XAML adds:

- A TextBlock element named `pageTitle` with its Text property set to `Main Page` as a child element of the root Grid.
- A HyperlinkButton element that is used to navigate to the next page as a child element of the root Grid.

In the `MainPage` code-behind file, add the following code to handle the `Click` event of the HyperlinkButton you added to enable navigation to `Page2.xaml`.

Windows App SDK

C#

```
// MainPage.xaml.cs

private void HyperlinkButton_Click(object sender, RoutedEventArgs e)
{
    Frame.Navigate(typeof(Page2));
}
```

`MainPage` is a subclass of the Page class. The `Page` class has a read-only Frame property that gets the `Frame` containing the `Page`. When the `Click` event handler of the `HyperlinkButton` in `MainPage` calls `Frame.Navigate(typeof(Page2))`, the `Frame` displays the content of `Page2.xaml`.

Whenever a page is loaded into the frame, that page is added as a PageStackEntry to the BackStack or ForwardStack of the Frame, allowing for history and backwards navigation.

Now, do the same in `Page2.xaml`. Replace the existing page content with the following content:

XAML

```
<Grid>
    <TextBlock x:Name="pageTitle" Text="Page 2"
               Margin="16" Style="{StaticResource TitleTextBlockStyle}"/>
    <HyperlinkButton Content="Click to go to main page"
                     Click="HyperlinkButton_Click"
                     HorizontalAlignment="Center"/>
</Grid>
```

In the `Page2` code-behind file, add the following code to handle the `Click` event of the HyperlinkButton to navigate to `MainPage.xaml`.

Windows App SDK

C#

```
// Page2.xaml.cs

private void HyperlinkButton_Click(object sender, RoutedEventArgs e)
{
    Frame.Navigate(typeof(MainPage));
}
```

Build and run the app. Click the link that says "Click to go to page 2". The second page that says "Page 2" at the top should be loaded and displayed in the frame. Now click the link on Page 2 to go back to Main Page.

4. Pass information between pages

Your app now navigates between two pages, but it really doesn't do anything interesting yet. Often, when an app has multiple pages, the pages need to share information. Now you'll pass some information from the first page to the second page.

In `MainPage.xaml`, replace the `HyperlinkButton` you added earlier with the following StackPanel. This adds a TextBlock label and a TextBox `name` for entering a text string.

XAML

```
<StackPanel VerticalAlignment="Center">
    <TextBlock HorizontalAlignment="Center" Text="Enter your name"/>
    <TextBox HorizontalAlignment="Center" Width="200" x:Name="name"/>
    <HyperlinkButton Content="Click to go to page 2"
                             Click="HyperlinkButton_Click"
```

```
                                HorizontalAlignment="Center"/>
</StackPanel>
```

Now you'll use the second overload of the `Navigate` method and pass the text from the text box as the second parameter. Here's the signature of this `Navigate` overload:

C#

```
public bool Navigate(System.Type sourcePageType, object parameter);
```

In the `HyperlinkButton_Click` event handler of the `MainPage` code-behind file, add a second parameter to the `Navigate` method that references the `Text` property of the `name` text box.

Windows App SDK

C#

```
// MainPage.xaml.cs

private void HyperlinkButton_Click(object sender, RoutedEventArgs e)
{
    Frame.Navigate(typeof(Page2), name.Text);
}
```

In `Page2.xaml`, replace the `HyperlinkButton` you added earlier with the following `StackPanel`. This adds a TextBlock for displaying the text string passed from `MainPage`.

XAML

```
<StackPanel VerticalAlignment="Center">
    <TextBlock HorizontalAlignment="Center" x:Name="greeting"/>
    <HyperlinkButton Content="Click to go to page 1"
                     Click="HyperlinkButton_Click"
                     HorizontalAlignment="Center"/>
</StackPanel>
```

In the `Page2` code-behind file, add the following code to override the `OnNavigatedTo` method:

Windows App SDK

C#

```
// Page2.xaml.cs

protected override void OnNavigatedTo(NavigationEventArgs e)
{
    if (e.Parameter is string &&
!string.IsNullOrWhiteSpace((string)e.Parameter))
    {
        greeting.Text = $"Hello, {e.Parameter.ToString()}";
    }
    else
    {
        greeting.Text = "Hello!";
    }
    base.OnNavigatedTo(e);
}
```

Run the app, type your name in the text box, and then click the link that says `Click to go to page 2`.

When the `Click` event of the `HyperlinkButton` in `MainPage` calls `Frame.Navigate(typeof(Page2), name.Text)`, the `name.Text` property is passed to `Page2`, and the value from the event data is used for the message displayed on the page.

5. Cache a page

Page content and state is not cached by default, so if you'd like to cache information, you must enable it in each page of your app.

In our basic peer-to-peer example, when you click the `Click to go to page 1` link on `Page2`, the `TextBox` (and any other field) on `MainPage` is set to its default state. One way to work around this is to use the NavigationCacheMode property to specify that a page be added to the frame's page cache.

By default, a new page instance is created with its default values every time navigation occurs. In `MainPage.xaml`, set `NavigationCacheMode` to `Enabled` (in the opening `Page` tag) to cache the page and retain all content and state values for the page until the page cache for the frame is exceeded. Set NavigationCacheMode to Required if you want to ignore CacheSize limits, which specify the number of pages in the navigation history that can be cached for the frame. However, keep in mind that cache size limits might be crucial, depending on the memory limits of a device.

XAML

```
<Page
    x:Class="BasicNavigation.MainPage"
    ...
    mc:Ignorable="d"
    NavigationCacheMode="Enabled">
```

Now, when you click back to main page, the name you entered in the text box is still there.

6. Customize page transition animations

By default, each page is animated into the frame when navigation occurs. The default animation is an "entrance" animation that causes the page to slide up from the bottom of the window. However, you can choose different animation options that better suit the navigation of your app. For example, you can use a "drill in" animation to give the feeling that the user is going deeper into your app, or a horizontal slide animation to give the feeling that two pages are peers. For more info, see Page transitions.

These animations are represented by sub-classes of NavigationTransitionInfo. To specify the animation to use for a page transition, you'll use the third overload of the `Navigate` method and pass a `NavigationTransitionInfo` sub-class as the third parameter (`infoOverride`). Here's the signature of this `Navigate` overload:

C#

```
public bool Navigate(System.Type sourcePageType,
                     object parameter,
                     NavigationTransitionInfo infoOverride);
```

In the `HyperlinkButton_Click` event handler of the `MainPage` code-behind file, add a third parameter to the `Navigate` method that sets the `infoOverride` parameter to a SlideNavigationTransitionInfo with its Effect property set to FromRight.

Windows App SDK

C#

```
// MainPage.xaml.cs

private void HyperlinkButton_Click(object sender, RoutedEventArgs e)
{
    Frame.Navigate(typeof(Page2),
                   name.Text,
```

```
                    new SlideNavigationTransitionInfo()
                        { Effect =
SlideNavigationTransitionEffect.FromRight});
}
```

In the `HyperlinkButton_Click` event handler of the `Page2` code-behind file, set the `infoOverride` parameter to a SlideNavigationTransitionInfo with its Effect property set to FromLeft.

Windows App SDK

C#

```
// Page2.xaml.cs

private void HyperlinkButton_Click(object sender, RoutedEventArgs e)
{
    Frame.Navigate(typeof(MainPage),
                    null,
                    new SlideNavigationTransitionInfo()
                        { Effect =
SlideNavigationTransitionEffect.FromLeft});
}
```

Now, when you navigate between pages, the pages slide left and right, which provides a more natural feeling for this transition and reinforces the connection between the pages.

Related articles

- Navigation design basics for Windows apps
- Navigation view
- Navigation history and backwards navigation

Navigation history and backwards navigation for Windows apps

Article • 05/24/2022

> **Important APIs**: BackRequested event, SystemNavigationManager class, OnNavigatedTo

The Windows app provides a consistent back navigation system for traversing the user's navigation history within an app and, depending on the device, from app to app.

To implement backwards navigation in your app, place a back button at the top left corner of your app's UI. The user expects the back button to navigate to the previous location in the app's navigation history. Note that it's up to you to decide which navigation actions to add to the navigation history and how to respond to the back button press.

For most apps that have multiple pages, we recommend that you use the NavigationView control to provide the navigation framework for your app. It adapts to a variety of screen sizes and supports both *top* and *left* navigation styles. If your app uses the `NavigationView` control, then you can use NavigationView's built-in back button.

> ⓘ **Note**
>
> The guidelines and examples in this article should be used when you implement navigation without using the `NavigationView` control. If you use `NavigationView`, this information provides useful background knowledge, but you should use the specific guidance and examples in the **NavigationView** article

Back button

To create a back button, use the Button control with the `NavigationBackButtonNormalStyle` style, and place the button at the top left hand corner of your app's UI (for details, see the XAML code examples below).

XAML

```
<Page>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <Button x:Name="BackButton"
                Style="{StaticResource NavigationBackButtonNormalStyle}"
                IsEnabled="{x:Bind Frame.CanGoBack, Mode=OneWay}"
                ToolTipService.ToolTip="Back"/>

    </Grid>
</Page>
```

If your app has a top CommandBar, the Button control that is 44epx in height will not align with 48epx AppBarButtons very nicely. However, to avoid inconsistency, align the top of the Button control inside the 48epx bounds.

XAML

```
<Page>
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <CommandBar>
            <CommandBar.Content>
                <Button x:Name="BackButton"
                        Style="{StaticResource
NavigationBackButtonNormalStyle}"
                        IsEnabled="{x:Bind Frame.CanGoBack, Mode=OneWay}"
                        ToolTipService.ToolTip="Back"
                        VerticalAlignment="Top"/>
            </CommandBar.Content>

            <AppBarButton Icon="Delete" Label="Delete"/>
            <AppBarButton Icon="Save" Label="Save"/>
        </CommandBar>
    </Grid>
</Page>
```

In order to minimize UI elements moving around in your app, show a disabled back button when there is nothing in the backstack (`IsEnabled="{x:Bind Frame.CanGoBack, Mode=OneWay}"`). However, if you expect your app will never have a backstack, you don't need to display the back button at all.

Optimize for different devices and inputs

This backwards navigation design guidance is applicable to all devices, but your users will benefit if you optimize for different form factors and methods of input.

To optimize your UI:

- **Desktop/Hub**: Draw the in-app back button on the top left corner of your app's UI.
- **Tablet Mode**: A hardware or software back button might be present on tablets, but we recommend drawing an in-app back button for clarity.
- **Xbox/TV**: Do not draw a back button; it will add unnecessary UI clutter. Instead, rely on the game pad B button to navigate backwards.

If your app will run on Xbox, create a custom visual trigger for Xbox to toggle the visibility of the button. If you use a NavigationView control, it will automatically toggle the back button's visibility when your app is running on Xbox.

We recommend handling the following events (in addition to the back button Click) to support the most common inputs for back navigation.

Event	Input
CoreDispatcher.AcceleratorKeyActivated	Alt+Left arrow, VirtualKey.GoBack
SystemNavigationManager.BackRequested	Windows + Backspace, Gamepad B button, Tablet Mode back button, Hardware back button
CoreWindow.PointerPressed	VirtualKey.XButton1 (Such as the back button found on some mice.)

Code examples

This section demonstrates how to handle back navigation using a variety of inputs.

Back button and back navigation

At a minimum, you need to handle the back button `Click` event and provide the code to perform the back navigation. You should also disable the back button when the backstack is empty.

This example code demonstrates how to implement backwards navigation behavior with a back button. The code responds to the Button Click event to navigate. The back

button is enabled or disabled in the OnNavigatedTo method, which is called when navigating to a new page.

The code is shown for `MainPage`, but you add this code to each page that supports back navigation. To avoid duplication, you can put the navigation related code in the `App` class in the `App.xaml.*` code-behind page.

XAML

```
<!-- MainPage.xaml -->
<Page x:Class="AppName.MainPage">
...
        <Button x:Name="BackButton" Click="BackButton_Click"
                Style="{StaticResource NavigationBackButtonNormalStyle}"
                IsEnabled="{x:Bind Frame.CanGoBack, Mode=OneWay}"
                ToolTipService.ToolTip="Back"/>
...
<Page/>
```

Code-behind:

C#

```
// MainPage.xaml.cs
private void BackButton_Click(object sender, RoutedEventArgs e)
{
    App.TryGoBack();
}

// App.xaml.cs
//
// Add this method to the App class.
public static bool TryGoBack()
{
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame.CanGoBack)
    {
        rootFrame.GoBack();
        return true;
    }
    return false;
}
```

Support access keys

Keyboard support is integral to ensuring your applications work great for users with different skills, abilities, and expectations. We recommend that you support accelerator

keys for both forward and back navigation because users who rely on these will expect both. For more info, see Keyboard interactions and Keyboard accelerators.

The common accelerator keys for forward and back navigation are Alt+Right arrow (forward) and Alt+Left arrow (back). To support these keys for navigation, handle the CoreDispatcher.AcceleratorKeyActivated event. You handle an event that's directly on the Window (rather than an element on the page) so the app responds to the accelerator keys regardless of which element has focus.

Add the code to the `App` class to support accelerator keys and forward navigation, as shown here. (This assumes that the previous code to support the back button has already been added.) You can see all the `App` code together at the end of the Code examples section.

C#

```
// App.xaml.cs
// Add event handler in OnLaunced.
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    // ...
    // Do not repeat app initialization when the Window already has content,
    // just ensure that the window is active
    if (rootFrame == null)
    {
        // ...
        // rootFrame.NavigationFailed += OnNavigationFailed;

        // Add support for accelerator keys.
        // Listen to the window directly so the app responds
        // to accelerator keys regardless of which element has focus.
        Window.Current.CoreWindow.Dispatcher.AcceleratorKeyActivated +=
            CoreDispatcher_AcceleratorKeyActivated;

        // ...

    }
}

// ...

// Add this code after the TryGoBack method added previously.
// Perform forward navigation if possible.
private bool TryGoForward()
{
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame.CanGoForward)
    {
        rootFrame.GoForward();
        return true;
    }
```

```
    return false;
}

// Invoked on every keystroke, including system keys such as Alt key
combinations.
// Used to detect keyboard navigation between pages even when the page
itself
// doesn't have focus.
private void CoreDispatcher_AcceleratorKeyActivated(CoreDispatcher sender,
AcceleratorKeyEventArgs e)
{
    // When Alt+Left are pressed navigate back.
    // When Alt+Right are pressed navigate forward.
    if (e.EventType == CoreAcceleratorKeyEventType.SystemKeyDown
        && (e.VirtualKey == VirtualKey.Left || e.VirtualKey ==
VirtualKey.Right)
        && e.KeyStatus.IsMenuKeyDown == true
        && !e.Handled)
    {
        if (e.VirtualKey == VirtualKey.Left)
        {
            e.Handled = TryGoBack();
        }
        else if (e.VirtualKey == VirtualKey.Right)
        {
            e.Handled = TryGoForward();
        }
    }
}
```

Handle system back requests

Windows devices provide various ways that the system can pass a back navigation request to your app. Some common ways are the B button on a gamepad, the Windows key + Backspace key shortcut, or the system back button in Tablet Mode; the exact options available depend on the device.

You can support system provided back requests from hardware and software system back keys by registering a listener for the SystemNavigationManager.BackRequested event.

Here's the code added to the `App` class to support system provided back requests. (This assumes that the previous code to support the back button has already been added.) You can see all the `App` code together at the end of the Code examples section.

C#

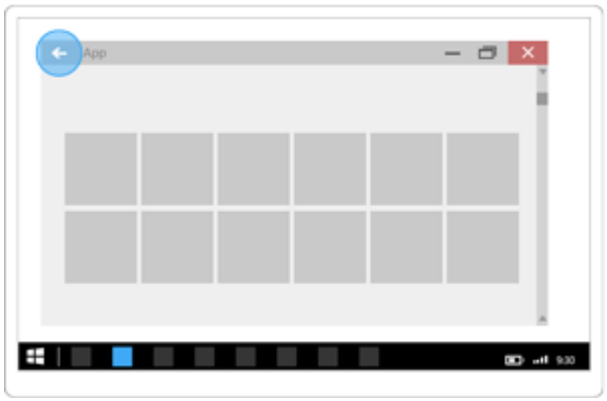
```
// App.xaml.cs
// Add event handler in OnLaunced.
```

```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    // ...
    // Do not repeat app initialization when the Window already has content,
    // just ensure that the window is active
    if (rootFrame == null)
    {
        // ...
        // Add support for accelerator keys.
        // ... (Previously added code.)

        // Add support for system back requests.
        SystemNavigationManager.GetForCurrentView().BackRequested
            += System_BackRequested;

        // ...

    }
}

// ...
// Handle system back requests.
private void System_BackRequested(object sender, BackRequestedEventArgs e)
{
    if (!e.Handled)
    {
        e.Handled = TryGoBack();
    }
}
```

System back behavior for backward compatibility

Previously, UWP apps used SystemNavigationManager.AppViewBackButtonVisibility to show or hide a system back button for backwards navigation. (This button raises a SystemNavigationManager.BackRequested event.) This API will continue to be supported to ensure backward compatibility, but we no longer recommend using the back button exposed by `AppViewBackButtonVisibility`. Instead, you should provide your own in-app back button as described in this article.

If you continue to use AppViewBackButtonVisibility, then the system UI renders the system back button inside the title bar. (The appearance and user interactions for the back button are unchanged from previous builds.)



Handle mouse navigation buttons

Some mice provide hardware navigation buttons for forward and back navigation. You can support these mouse buttons by handling the CoreWindow.PointerPressed event and checking for IsXButton1Pressed (back) or IsXButton2Pressed (forward).

Here's the code added to the `App` class to support mouse button navigation. (This assumes that the previous code to support the back button has already been added.) You can see all the `App` code together at the end of the Code examples section.

C#

```
// App.xaml.cs
// Add event handler in OnLaunced.
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    // ...
    // Do not repeat app initialization when the Window already has content,
    // just ensure that the window is active
    if (rootFrame == null)
    {
        // ...
        // Add support for system back requests.
        // ... (Previously added code.)

        // Add support for mouse navigation buttons.
        Window.Current.CoreWindow.PointerPressed +=
CoreWindow_PointerPressed;

        // ...

    }
}

// ...

// Handle mouse back button.
private void CoreWindow_PointerPressed(CoreWindow sender, PointerEventArgs
e)
```

```
{
    // For this event, e.Handled arrives as 'true'.
    if (e.CurrentPoint.Properties.IsXButton1Pressed)
    {
        e.Handled = !TryGoBack();
    }
    else if (e.CurrentPoint.Properties.IsXButton2Pressed)
    {
        e.Handled = !TryGoForward();
    }
}
```

All code added to App class

C#

```
// App.xaml.cs
//
// (Add event handlers in OnLaunched override.)
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
    // ...
    // Do not repeat app initialization when the Window already has content,
    // just ensure that the window is active
    if (rootFrame == null)
    {
        // ...
        // rootFrame.NavigationFailed += OnNavigationFailed;

        // Add support for accelerator keys.
        // Listen to the window directly so the app responds
        // to accelerator keys regardless of which element has focus.
        Window.Current.CoreWindow.Dispatcher.AcceleratorKeyActivated +=
            CoreDispatcher_AcceleratorKeyActivated;

        // Add support for system back requests.
        SystemNavigationManager.GetForCurrentView().BackRequested
            += System_BackRequested;

        // Add support for mouse navigation buttons.
        Window.Current.CoreWindow.PointerPressed +=
CoreWindow_PointerPressed;

        // ...

    }
}

// ...

// (Add these methods to the App class.)
public static bool TryGoBack()
```

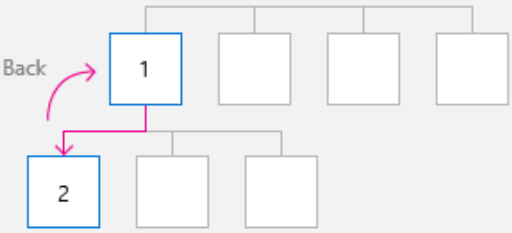
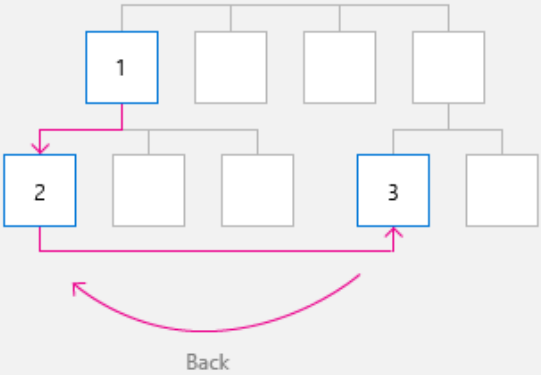
```
{
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame.CanGoBack)
    {
        rootFrame.GoBack();
        return true;
    }
    return false;
}

// Perform forward navigation if possible.
private bool TryGoForward()
{
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame.CanGoForward)
    {
        rootFrame.GoForward();
        return true;
    }
    return false;
}

// Invoked on every keystroke, including system keys such as Alt key
combinations.
// Used to detect keyboard navigation between pages even when the page
itself
// doesn't have focus.
private void CoreDispatcher_AcceleratorKeyActivated(CoreDispatcher sender,
AcceleratorKeyEventArgs e)
{
    // When Alt+Left are pressed navigate back.
    // When Alt+Right are pressed navigate forward.
    if (e.EventType == CoreAcceleratorKeyEventType.SystemKeyDown
        && (e.VirtualKey == VirtualKey.Left || e.VirtualKey ==
VirtualKey.Right)
        && e.KeyStatus.IsMenuKeyDown == true
        && !e.Handled)
    {
        if (e.VirtualKey == VirtualKey.Left)
        {
            e.Handled = TryGoBack();
        }
        else if (e.VirtualKey == VirtualKey.Right)
        {
            e.Handled = TryGoForward();
        }
    }
}

// Handle system back requests.
private void System_BackRequested(object sender, BackRequestedEventArgs e)
{
    if (!e.Handled)
    {
        e.Handled = TryGoBack();
```

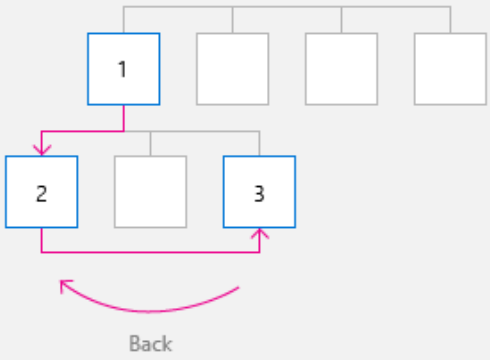
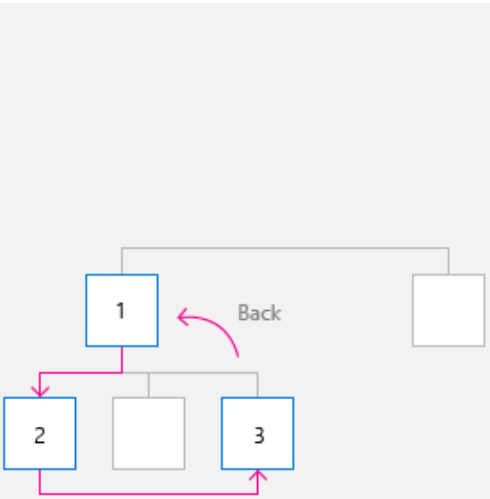
```
    }
}

// Handle mouse back button.
private void CoreWindow_PointerPressed(CoreWindow sender, PointerEventArgs
e)
{
    // For this event, e.Handled arrives as 'true'.
    if (e.CurrentPoint.Properties.IsXButton1Pressed)
    {
        e.Handled = !TryGoBack();
    }
    else if (e.CurrentPoint.Properties.IsXButton2Pressed)
    {
        e.Handled = !TryGoForward();
    }
}
```

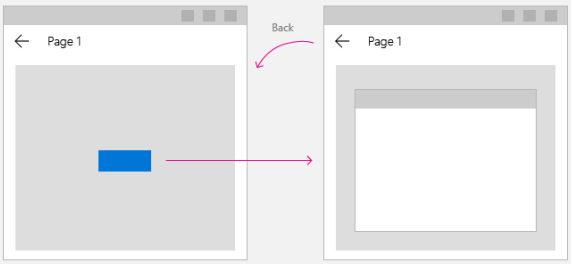
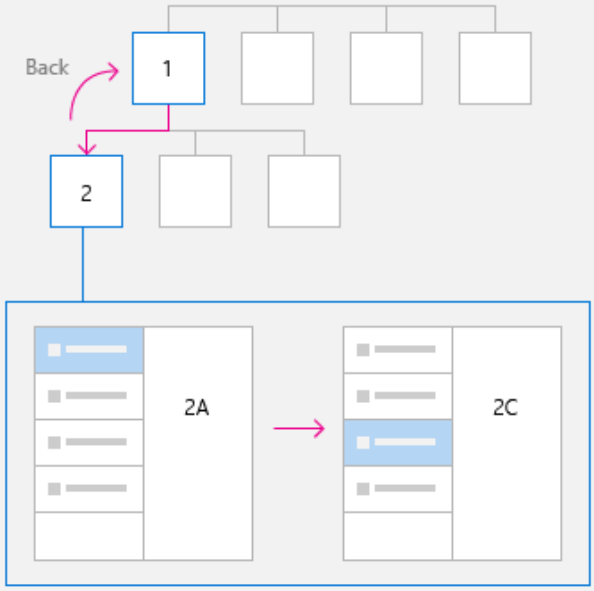
Guidelines for custom back navigation behavior

If you choose to provide your own back stack navigation, the experience should be consistent with other apps. We recommend that you follow the following patterns for navigation actions:

Navigation action	Add to navigation history?
Page to page, different peer groups	Yes In this illustration, the user navigates from level 1 of the app to level 2, crossing peer groups, so the navigation is added to the navigation history.

Navigation action	Add to navigation history?
	 In the next illustration, the user navigates between two peer groups at the same level, again crossing peer groups, so the navigation is added to the navigation history. 
Page to page, same peer group, no on-screen navigation element The user navigates from one page to another with the same peer group. There is no on-screen navigation element (such as NavigationView)	**Yes** In the following illustration, the user navigates between two pages in the same peer group, and the navigation should be added to the navigation history.

Navigation action	Add to navigation history?
that provides direct navigation to both pages.	
Page to page, same peer group, with an on-screen navigation element The user navigates from one page to another in the same peer group. Both pages are shown in the same navigation element, such as NavigationView.	**It depends** Yes, add to the navigation history, with two notable exceptions. If you expect users of your app to switch between pages in the peer group frequently, or if you wish to preserve the navigational hierarchy, then do not add to the navigation history. In this case, when the user presses back, go back to the last page before the user navigated to the current peer group. 
Show a transient UI	**No**

Navigation action	Add to navigation history?
The app displays a pop-up or child window, such as a dialog, splash screen, or on-screen keyboard, or the app enters a special mode, such as multiple selection mode.	When the user presses the back button, dismiss the transient UI (hide the on-screen keyboard, cancel the dialog, etc) and return to the page that spawned the transient UI. 
Enumerate items The app displays content for an on-screen item, such as the details for the selected item in list/details list.	**No** Enumerating items is similar to navigating within a peer group. When the user presses back, navigate to the page that preceded the current page that has the item enumeration. 

Resuming

When the user switches to another app and returns to your app, we recommend returning to the last page in the navigation history.

Related articles

- Navigation basics

Controls for Windows apps

Article • 03/03/2025

In Windows app development, a *control* is a UI element that displays content or enables interaction. Controls are the building blocks of the user interface. A *pattern* is a recipe for combining several controls to make something new.

We provide 45+ controls for you to use, ranging from simple buttons to powerful data controls like the grid view. These controls are a part of the Fluent Design System and can help you create a bold, scalable UI that looks great on all devices and screen sizes.

The articles in this section provide design guidance and coding instructions for adding controls & patterns to your Windows app.

General instructions and code examples

The topics highlighted here provide instructions and code examples for adding and styling controls in XAML and C#.

Add controls and handle events

There are 3 key steps to adding controls to your app: Add a control to your app UI, set properties on the control, and add code to the control's event handlers so that it does something.

Styling controls

You can customize the appearance of your apps in many ways by using the XAML framework. Styles let you set control properties and reuse those settings for a consistent appearance across multiple controls.

Get WinUI

Some controls are only available in WinUI, a NuGet package that contains new controls and UI features. To get it, see the WinUI overview and installation instructions.

Index of controls

The following table lists the common Windows app controls and patterns along with those that are exclusive to WinUI.

Common Windows app controls

WinUI only

Auto-suggest box

Button

Animated icon

Calendar date picker

Calendar view

Animated visual player (see Lottie)

Checkbox

Combo box

Breadcrumb bar

Command bar

Contact card

Color picker

Content dialog

Content link

Command bar flyout

Context menu

Date picker

Drop down button

Dialogs and flyouts

Flip view

Expander

Flyout

Forms (pattern)

Info bar

Grid view

Hyperlink

Menu bar

Hyperlink button

Images and image brushes

Navigation view

Inking controls

List/details (pattern)

Number box

List view

Map control

Parallax view

Media playback

Menu flyout

Person picture

Password box

Repeat button

Pips pager

Rich edit box

Rich text block

Progress bar

Scroll viewer

Semantic zoom

Progress ring

Shapes

Slider

Radio button

Split view

Text block

Rating control

Text box

Time picker

Split button

Toggle switch

Toggle button

Swipe control

Toggle split button

Tooltips

Tab view

Web view

Teaching tip

Tree view

Two-pane view

WinUI Gallery

Get the *WinUI Gallery* apps from the Microsoft Store to see XAML controls and the Fluent Design System in action. The **WinUI 3 Gallery** and **WinUI 2 Gallery** apps include interactive examples of most WinUI 3 and WinUI 2 controls, features, and functionality. The apps are an interactive companion to this website. When you have them installed, you can use links on individual control pages to launch the app and see the control in action.

- ✓ Get the **WinUI 3 Gallery** and the **WinUI 2 Gallery** from the Microsoft Store.
- ✓ Get the source code for both from GitHub (use the *main* branch for WinUI 3 and the *winui2* branch for WinUI 2).

Additional controls

Additional controls for Windows development are available from companies such as Telerik, SyncFusion, DevExpress, Infragistics, ComponentOne, and ActiPro. These controls provide additional support for enterprise and .NET developers by augmenting the standard system controls with custom controls and services.

Intro to controls and patterns

Article • 06/09/2022

In Windows app development, a *control* is a UI element that displays content or enables interaction. You create the UI for your app by using controls such as buttons, text boxes, and combo boxes to display data and get user input.

> **Important APIs**: Windows.UI.Xaml.Controls namespace

A *pattern* is a recipe for modifying a control or combining several controls to make something new. For example, the list/details pattern is a way that you can use a SplitView control for app navigation. Similarly, you can customize the template of a NavigationView control to implement the tab pattern.

In many cases, you can use a control as-is. But XAML controls separate function from structure and appearance, so you can make various levels of modification to make them fit your needs. In the Style section, you can learn how to use XAML styles and control templates to modify a control.

In this section, we provide guidance for each of the XAML controls you can use to build your app UI. To start, this article shows you how to add controls to your app. There are 3 key steps to using controls to your app:

- Add a control to your app UI.
- Set properties on the control, such as width, height, or foreground color.
- Add code to the control's event handlers so that it does something.

Add a control

You can add a control to an app in several ways:

- Use a design tool like Blend for Visual Studio or the Microsoft Visual Studio Extensible Application Markup Language (XAML) designer.
- Add the control to the XAML markup in the Visual Studio XAML editor.
- Add the control in code. Controls that you add in code are visible when the app runs, but are not visible in the Visual Studio XAML designer.

In Visual Studio, when you add and manipulate controls in your app, you can use many of the program's features, including the Toolbox, XAML designer, XAML editor, and the Properties window.

The Visual Studio Toolbox displays many of the controls that you can use in your app. To add a control to your app, double-click it in the Toolbox. For example, when you double-click the TextBox control, this XAML is added to the XAML view.

XAML

```
<TextBox HorizontalAlignment="Left" Text="TextBox" VerticalAlignment="Top"/>
```

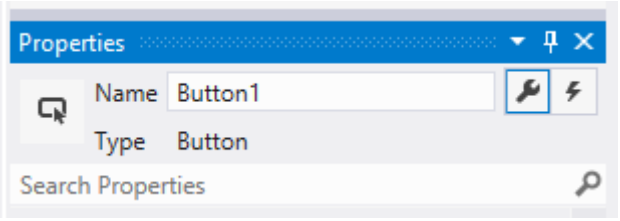
You can also drag the control from the Toolbox to the XAML designer.

Set the name of a control

To work with a control in code, you set its x:Name attribute and reference it by name in your code. You can set the name in the Visual Studio Properties window or in XAML. Here's how to set the name of the currently selected control by using the Name text box at the top of the Properties window.

To name a control

1. Select the element to name.
2. In the Properties panel, type a name into the Name text box.
3. Press Enter to commit the name.



Here's how to set the name of a control in the XAML editor by adding the x:Name attribute.

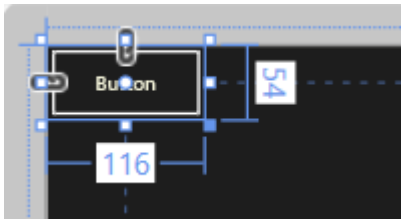
XAML

```
<Button x:Name="Button1" Content="Button"/>
```

Set the control properties

You use properties to specify the appearance, content, and other attributes of controls. When you add a control using a design tool, some properties that control size, position, and content might be set for you by Visual Studio. You can change some properties,

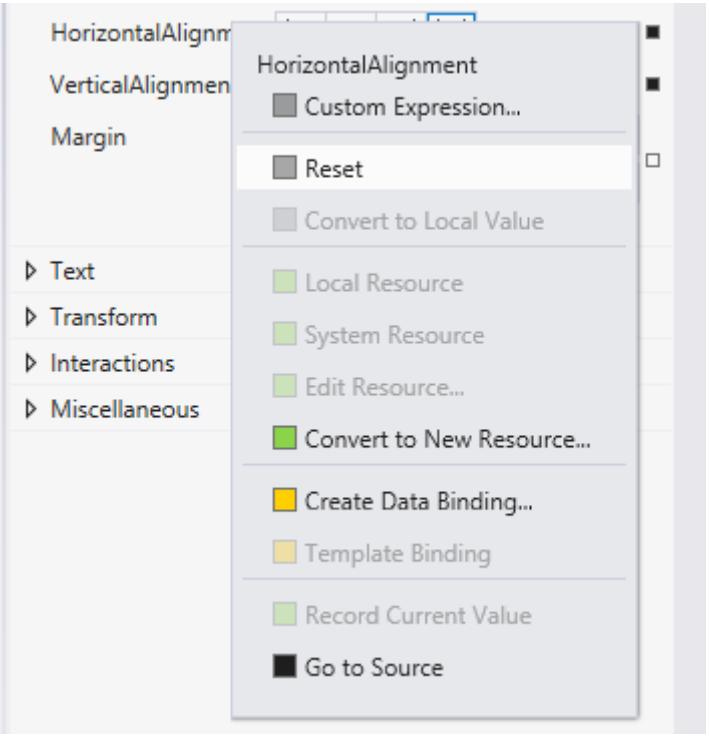
such as Width, Height or Margin, by selecting and manipulating the control in the Design view. This illustration shows some of the resizing tools available in Design view.



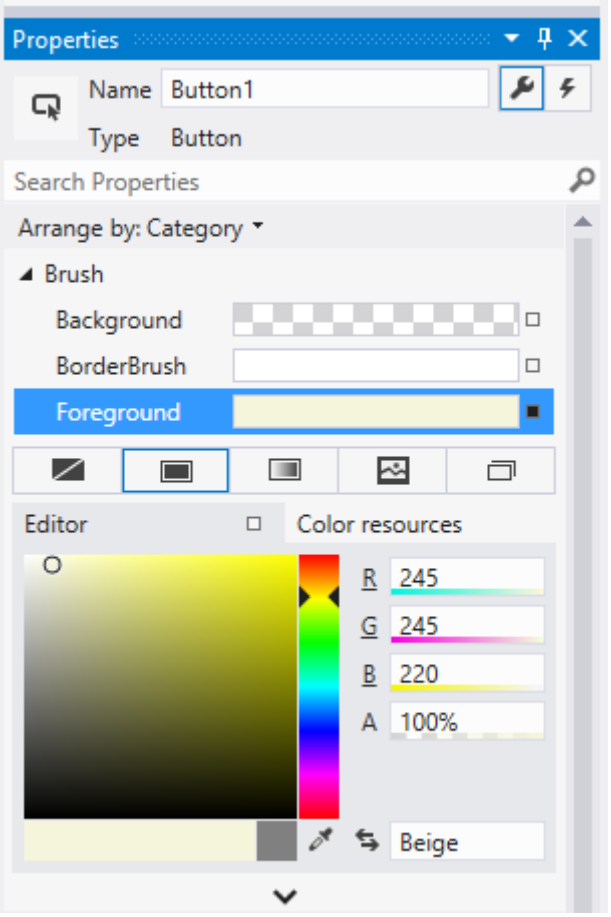
You might want to let the control be sized and positioned automatically. In this case, you can reset the size and position properties that Visual Studio set for you.

To reset a property

1. In the Properties panel, click the property marker next to the property value. The property menu opens.
2. In the property menu, click Reset.



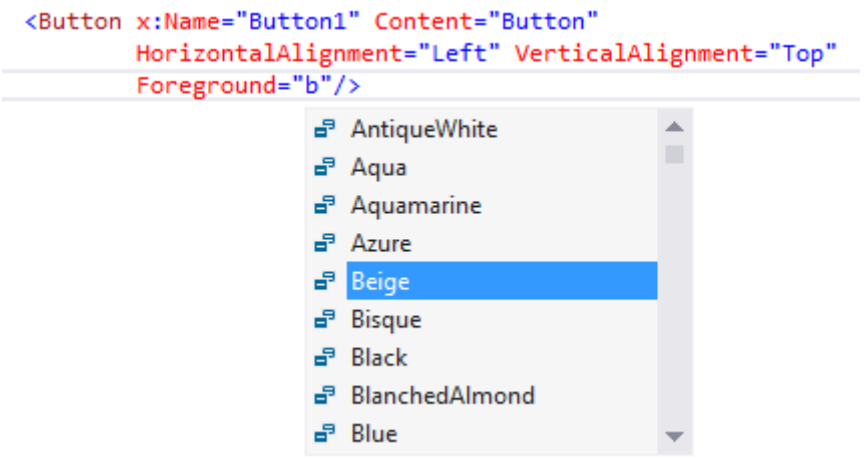
You can set control properties in the Properties window, in XAML, or in code. For example, to change the foreground color for a Button, you set the control's Foreground property. This illustration shows how to set the Foreground property by using the color picker in the Properties window.



Here's how to set the Foreground property in the XAML editor. Notice the Visual Studio IntelliSense window that opens to help you with the syntax.

```
<Button x:Name="Button1" Content="Button"
        HorizontalAlignment="Left" VerticalAlignment="Top"
        For/>
```

FontSize
FontStretch
FontStyle
FontWeight
Foreground
GotFocus
Grid
Height
Holding



Here's the resulting XAML after you set the Foreground property.

XAML

```
<Button x:Name="Button1" Content="Button"
        HorizontalAlignment="Left" VerticalAlignment="Top"
        Foreground="Beige"/>
```

Here's how to set the Foreground property in code.

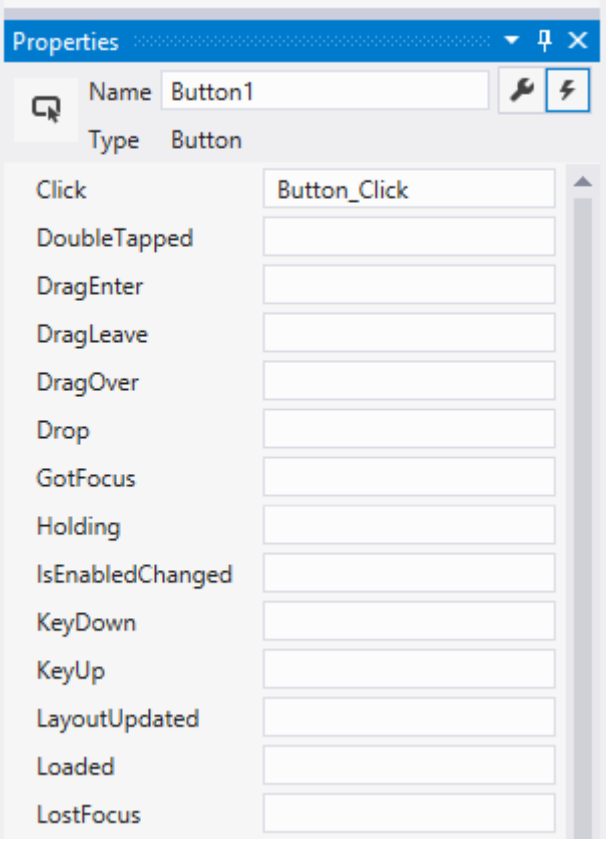
C#

```
Button1.Foreground = new SolidColorBrush(Windows.UI.Colors.Beige);
```

Create an event handler

Each control has events that enable you to respond to actions from your user or other changes in your app. For example, a Button control has a Click event that is raised when a user clicks the Button. You create a method, called an event handler, to handle the event. You can associate a control's event with an event handler method in the Properties window, in XAML, or in code. For more info about events, see Events and routed events overview.

To create an event handler, select the control and then click the Events tab at the top of the Properties window. The Properties window lists all of the events available for that control. Here are some of the events for a Button.



To create an event handler with the default name, double-click the text box next to the event name in the Properties window. To create an event handler with a custom name, type the name of your choice into the text box and press enter. The event handler is created and the code-behind file is opened in the code editor. The event handler method has 2 parameters. The first is `sender`, which is a reference to the object where the handler is attached. The `sender` parameter is an **Object** type. You typically cast `sender` to a more precise type if you expect to check or change the state on the `sender` object itself. Based on your own app design, you expect a type that is safe to cast the `sender` to, based on where the handler is attached. The second value is event data, which generally appears in signatures as the `e` or `args` parameter.

Here's code that handles the Click event of a Button named `Button1`. When you click the button, the Foreground property of the Button you clicked is set to blue.

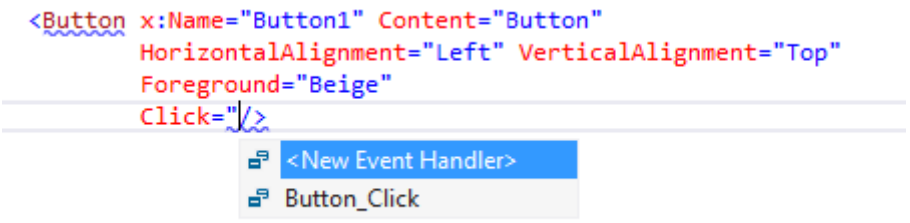
C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    Button b = (Button)sender;
    b.Foreground = new SolidColorBrush(Windows.UI.Colors.Blue);
}
```

You can also associate an event handler in XAML. In the XAML editor, type in the event name that you want to handle. Visual Studio shows an IntelliSense window when you begin typing. After you specify the event, you can double-click `<New Event Handler>` in

the IntelliSense window to create a new event handler with the default name, or select an existing event handler from the list.

Here's the IntelliSense window that appears. It helps you create a new event handler or select an existing event handler.



This example shows how to associate a Click event with an event handler named Button_Click in XAML.

XAML

```
<Button Name="Button1" Content="Button" Click="Button_Click"/>
```

You can also associate an event with its event handler in the code-behind. Here's how to associate an event handler in code.

C#

```
Button1.Click += new RoutedEventHandler(Button_Click);
```

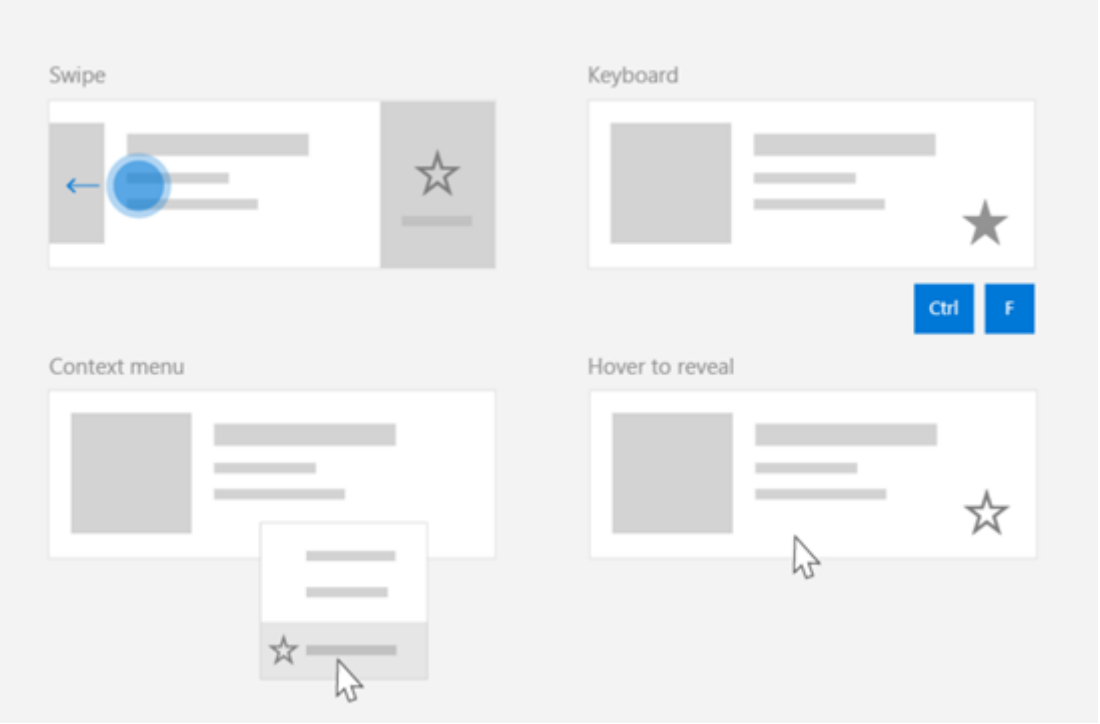
Related topics

- Index of controls by function
- Windows.UI.Xaml.Controls namespace
- Layout
- Style
- Usability

Commanding in Windows apps using StandardUICommand, XamlUICommand, and ICommand

Article • 01/03/2023

In this topic, we describe commanding in Windows applications. Specifically, we discuss how you can use the XamlUICommand and StandardUICommand classes (along with the ICommand interface) to share and manage commands across various control types, regardless of the device and input type being used.



Share commands across various controls, regardless of device and input type

Important APIs

- Windows.UI.Xaml.Input.ICommand and System.Windows.Input.ICommand
- XamlUICommand
- StandardUICommand

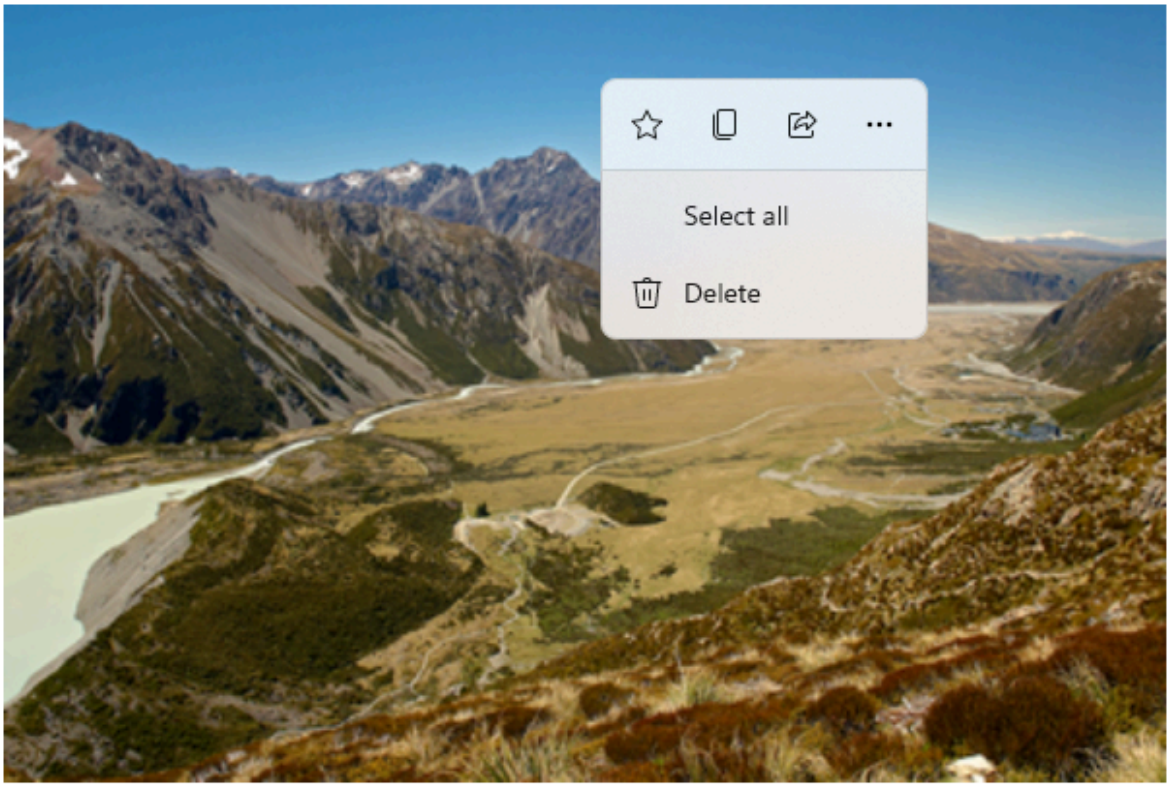
Overview

Commands can be invoked directly through UI interactions like clicking a button or selecting an item from a context menu. They can also be invoked indirectly through an input device such as a keyboard accelerator, gesture, speech recognition, or an

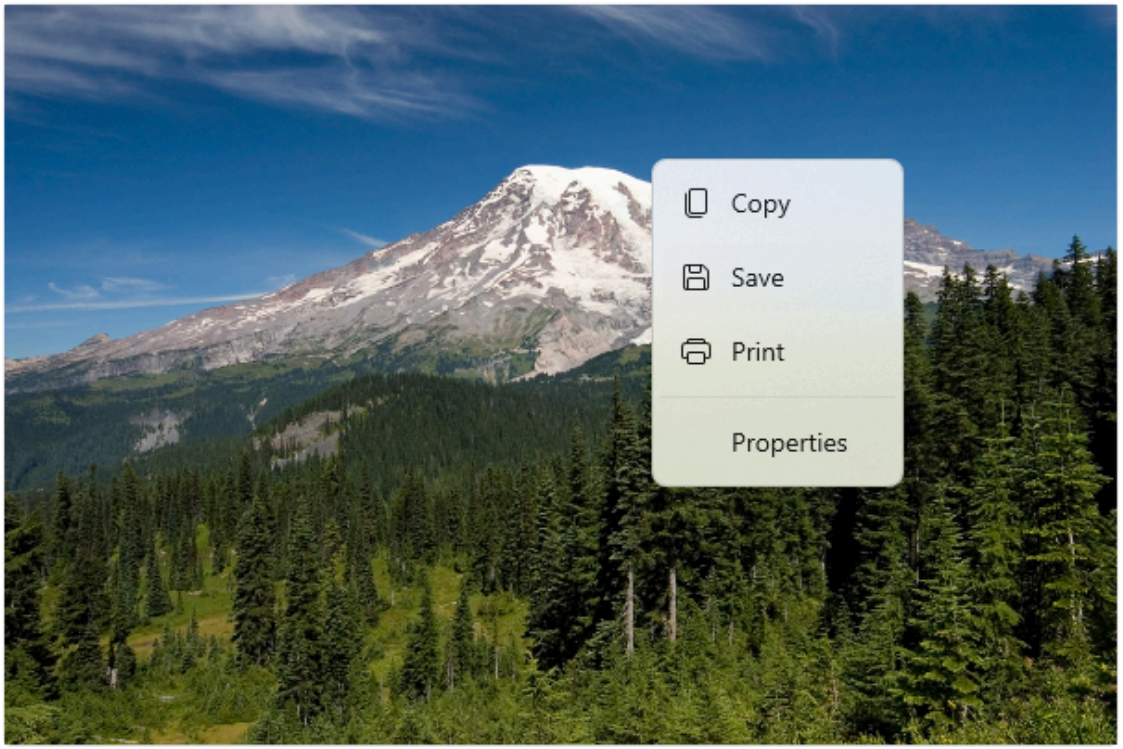
automation/accessibility tool. Once invoked, the command can then be handled by a control (text navigation in an edit control), a window (back navigation), or the application (exit).

Commands can operate on a specific context within your app, such as deleting text or undoing an action, or they can be context-free, such as muting audio or adjusting brightness.

The following image shows two command interfaces (a CommandBar and a floating contextual CommandBarFlyout) that share some of the same commands.



Command bar



Context menu in the Microsoft Photos gallery

Command interactions

Due to the variety of devices, input types, and UI surfaces that can affect how a command is invoked, we recommend exposing your commands through as many commanding surfaces as possible. These can include a combination of Swipe, MenuBar, CommandBar, CommandBarFlyout, and traditional context menu.

For critical commands, use input-specific accelerators. Input accelerators let a user perform actions more quickly depending on the input device they're using.

Here are some common input accelerators for various input types:

- **Pointer** - Mouse & Pen hover buttons
- **Keyboard** - Shortcuts (access keys and accelerator keys)
- **Touch** - Swipe
- **Touch** - Pull to refresh data

You must consider the input type and user experiences to make your application's functionality universally accessible. For example, collections (especially user editable ones) typically include a variety of specific commands that are performed quite differently depending on the input device.

The following table shows some typical collection commands and ways to expose those commands.

Expand table

Command	Input-agnostic	Mouse accelerator	Keyboard accelerator	Touch accelerator
Delete item	Context menu	Hover button	DEL key	Swipe to delete
Flag item	Context menu	Hover button	Ctrl+Shift+G	Swipe to flag
Refresh data	Context menu	N/A	F5 key	Pull to refresh
Favorite an item	Context menu	Hover button	F, Ctrl+S	Swipe to favorite

Always provide a context menu We recommend including all relevant contextual commands in a traditional context menu or CommandBarFlyout, as both are supported for all input types. For example, if a command is exposed only during a pointer hover event, it cannot be used on a touch-only device.

Commands in Windows applications

There are a few ways you can share and manage commanding experiences in a Windows application. You can define event handlers for standard interactions, such as Click, in code-behind (this can be quite inefficient, depending on the complexity of your UI), you can bind event listener for standard interactions to a shared handler, or you can bind the control's Command property to an ICommand implementation that describes the command logic.

To provide rich and comprehensive user experiences across command surfaces efficiently and with minimal code duplication, we recommend using the command binding features described in this topic(for standard event handling, see the individual event topics).

To bind a control to a shared command resource, you can implement the ICommand interfaces yourself, or you can build your command from either the XamlUICommand base class or one of the platform commands defined by the StandardUICommand derived class.

- The ICommand interface (Windows.UI.Xaml.Input.ICommand or System.Windows.Input.ICommand) lets you create fully customized, reusable commands across your app.

- XamlUICommand also provides this capability but simplifies development by exposing a set of built-in command properties such as the command behavior, keyboard shortcuts (access key and accelerator key), icon, label, and description.
- StandardUICommand simplifies things further by letting you choose from a set of standard platform commands with predefined properties.

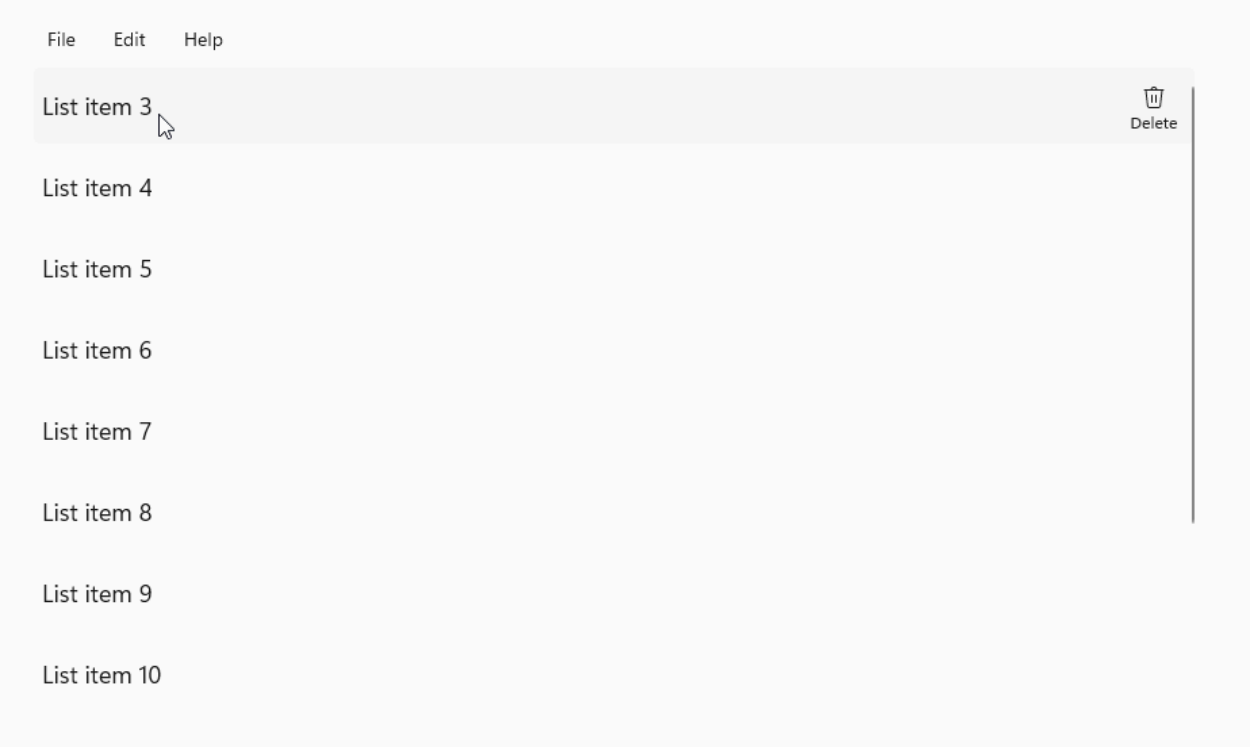
> **Important**
>
> In UWP applications, commands are implementations of either the **Windows.UI.Xaml.Input.ICommand** (C++) or the **System.Windows.Input.ICommand** (C#) interface, depending on your chosen language framework.

Command experiences using the StandardUICommand class

Derived from XamlUICommand (derived from Windows.UI.Xaml.Input.ICommand for C++ or System.Windows.Input.ICommand for C#), the StandardUICommand class exposes a set of standard platform commands with pre-defined properties such as icon, keyboard accelerator, and description.

A StandardUICommand provides a quick and consistent way to define common commands such as `Save` or `Delete`. All you have to do is provide the execute and canExecute functions.

Example



StandardUICommandSample

Expand table

Download the code for this example
UWP commanding sample (StandardUICommand)

In this example, we show how to enhance a basic ListView with a Delete item command implemented through the StandardUICommand class, while optimizing the user experience for a variety of input types using a MenuBar, Swipe control, hover buttons, and context menu.

> **Note**
>
> This sample requires the Microsoft.UI.Xaml.Controls NuGet package, a part of WinUI 2.

Xaml:

The sample UI includes a ListView of five items. The Delete StandardUICommand is bound to a MenuBarItem, a SwipeItem, an AppBarButton, and ContextFlyout menu.

XAML

```
<Page
    x:Class="StandardUICommandSample.MainPage"
```

```
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:StandardUICommandSample"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:muxcontrols="using:Microsoft.UI.Xaml.Controls"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <Page.Resources>
        <Style x:Key="HorizontalSwipe"
               TargetType="ListViewItem"
               BasedOn="{StaticResource ListViewItemRevealStyle}">
            <Setter Property="Height" Value="60"/>
            <Setter Property="Padding" Value="0"/>
            <Setter Property="HorizontalContentAlignment" Value="Stretch"/>
            <Setter Property="VerticalContentAlignment" Value="Stretch"/>
            <Setter Property="BorderThickness" Value="0"/>
        </Style>
    </Page.Resources>

    <Grid Loaded="ControlExample_Loaded">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <StackPanel Grid.Row="0"
                    Padding="10"
                    BorderThickness="0,0,0,1"
                    BorderBrush="LightBlue"
                    Background="AliceBlue">
            <TextBlock Style="{StaticResource HeaderTextBlockStyle}">
                StandardUICommand sample
            </TextBlock>
            <TextBlock Style="{StaticResource SubtitleTextBlockStyle}"
Margin="0,0,0,10">
                This sample shows how to use the StandardUICommand class to
                share a platform command and consistent user experiences
                across various controls.
            </TextBlock>
            <TextBlock Style="{StaticResource SubtitleTextBlockStyle}"
Margin="0,0,0,0">
                Specifically, we define a standard delete command and add it
                to a variety of command surfaces, all of which share a
common
                icon, label, keyboard accelerator, and description.
            </TextBlock>
        </StackPanel>

        <muxcontrols:MenuBar Grid.Row="1" Padding="10">
            <muxcontrols:MenuBarItem Title="File">
            </muxcontrols:MenuBarItem>
            <muxcontrols:MenuBarItem Title="Edit">
```

```
                <MenuFlyoutItem x:Name="DeleteFlyoutItem"/>
            </muxcontrols:MenuBarItem>
            <muxcontrols:MenuBarItem Title="Help">
            </muxcontrols:MenuBarItem>
        </muxcontrols:MenuBar>

        <ListView x:Name="ListViewRight" Grid.Row="2"
                  Loaded="ListView_Loaded"
                  IsItemClickEnabled="True"
                  SelectionMode="Single"
                  SelectionChanged="ListView_SelectionChanged"
                  ItemContainerStyle="{StaticResource HorizontalSwipe}">
            <ListView.ItemTemplate>
                <DataTemplate x:DataType="local:ListItemData">
                    <UserControl
PointerEntered="ListViewSwipeContainer_PointerEntered"

PointerExited="ListViewSwipeContainer_PointerExited">
                        <UserControl.ContextFlyout>
                            <MenuFlyout>
                                <MenuFlyoutItem
                                    Command="{x:Bind Command}"
                                    CommandParameter="{x:Bind Text}" />
                            </MenuFlyout>
                        </UserControl.ContextFlyout>
                        <Grid AutomationProperties.Name="{x:Bind Text}">
                            <VisualStateManager.VisualStateGroups>
                                <VisualStateGroup x:Name="HoveringStates">
                                    <VisualState x:Name="HoverButtonsHidden"
/>
                                    <VisualState x:Name="HoverButtonsShown">
                                        <VisualState.Setters>
                                            <Setter
Target="HoverButton.Visibility"
                                                    Value="Visible" />
                                        </VisualState.Setters>
                                    </VisualState>
                                </VisualStateGroup>
                            </VisualStateManager.VisualStateGroups>
                            <SwipeControl x:Name="ListViewSwipeContainer" >
                                <SwipeControl.RightItems>
                                    <SwipeItems Mode="Execute">
                                        <SwipeItem x:Name="DeleteSwipeItem"
                                                   Background="Red"
                                                   Command="{x:Bind
Command}"
                                                   CommandParameter="{x:Bind
Text}"/>
                                    </SwipeItems>
                                </SwipeControl.RightItems>
                                <Grid VerticalAlignment="Center">
                                    <TextBlock Text="{x:Bind Text}"
                                               Margin="10"
                                               FontSize="18"
                                               HorizontalAlignment="Left"
```

```
                                                VerticalAlignment="Center"/>
                                    <AppBarButton x:Name="HoverButton"
                                                  IsTabStop="False"

HorizontalAlignment="Right"
                                                  Visibility="Collapsed"
                                                  Command="{x:Bind Command}"
                                                  CommandParameter="{x:Bind
Text}"/>
                                </Grid>
                            </SwipeControl>
                        </Grid>
                    </UserControl>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </Grid>
</Page>
```

Code-behind

1. First, we define a `ListItemData` class that contains a text string and ICommand for each ListViewItem in our ListView.

C#

```
public class ListItemData
{
    public String Text { get; set; }
    public ICommand Command { get; set; }
}
```

2. In the MainPage class, we define a collection of `ListItemData` objects for the DataTemplate of the ListView ItemTemplate. We then populate it with an initial collection of five items (with text and associated StandardUICommand Delete).

C#

```
/// <summary>
/// ListView item collection.
/// </summary>
ObservableCollection<ListItemData> collection =
    new ObservableCollection<ListItemData>();

/// <summary>
/// Handler for the layout Grid control load event.
/// </summary>
/// <param name="sender">Source of the control loaded event</param>
/// <param name="e">Event args for the loaded event</param>
private void ControlExample_Loaded(object sender, RoutedEventArgs e)
```

```
{
    // Create the standard Delete command.
    var deleteCommand = new StandardUICommand(StandardUICommandKind.Delete);
    deleteCommand.ExecuteRequested += DeleteCommand_ExecuteRequested;

    DeleteFlyoutItem.Command = deleteCommand;

    for (var i = 0; i < 5; i++)
    {
        collection.Add(
            new ListItemData {
                Text = "List item " + i.ToString(),
                Command = deleteCommand });
    }
}

/// <summary>
/// Handler for the ListView control load event.
/// </summary>
/// <param name="sender">Source of the control loaded event</param>
/// <param name="e">Event args for the loaded event</param>
private void ListView_Loaded(object sender, RoutedEventArgs e)
{
    var listView = (ListView)sender;
    // Populate the ListView with the item collection.
    listView.ItemsSource = collection;
}
```

3. Next, we define the ICommand ExecuteRequested handler where we implement the item delete command.

C#

```
/// <summary>
/// Handler for the Delete command.
/// </summary>
/// <param name="sender">Source of the command event</param>
/// <param name="e">Event args for the command event</param>
private void DeleteCommand_ExecuteRequested(
    XamlUICommand sender, ExecuteRequestedEventArgs args)
{
    // If possible, remove specified item from collection.
    if (args.Parameter != null)
    {
        foreach (var i in collection)
        {
            if (i.Text == (args.Parameter as string))
            {
                collection.Remove(i);
                return;
            }
        }
    }
```

```
    if (ListViewRight.SelectedIndex != -1)
    {
        collection.RemoveAt(ListViewRight.SelectedIndex);
    }
}
```

4. Finally, we define handlers for various ListView events, including PointerEntered, PointerExited, and SelectionChanged events. The pointer event handlers are used to show or hide the Delete button for each item.

C#

```
/// <summary>
/// Handler for the ListView selection changed event.
/// </summary>
/// <param name="sender">Source of the selection changed event</param>
/// <param name="e">Event args for the selection changed event</param>
private void ListView_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (ListViewRight.SelectedIndex != -1)
    {
        var item = collection[ListViewRight.SelectedIndex];
    }
}

/// <summary>
/// Handler for the pointer entered event.
/// Displays the delete item "hover" buttons.
/// </summary>
/// <param name="sender">Source of the pointer entered event</param>
/// <param name="e">Event args for the pointer entered event</param>
private void ListViewSwipeContainer_PointerEntered(
    object sender, PointerRoutedEventArgs e)
{
    if (e.Pointer.PointerDeviceType ==
        Windows.Devices.Input.PointerDeviceType.Mouse ||
        e.Pointer.PointerDeviceType ==
        Windows.Devices.Input.PointerDeviceType.Pen)
    {
        VisualStateManager.GoToState(
            sender as Control, "HoverButtonsShown", true);
    }
}

/// <summary>
/// Handler for the pointer exited event.
/// Hides the delete item "hover" buttons.
/// </summary>
/// <param name="sender">Source of the pointer exited event</param>
/// <param name="e">Event args for the pointer exited event</param>

private void ListViewSwipeContainer_PointerExited(
```

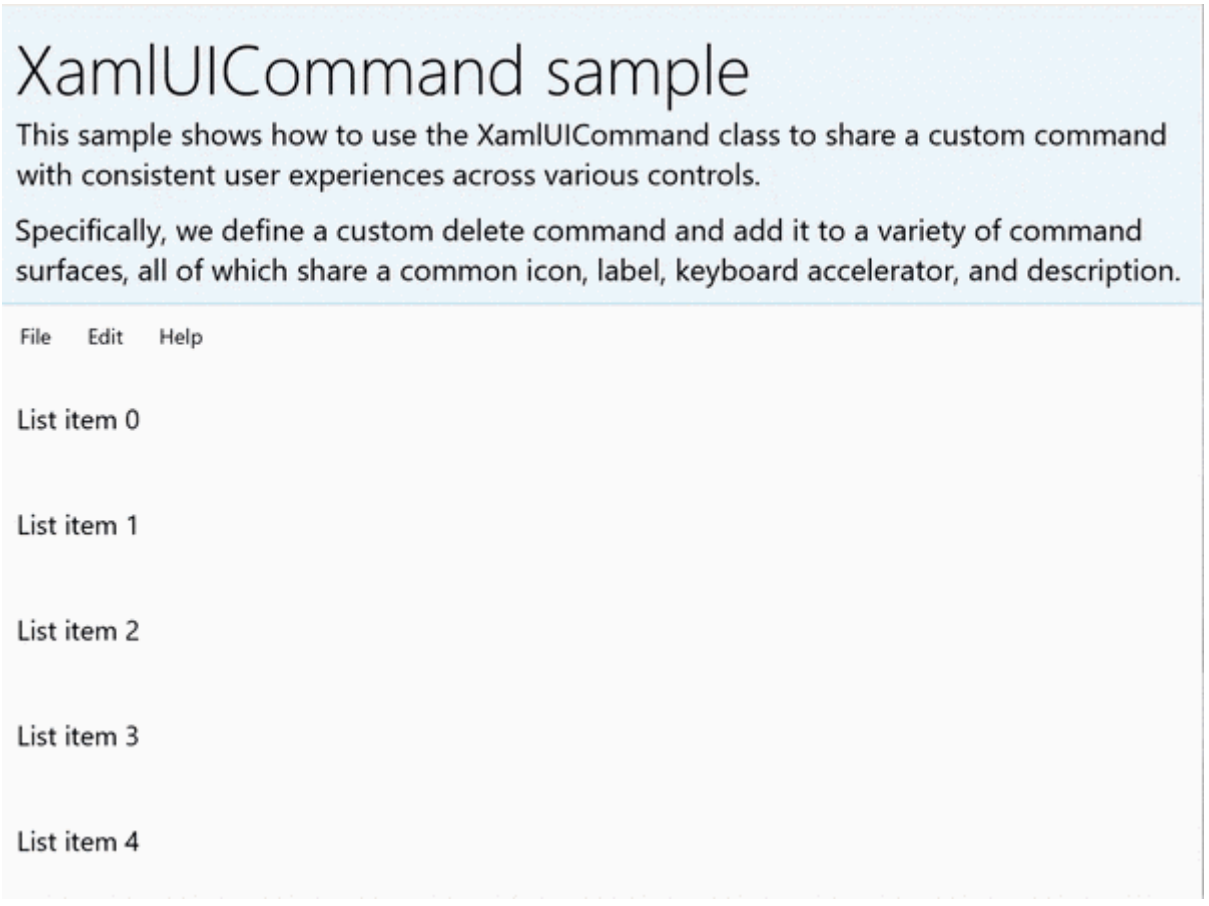
```
    object sender, PointerRoutedEventArgs e)
{
    VisualStateManager.GoToState(
        sender as Control, "HoverButtonsHidden", true);
}
```

Command experiences using the XamlUICommand class

If you need to create a command that isn't defined by the StandardUICommand class, or you'd like more control over the command appearance, the XamlUICommand class derives from the ICommand interface, adding various UI properties (such as an icon, label, description, and keyboard shortcuts), methods, and events to quickly define the UI and behavior of a custom command.

XamlUICommand lets you specify UI through the control binding, such as an icon, label, description, and keyboard shortcuts (both an access key and a keyboard accelerator), without setting the individual properties.

Example



XamlUICommandSample

Expand table

Download the code for this example
UWP commanding sample (XamlUICommand)

This example shares the Delete functionality of the previous StandardUICommand example, but shows how the XamlUICommand class lets you define a custom delete command with your own font icon, label, keyboard accelerator, and description. Like the StandardUICommand example, we enhance a basic ListView with a Delete item command implemented through the XamlUICommand class, while optimizing the user experience for a variety of input types using a MenuBar, Swipe control, hover buttons, and context menu.

Many platform controls use the XamlUICommand properties under the covers, just like our StandardUICommand example in the previous section.

> **Note**
>
> This sample requires the Microsoft.UI.Xaml.Controls NuGet package, a part of WinUI 2.

Xaml:

The sample UI includes a ListView of five items. The custom XamlUICommand CustomXamlUICommand is bound to a MenuBarItem, a SwipeItem, an AppBarButton, and ContextFlyout menu.

XAML

```
<Page
    x:Class="XamlUICommand_Sample.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:XamlUICommand_Sample"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:muxcontrols="using:Microsoft.UI.Xaml.Controls"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <Page.Resources>
        <XamlUICommand x:Name="CustomXamlUICommand"
                       ExecuteRequested="DeleteCommand_ExecuteRequested"
                       Description="Custom XamlUICommand"
                       Label="Custom XamlUICommand">
            <XamlUICommand.IconSource>
```

```
                <FontIconSource FontFamily="Wingdings" Glyph="&#x4D;"/>
            </XamlUICommand.IconSource>
            <XamlUICommand.KeyboardAccelerators>
                <KeyboardAccelerator Key="D" Modifiers="Control"/>
            </XamlUICommand.KeyboardAccelerators>
        </XamlUICommand>

        <Style x:Key="HorizontalSwipe"
               TargetType="ListViewItem"
               BasedOn="{StaticResource ListViewItemRevealStyle}">
            <Setter Property="Height" Value="70"/>
            <Setter Property="Padding" Value="0"/>
            <Setter Property="HorizontalContentAlignment" Value="Stretch"/>
            <Setter Property="VerticalContentAlignment" Value="Stretch"/>
            <Setter Property="BorderThickness" Value="0"/>
        </Style>

    </Page.Resources>

    <Grid Loaded="ControlExample_Loaded" Name="MainGrid">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <StackPanel Grid.Row="0"
                    Padding="10"
                    BorderThickness="0,0,0,1"
                    BorderBrush="LightBlue"
                    Background="AliceBlue">
            <TextBlock Style="{StaticResource HeaderTextBlockStyle}">
                XamlUICommand sample
            </TextBlock>
            <TextBlock Style="{StaticResource SubtitleTextBlockStyle}"
Margin="0,0,0,10">
                This sample shows how to use the XamlUICommand class to
                share a custom command with consistent user experiences
                across various controls.
            </TextBlock>
            <TextBlock Style="{StaticResource SubtitleTextBlockStyle}"
Margin="0,0,0,0">
                Specifically, we define a custom delete command and add it
                to a variety of command surfaces, all of which share a
common
                icon, label, keyboard accelerator, and description.
            </TextBlock>
        </StackPanel>

        <muxcontrols:MenuBar Grid.Row="1">
            <muxcontrols:MenuBarItem Title="File">
            </muxcontrols:MenuBarItem>
            <muxcontrols:MenuBarItem Title="Edit">
                <MenuFlyoutItem x:Name="DeleteFlyoutItem"
                                Command="{StaticResource
```

```
CustomXamlUICommand}"/>
            </muxcontrols:MenuBarItem>
            <muxcontrols:MenuBarItem Title="Help">
            </muxcontrols:MenuBarItem>
        </muxcontrols:MenuBar>

        <ListView x:Name="ListViewRight" Grid.Row="2"
                  Loaded="ListView_Loaded"
                  IsItemClickEnabled="True"
                  SelectionMode="Single"
                  SelectionChanged="ListView_SelectionChanged"
                  ItemContainerStyle="{StaticResource HorizontalSwipe}">
            <ListView.ItemTemplate>
                <DataTemplate x:DataType="local:ListItemData">
                    <UserControl
PointerEntered="ListViewSwipeContainer_PointerEntered"

PointerExited="ListViewSwipeContainer_PointerExited">
                        <UserControl.ContextFlyout>
                            <MenuFlyout>
                                <MenuFlyoutItem
                                    Command="{x:Bind Command}"
                                    CommandParameter="{x:Bind Text}" />
                            </MenuFlyout>
                        </UserControl.ContextFlyout>
                        <Grid AutomationProperties.Name="{x:Bind Text}">
                            <VisualStateManager.VisualStateGroups>
                                <VisualStateGroup x:Name="HoveringStates">
                                    <VisualState x:Name="HoverButtonsHidden"
/>
                                    <VisualState x:Name="HoverButtonsShown">
                                        <VisualState.Setters>
                                            <Setter
Target="HoverButton.Visibility"
                                                    Value="Visible" />
                                        </VisualState.Setters>
                                    </VisualState>
                                </VisualStateGroup>
                            </VisualStateManager.VisualStateGroups>
                            <SwipeControl x:Name="ListViewSwipeContainer">
                                <SwipeControl.RightItems>
                                    <SwipeItems Mode="Execute">
                                        <SwipeItem x:Name="DeleteSwipeItem"
                                                   Background="Red"
                                                   Command="{x:Bind
Command}"
                                                   CommandParameter="{x:Bind
Text}"/>
                                    </SwipeItems>
                                </SwipeControl.RightItems>
                                <Grid VerticalAlignment="Center">
                                    <TextBlock Text="{x:Bind Text}"
                                               Margin="10"
                                               FontSize="18"
                                               HorizontalAlignment="Left"
```

```
                                                VerticalAlignment="Center"/>
                                        <AppBarButton x:Name="HoverButton"
                                                      IsTabStop="False"

HorizontalAlignment="Right"
                                                      Visibility="Collapsed"
                                                      Command="{x:Bind Command}"
                                                      CommandParameter="{x:Bind
Text}"/>
                                    </Grid>
                                </SwipeControl>
                            </Grid>
                        </UserControl>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
        </Grid>
</Page>
```

Code-behind

1. First, we define a `ListItemData` class that contains a text string and ICommand for each ListViewItem in our ListView.

C#

```
public class ListItemData
{
    public String Text { get; set; }
    public ICommand Command { get; set; }
}
```

2. In the MainPage class, we define a collection of `ListItemData` objects for the DataTemplate of the ListView ItemTemplate. We then populate it with an initial collection of five items (with text and associated XamlUICommand).

C#

```
ObservableCollection<ListItemData> collection = new
ObservableCollection<ListItemData>();

private void ControlExample_Loaded(object sender, RoutedEventArgs e)
{
    for (var i = 0; i < 5; i++)
    {
        collection.Add(
           new ListItemData { Text = "List item " + i.ToString(), Command =
CustomXamlUICommand });
    }
}
```

```
private void ListView_Loaded(object sender, RoutedEventArgs e)
{
    var listView = (ListView)sender;
    listView.ItemsSource = collection;
}
```

3. Next, we define the ICommand ExecuteRequested handler where we implement the item delete command.

C#

```
private void DeleteCommand_ExecuteRequested(
   XamlUICommand sender, ExecuteRequestedEventArgs args)
{
    if (args.Parameter != null)
    {
        foreach (var i in collection)
        {
            if (i.Text == (args.Parameter as string))
            {
                collection.Remove(i);
                return;
            }
        }
    }
    if (ListViewRight.SelectedIndex != -1)
    {
        collection.RemoveAt(ListViewRight.SelectedIndex);
    }
}
```

4. Finally, we define handlers for various ListView events, including PointerEntered, PointerExited, and SelectionChanged events. The pointer event handlers are used to show or hide the Delete button for each item.

C#

```
private void ListView_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (ListViewRight.SelectedIndex != -1)
    {
        var item = collection[ListViewRight.SelectedIndex];
    }
}

private void ListViewSwipeContainer_PointerEntered(object sender,
PointerRoutedEventArgs e)
{
    if (e.Pointer.PointerDeviceType ==
```

```
        Windows.Devices.Input.PointerDeviceType.Mouse ||
        e.Pointer.PointerDeviceType ==
Windows.Devices.Input.PointerDeviceType.Pen)
    {
        VisualStateManager.GoToState(sender as Control, "HoverButtonsShown",
true);
    }
}

private void ListViewSwipeContainer_PointerExited(object sender,
PointerRoutedEventArgs e)
{
    VisualStateManager.GoToState(sender as Control, "HoverButtonsHidden",
true);
}
```

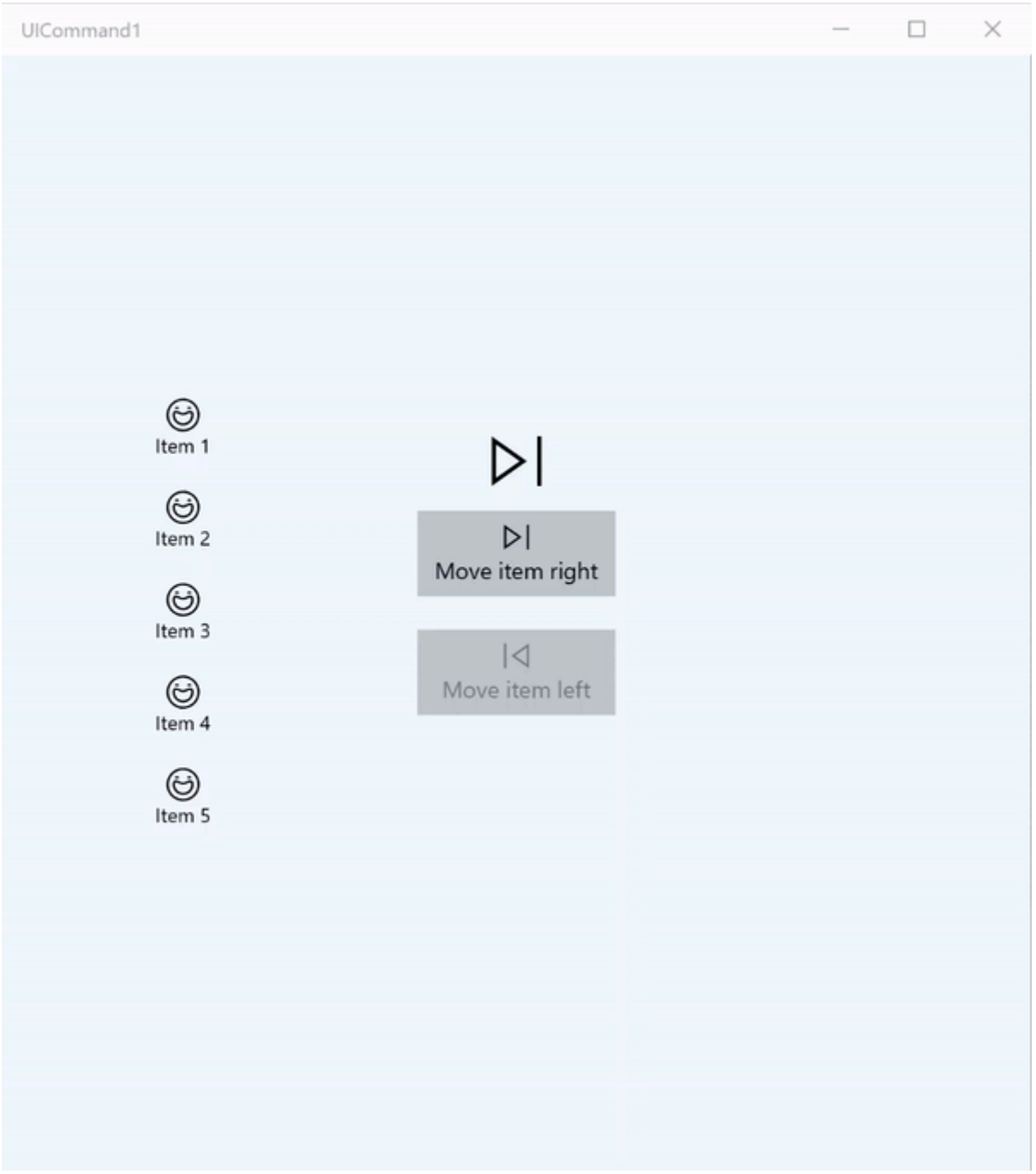
Command experiences using the ICommand interface

Standard UWP controls (button, list, selection, calendar, predictive text) provide the basis for many common command experiences. For a complete list of control types, see Controls and patterns for Windows apps.

The most basic way to support a structured commanding experience is to define an implementation of the ICommand interface (Windows.UI.Xaml.Input.ICommand for C++ or System.Windows.Input.ICommand for C#). This ICommand instance can then be bound to controls such as buttons.

> **Note**
>
> In some cases, it might be just as efficient to bind a method to the Click event and a property to the IsEnabled property.

Example



ICommand example

Expand table

Download the code for this example
UWP commanding sample (ICommand)

In this basic example, we demonstrate how a single command can be invoked with a button click, a keyboard accelerator, and rotating a mouse wheel.

We use two ListViews, one populated with five items and the other empty, and two buttons, one for moving items from the ListView on the left to the ListView on the right, and the other for moving items from the right to the left. Each button is bound to a

corresponding command (ViewModel.MoveRightCommand and ViewModel.MoveLeftCommand, respectively), and are enabled and disabled automatically based on the number of items in their associated ListView.

The following XAML code defines the UI for our example.

XAML

```
<Page
    x:Class="UICommand1.View.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:vm="using:UICommand1.ViewModel"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <Page.Resources>
        <vm:OpacityConverter x:Key="opaque" />
    </Page.Resources>

    <Grid Name="ItemGrid"
          Background="AliceBlue"
          PointerWheelChanged="Page_PointerWheelChanged">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="2*"/>
            <ColumnDefinition Width="*"/>
        </Grid.ColumnDefinitions>
        <ListView Grid.Column="0" VerticalAlignment="Center"
                  x:Name="CommandListView"
                  ItemsSource="{x:Bind Path=ViewModel.ListItemLeft}"
                  SelectionMode="None" IsItemClickEnabled="False"
                  HorizontalAlignment="Right">
            <ListView.ItemTemplate>
                <DataTemplate x:DataType="vm:ListItemData">
                    <Grid VerticalAlignment="Center">
                        <AppBarButton Label="{x:Bind ListItemText}">
                            <AppBarButton.Icon>
                                <SymbolIcon Symbol="{x:Bind ListItemIcon}"/>
                            </AppBarButton.Icon>
                        </AppBarButton>
                    </Grid>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
        <Grid Grid.Column="1" Margin="0,0,0,0"
              HorizontalAlignment="Center"
              VerticalAlignment="Center">
            <Grid.RowDefinitions>
                <RowDefinition Height="*"/>
                <RowDefinition Height="*"/>
```

```
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>
            <StackPanel Grid.Row="1">
                <FontIcon FontFamily="{StaticResource
SymbolThemeFontFamily}"
                          FontSize="40" Glyph="&#xE893;"
                          Opacity="{x:Bind
Path=ViewModel.ListItemLeft.Count,
                                           Mode=OneWay, Converter=
{StaticResource opaque}}"/>
                <Button Name="MoveItemRightButton"
                        Margin="0,10,0,10" Width="120"
HorizontalAlignment="Center"
                        Command="{x:Bind Path=ViewModel.MoveRightCommand}">
                    <Button.KeyboardAccelerators>
                        <KeyboardAccelerator
                            Modifiers="Control"
                            Key="Add" />
                    </Button.KeyboardAccelerators>
                    <StackPanel>
                        <SymbolIcon Symbol="Next"/>
                        <TextBlock>Move item right</TextBlock>
                    </StackPanel>
                </Button>
                <Button Name="MoveItemLeftButton"
                            Margin="0,10,0,10" Width="120"
HorizontalAlignment="Center"
                            Command="{x:Bind
Path=ViewModel.MoveLeftCommand}">
                    <Button.KeyboardAccelerators>
                        <KeyboardAccelerator
                            Modifiers="Control"
                            Key="Subtract" />
                    </Button.KeyboardAccelerators>
                    <StackPanel>
                        <SymbolIcon Symbol="Previous"/>
                        <TextBlock>Move item left</TextBlock>
                    </StackPanel>
                </Button>
                <FontIcon FontFamily="{StaticResource
SymbolThemeFontFamily}"
                          FontSize="40" Glyph="&#xE892;"
                          Opacity="{x:Bind
Path=ViewModel.ListItemRight.Count,
                                           Mode=OneWay, Converter=
{StaticResource opaque}}"/>
            </StackPanel>
        </Grid>
        <ListView Grid.Column="2"
                  x:Name="CommandListViewRight"
                  VerticalAlignment="Center"
                  IsItemClickEnabled="False"
                  SelectionMode="None"
                  ItemsSource="{x:Bind Path=ViewModel.ListItemRight}"
                  HorizontalAlignment="Left">
```

```
            <ListView.ItemTemplate>
                <DataTemplate x:DataType="vm:ListItemData">
                    <Grid VerticalAlignment="Center">
                        <AppBarButton Label="{x:Bind ListItemText}">
                            <AppBarButton.Icon>
                                <SymbolIcon Symbol="{x:Bind ListItemIcon}"/>
                            </AppBarButton.Icon>
                        </AppBarButton>
                    </Grid>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </Grid>
</Page>
```

Here's the code-behind for the preceding UI.

In code-behind, we connect to our view model that contains our command code. In addition, we define a handler for input from the mouse wheel, which also connects our command code.

C#

```
using Windows.UI.Xaml;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Controls;
using UICommand1.ViewModel;
using Windows.System;
using Windows.UI.Core;

namespace UICommand1.View
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a
Frame.
    /// </summary>
    public sealed partial class MainPage : Page
    {
        // Reference to our view model.
        public UICommand1ViewModel ViewModel { get; set; }

        // Initialize our view and view model.
        public MainPage()
        {
            this.InitializeComponent();
            ViewModel = new UICommand1ViewModel();
        }

        /// <summary>
        /// Handle mouse wheel input and assign our
        /// commands to appropriate direction of rotation.
        /// </summary>
```

```
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void Page_PointerWheelChanged(object sender,
PointerRoutedEventArgs e)
        {
            var props = e.GetCurrentPoint(sender as UIElement).Properties;

            // Require CTRL key and accept only vertical mouse wheel
movement
            // to eliminate accidental wheel input.
            if ((Window.Current.CoreWindow.GetKeyState(VirtualKey.Control)
!=
                CoreVirtualKeyStates.None) && !props.IsHorizontalMouseWheel)
            {
                bool delta = props.MouseWheelDelta < 0 ? true : false;

                switch (delta)
                {
                    case true:
                        ViewModel.MoveRight();
                        break;
                    case false:
                        ViewModel.MoveLeft();
                        break;
                    default:
                        break;
                }
            }
        }
    }
}
```

Here's the code from our view model

Our view model is where we define the execution details for the two commands in our app, populate one ListView, and provide an opacity value converter for hiding or displaying some additional UI based on the item count of each ListView.

C#

```
using System;
using System.Collections.ObjectModel;
using System.ComponentModel;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Data;

namespace UICommand1.ViewModel
{
    /// <summary>
    /// UI properties for our list items.
    /// </summary>
    public class ListItemData
```

```
    {
        /// <summary>
        /// Gets and sets the list item content string.
        /// </summary>
        public string ListItemText { get; set; }
        /// <summary>
        /// Gets and sets the list item icon.
        /// </summary>
        public Symbol ListItemIcon { get; set; }
    }

    /// <summary>
    /// View Model that sets up a command to handle invoking the move item
buttons.
    /// </summary>
    public class UICommand1ViewModel
    {
        /// <summary>
        /// The command to invoke when the Move item left button is pressed.
        /// </summary>
        public RelayCommand MoveLeftCommand { get; private set; }

        /// <summary>
        /// The command to invoke when the Move item right button is
pressed.
        /// </summary>
        public RelayCommand MoveRightCommand { get; private set; }

        // Item collections
        public ObservableCollection<ListItemData> ListItemLeft { get; } =
           new ObservableCollection<ListItemData>();
        public ObservableCollection<ListItemData> ListItemRight { get; } =
           new ObservableCollection<ListItemData>();

        public ListItemData listItem;

        /// <summary>
        /// Sets up a command to handle invoking the move item buttons.
        /// </summary>
        public UICommand1ViewModel()
        {
            MoveLeftCommand =
               new RelayCommand(new Action(MoveLeft),
CanExecuteMoveLeftCommand);
            MoveRightCommand =
               new RelayCommand(new Action(MoveRight),
CanExecuteMoveRightCommand);

            LoadItems();
        }

        /// <summary>
        ///  Populate our list of items.
        /// </summary>
        public void LoadItems()
```

```
        {
            for (var x = 0; x <= 4; x++)
            {
                listItem = new ListItemData();
                listItem.ListItemText = "Item " + (ListItemLeft.Count +
1).ToString();
                listItem.ListItemIcon = Symbol.Emoji;
                ListItemLeft.Add(listItem);
            }
        }

        /// <summary>
        /// Move left command valid when items present in the list on right.
        /// </summary>
        /// <returns>True, if count is greater than 0.</returns>
        private bool CanExecuteMoveLeftCommand()
        {
            return ListItemRight.Count > 0;
        }

        /// <summary>
        /// Move right command valid when items present in the list on left.
        /// </summary>
        /// <returns>True, if count is greater than 0.</returns>
        private bool CanExecuteMoveRightCommand()
        {
            return ListItemLeft.Count > 0;
        }

        /// <summary>
        /// The command implementation to execute when the Move item right
button is pressed.
        /// </summary>
        public void MoveRight()
        {
            if (ListItemLeft.Count > 0)
            {
                listItem = new ListItemData();
                ListItemRight.Add(listItem);
                listItem.ListItemText = "Item " +
ListItemRight.Count.ToString();
                listItem.ListItemIcon = Symbol.Emoji;
                ListItemLeft.RemoveAt(ListItemLeft.Count - 1);
                MoveRightCommand.RaiseCanExecuteChanged();
                MoveLeftCommand.RaiseCanExecuteChanged();
            }
        }

        /// <summary>
        /// The command implementation to execute when the Move item left
button is pressed.
        /// </summary>
        public void MoveLeft()
        {
            if (ListItemRight.Count > 0)
```

```
            {
                listItem = new ListItemData();
                ListItemLeft.Add(listItem);
                listItem.ListItemText = "Item " +
ListItemLeft.Count.ToString();
                listItem.ListItemIcon = Symbol.Emoji;
                ListItemRight.RemoveAt(ListItemRight.Count - 1);
                MoveRightCommand.RaiseCanExecuteChanged();
                MoveLeftCommand.RaiseCanExecuteChanged();
            }
        }

        /// <summary>
        /// Views subscribe to this event to get notified of property
updates.
        /// </summary>
        public event PropertyChangedEventHandler PropertyChanged;

        /// <summary>
        /// Notify subscribers of updates to the named property
        /// </summary>
        /// <param name="propertyName">The full, case-sensitive, name of a
property.</param>
        protected void NotifyPropertyChanged(string propertyName)
        {
            PropertyChangedEventHandler handler = this.PropertyChanged;
            if (handler != null)
            {
                PropertyChangedEventArgs args = new
PropertyChangedEventArgs(propertyName);
                handler(this, args);
            }
        }
    }

    /// <summary>
    /// Convert a collection count to an opacity value of 0.0 or 1.0.
    /// </summary>
    public class OpacityConverter : IValueConverter
    {
        /// <summary>
        /// Converts a collection count to an opacity value of 0.0 or 1.0.
        /// </summary>
        /// <param name="value">The count passed in</param>
        /// <param name="targetType">Ignored.</param>
        /// <param name="parameter">Ignored</param>
        /// <param name="language">Ignored</param>
        /// <returns>1.0 if count > 0, otherwise returns 0.0</returns>
        public object Convert(object value, Type targetType, object
parameter, string language)
        {
            return ((int)value > 0 ? 1.0 : 0.0);
        }

        /// <summary>
```

```
        /// Not used, converter is not intended for two-way binding.
        /// </summary>
        /// <param name="value">Ignored</param>
        /// <param name="targetType">Ignored</param>
        /// <param name="parameter">Ignored</param>
        /// <param name="language">Ignored</param>
        /// <returns></returns>
        public object ConvertBack(object value, Type targetType, object
parameter, string language)
        {
            throw new NotImplementedException();
        }
    }
}
```

Finally, here's our implementation of the ICommand interface

Here, we define a command that implements the ICommand interface and simply relays its functionality to other objects.

C#

```
using System;
using System.Windows.Input;

namespace UICommand1
{
    /// <summary>
    /// A command whose sole purpose is to relay its functionality
    /// to other objects by invoking delegates.
    /// The default return value for the CanExecute method is 'true'.
    /// <see cref="RaiseCanExecuteChanged"/> needs to be called whenever
    /// <see cref="CanExecute"/> is expected to return a different value.
    /// </summary>
    public class RelayCommand : ICommand
    {
        private readonly Action _execute;
        private readonly Func<bool> _canExecute;

        /// <summary>
        /// Raised when RaiseCanExecuteChanged is called.
        /// </summary>
        public event EventHandler CanExecuteChanged;

        /// <summary>
        /// Creates a new command that can always execute.
        /// </summary>
        /// <param name="execute">The execution logic.</param>
        public RelayCommand(Action execute)
            : this(execute, null)
        {
        }
```

```
        /// <summary>
        /// Creates a new command.
        /// </summary>
        /// <param name="execute">The execution logic.</param>
        /// <param name="canExecute">The execution status logic.</param>
        public RelayCommand(Action execute, Func<bool> canExecute)
        {
            if (execute == null)
                throw new ArgumentNullException("execute");
            _execute = execute;
            _canExecute = canExecute;
        }

        /// <summary>
        /// Determines whether this <see cref="RelayCommand"/> can execute
in its current state.
        /// </summary>
        /// <param name="parameter">
        /// Data used by the command. If the command does not require
        /// data to be passed, this object can be set to null.
        /// </param>
        /// <returns>true if this command can be executed; otherwise, false.
</returns>
        public bool CanExecute(object parameter)
        {
            return _canExecute == null ? true : _canExecute();
        }

        /// <summary>
        /// Executes the <see cref="RelayCommand"/> on the current command
target.
        /// </summary>
        /// <param name="parameter">
        /// Data used by the command. If the command does not require
        /// data to be passed, this object can be set to null.
        /// </param>
        public void Execute(object parameter)
        {
            _execute();
        }

        /// <summary>
        /// Method used to raise the <see cref="CanExecuteChanged"/> event
        /// to indicate that the return value of the <see
cref="CanExecute"/>
        /// method has changed.
        /// </summary>
        public void RaiseCanExecuteChanged()
        {
            var handler = CanExecuteChanged;
            if (handler != null)
            {
                handler(this, EventArgs.Empty);
            }
        }
```

```
    }
}
```

Summary

The Universal Windows Platform provides a robust and flexible commanding system that lets you build apps that share and manage commands across control types, devices, and input types.

Use the following approaches when building commands for your Windows apps:

- Listen for and handle events in XAML/code-behind
- Bind to an event handling method such as Click
- Define your own ICommand implementation
- Create XamlUICommand objects with your own values for a pre-defined set of properties
- Create StandardUICommand objects with a set of pre-defined platform properties and values

Next steps

For a complete example that demonstrates a XamlUICommand and StandardUICommand implementation, see the WinUI 2 Gallery sample.

See also

Controls and patterns for Windows apps

Samples

Topic samples

- UWP commanding sample (StandardUICommand)
- UWP commanding sample (XamlUICommand)
- UWP commanding sample (ICommand)

Other samples

- Universal Windows Platform samples (C# and C++)
- WinUI 2 Gallery

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

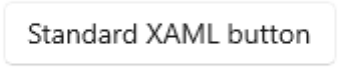
Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Buttons

Article • 02/26/2025

A button gives the user a way to trigger an immediate action. Some buttons are specialized for particular tasks, such as navigation, repeated actions, or presenting menus.



The Extensible Application Markup Language (XAML) framework provides a standard button control as well as several specialized button controls.

Expand table

Control	Description
Button	A button that initiates an immediate action. Can be used with a Click event or Command binding.
RepeatButton	A button that raises a Click event continuously while pressed.
HyperlinkButton	A button that's styled like a hyperlink and used for navigation. For more info about hyperlinks, see Hyperlinks.
DropDownButton	A button with a chevron to open an attached flyout.
SplitButton	A button with two sides. One side initiates an action, and the other side opens a menu.
ToggleSplitButton	A toggle button with two sides. One side toggles on/off, and the other side opens a menu.
ToggleButton	A button that can be on or off.

Is this the right control?

Use a **Button** control to let the user initiate an immediate action, such as submitting a form.

Don't use a **Button** control when the action is to navigate to another page; instead, use a HyperlinkButton control. For more info about hyperlinks, see Hyperlinks.

> **ⓘ Important**
>
> For wizard navigation, use buttons labeled *Back* and *Next*. For other types of backwards navigation or navigation to an upper level, use a **back button**.

Use a **RepeatButton** control when the user might want to trigger an action repeatedly. For example, use a **RepeatButton** control to increment or decrement a value in a counter.

Use a **DropDownButton** control when the button has a flyout that contains more options. The default chevron provides a visual indication that the button includes a flyout.

Use a **SplitButton** control when you want the user to be able to initiate an immediate action or choose from additional options independently.

Use a **ToggleButton** control when you want the user to be able to immediately switch between two mutually exclusive states, and a button is the best fit for your UI needs. Unless your UI benefits from a button, it might be a better choice to use an AppBarToggleButton, CheckBox, RadioButton, or ToggleSwitch.

Recommendations

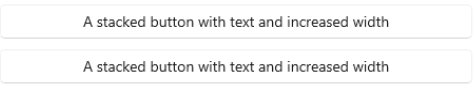
- Make sure the purpose and state of a button are clear to the user.
- When there are multiple buttons for the same decision (such as in a confirmation dialog), present the commit buttons in this order, where [Do it] and [Don't do it] are specific responses to the main instruction:
  - OK/[Do it]/Yes
    - [Don't do it]/No
    - Cancel
- Expose only one or two buttons to the user at a time, for example, **Accept** and **Cancel**. If you need to expose more actions to the user, consider using checkboxes or radio buttons from which the user can select actions, with a single command button to trigger those actions.
- For an action that needs to be available across multiple pages within your app, instead of duplicating a button on multiple pages, consider using a bottom app bar.

Button text

A button's content is usually text. When you design that text, use the following recommendations:

- Use a concise, specific, self-explanatory text that clearly describes the action that the button performs. Usually button text is a single word that is a verb.
- Use the default font, unless your brand guidelines tell you to use something different.
- For shorter text, avoid narrow command buttons by using a minimum button width of 120px.
- For longer text, avoid wide command buttons by limiting text to a maximum length of 26 characters.
- If the button's text content is dynamic (that is, it is localized), consider how the button will be resized and what will happen to controls around it.

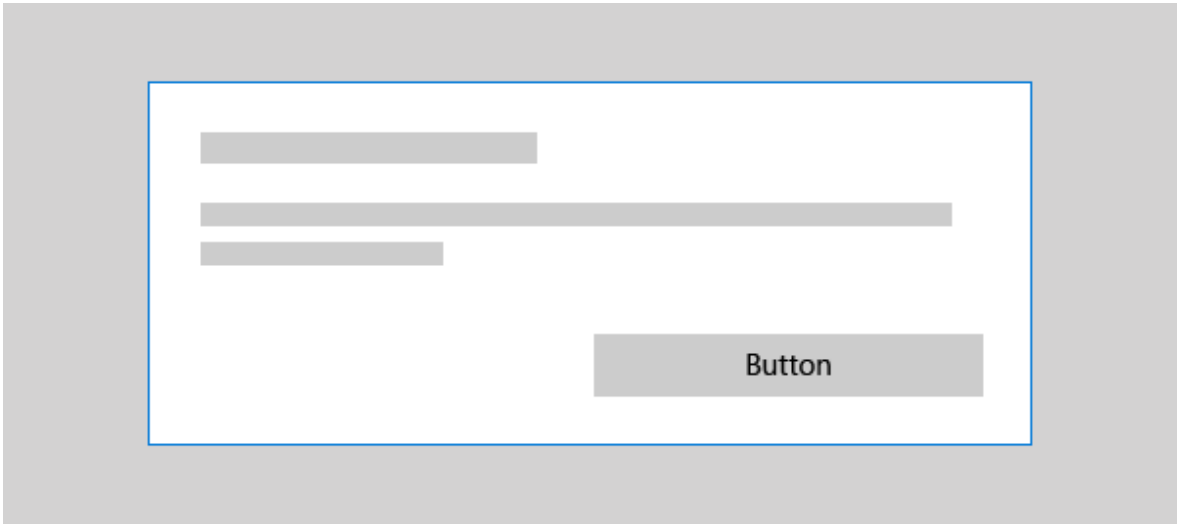
Expand table

Need to fix: Buttons with overflowing text.	
Option 1: Increase button width, stack buttons, and wrap if text length is greater than 26 characters.	
Option 2: Increase button height, and wrap text.	

Recommended single-button layout

If your layout requires only one button, it should be either left- or right-aligned based on its container context.

- Dialogs with only one button should **right-align** the button. If your dialog contains only one button, ensure that the button performs the safe, nondestructive action. If you use ContentDialog and specify a single button, it will be automatically right-aligned.



- If your button appears within a container UI (for example, within a toast notification, a flyout, or a list view item), you should **right-align** the button within the container.

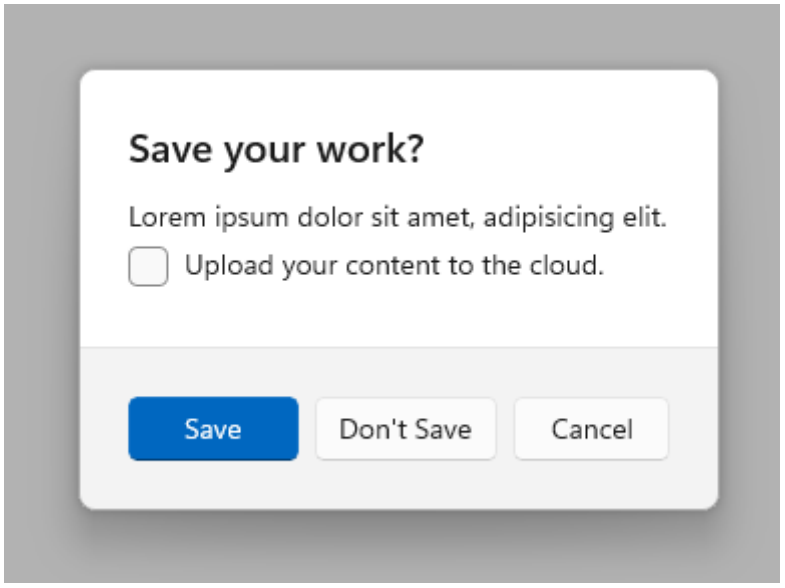
- In pages that contain a single button (for example, an **Apply** button at the bottom of a settings page), you should **left-align** the button. This ensures that the button aligns with the rest of the page content.

Back buttons

The back button is a system-provided UI element that enables backward navigation through either the back stack or navigation history of the user. You don't have to create your own back button, but you might have to do some work to enable a good backwards navigation experience. For more info, see Navigation history and backwards navigation for Windows apps.

Examples

This example uses three buttons, **Save**, **Don't Save**, and **Cancel**, in a dialog that asks users if they want to save their work.



Create a button

✔ **Important APIs:** Button class, Click event, Command property, Content property

Open the WinUI 3 Gallery app and see the Buttons in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This example shows a button that responds to a click.

Create the button in XAML.

XAML

```
<Button Content="Subscribe" Click="SubscribeButton_Click"/>
```

Or create the button in code.

C#

```
Button subscribeButton = new Button();
subscribeButton.Content = "Subscribe";
subscribeButton.Click += SubscribeButton_Click;

// Add the button to a parent container in the visual tree.
stackPanel1.Children.Add(subscribeButton);
```

Handle the Click event.

C#

```
private async void SubscribeButton_Click(object sender, RoutedEventArgs e)
{
    // Call app specific code to subscribe to the service. For example:
    ContentDialog subscribeDialog = new ContentDialog
    {
        Title = "Subscribe to App Service?",
        Content = "Listen, watch, and play in high definition for only
$9.99/month. Free to try, cancel anytime.",
        CloseButtonText = "Not Now",
        PrimaryButtonText = "Subscribe",
        SecondaryButtonText = "Try it"
    };

    ContentDialogResult result = await subscribeDialog.ShowAsync();
}
```

Button interaction

When you tap a **Button** control with a finger or stylus, or press a left mouse button while the pointer is over it, the button raises the Click event. If a button has keyboard focus, pressing the Enter key or the Spacebar also raises the **Click** event.

You generally can't handle low-level PointerPressed events on a **Button** object because it has the **Click** behavior instead. For more info, see Events and routed events overview.

You can change how a button raises the **Click** event by changing the ClickMode property. The default value of **ClickMode** is **Release**, but you also can set a button's **ClickMode** value to **Hover** or **Press**. If **ClickMode** is **Hover**, the **Click** event can't be raised by using the keyboard or touch.

Button content

Button is a content control of the ContentControl class. Its XAML content property is Content, which enables a syntax like this for XAML: `<Button>A button's content</Button>`. You can set any object as the button's content. If the content is a UIElement object, it is rendered in the button. If the content is another type of object, its string representation is shown in the button.

A button's content is usually text. When you design that text, follow the Button text recommendations listed previously.

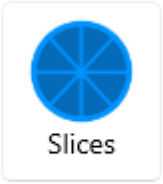
You can also customize visuals that make up the button's appearance. For example, you could replace the text with an icon, or use an icon in addition to text.

Here, a **StackPanel** that contains an image and text is set as the content of a button.

XAML

```
<Button x:Name="Button2" Click="Button_Click" Width="80" Height="90">
    <StackPanel>
        <Image Source="/Assets/Slices.png" Height="52"/>
        <TextBlock Text="Slices" Foreground="Black"
HorizontalAlignment="Center"/>
    </StackPanel>
</Button>
```

The button looks like this.



Create a repeat button

✔ **Important APIs:** RepeatButton class, Click event, Content property

Open the WinUI 3 Gallery app and see the RepeatButton in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

A RepeatButton control is a button that raises Click events repeatedly from the time it's pressed until it's released. Set the Delay property to specify the time that the **RepeatButton** control waits after it is pressed before it starts repeating the click action. Set the Interval property to specify the time between repetitions of the click action. Times for both properties are specified in milliseconds.

The following example shows two **RepeatButton** controls whose respective **Click** events are used to increase and decrease the value shown in a text block.

XAML

```
<StackPanel>
    <RepeatButton Width="100" Delay="500" Interval="100"
Click="Increase_Click">Increase</RepeatButton>
    <RepeatButton Width="100" Delay="500" Interval="100"
Click="Decrease_Click">Decrease</RepeatButton>
    <TextBlock x:Name="clickTextBlock" Text="Number of Clicks:" />
</StackPanel>
```

C#

```
private static int _clicks = 0;
private void Increase_Click(object sender, RoutedEventArgs e)
{
    _clicks += 1;
```

```
    clickTextBlock.Text = "Number of Clicks: " + _clicks;
}

private void Decrease_Click(object sender, RoutedEventArgs e)
{
    if(_clicks > 0)
    {
        _clicks -= 1;
        clickTextBlock.Text = "Number of Clicks: " + _clicks;
    }
}
```

Create a drop down button

✔ **Important APIs**: DropDownButton class, Flyout property

Open the WinUI 3 Gallery app and see the DropdownButton in action

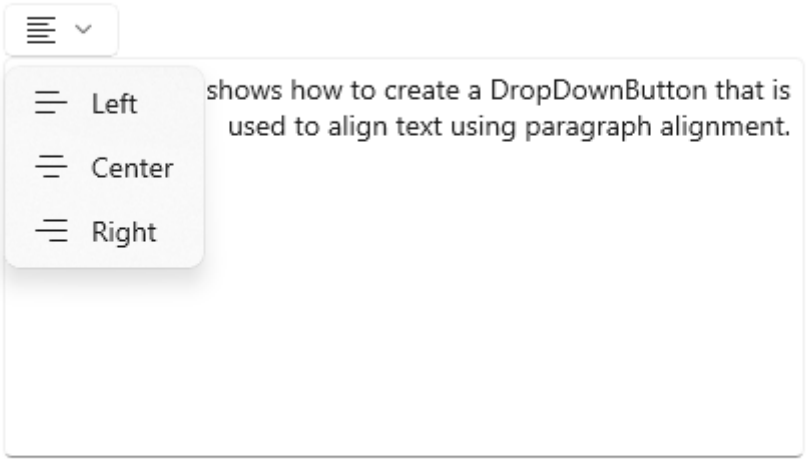
> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

A DropDownButton is a button that shows a chevron as a visual indicator that it has an attached flyout that contains more options. It has the same behavior as a standard **Button** control with a flyout; only the appearance is different.

The drop down button inherits the **Click** event, but you typically don't use it. Instead, you use the Flyout property to attach a flyout and invoke actions by using menu options in the flyout. The flyout opens automatically when the button is clicked. Be sure to specify the Placement property of your flyout to ensure the desired placement in relation to the button. The default placement algorithm might not produce the intended placement in all situations. For more info about flyouts, see Flyouts and Menu flyout and menu bar.

Example - Drop down button

This example shows how to create a drop down button with a flyout that contains commands for paragraph alignment in a **RichEditBox** control. (For more info and code, see Rich edit box).



XAML

```
<DropDownButton ToolTipService.ToolTip="Alignment">
    <TextBlock FontFamily="Segoe MDL2 Assets" FontSize="14"
Text="&#xE8E4;"/>
    <DropDownButton.Flyout>
        <MenuFlyout Placement="BottomEdgeAlignedLeft">
            <MenuFlyoutItem Text="Left" Icon="AlignLeft" Tag="left"
                            Click="AlignmentMenuFlyoutItem_Click"/>
            <MenuFlyoutItem Text="Center" Icon="AlignCenter" Tag="center"
                            Click="AlignmentMenuFlyoutItem_Click"/>
            <MenuFlyoutItem Text="Right" Icon="AlignRight" Tag="right"
                            Click="AlignmentMenuFlyoutItem_Click"/>
        </MenuFlyout>
    </DropDownButton.Flyout>
</DropDownButton>
```

C#

```
private void AlignmentMenuFlyoutItem_Click(object sender, RoutedEventArgs e)
{
    var option = ((MenuFlyoutItem)sender).Tag.ToString();

    Windows.UI.Text.ITextSelection selectedText = editor.Document.Selection;
    if (selectedText != null)
    {
        // Apply the alignment to the selected paragraphs.
        var paragraphFormatting = selectedText.ParagraphFormat;
        if (option == "left")
        {
            paragraphFormatting.Alignment =
Windows.UI.Text.ParagraphAlignment.Left;
        }
        else if (option == "center")
        {
```

```
            paragraphFormatting.Alignment =
Windows.UI.Text.ParagraphAlignment.Center;
        }
        else if (option == "right")
        {
            paragraphFormatting.Alignment =
Windows.UI.Text.ParagraphAlignment.Right;
        }
    }
}
```

Create a split button

✔ **Important APIs**: SplitButton class, Click event, Flyout property

Open the WinUI 3 Gallery app and see the SplitButton in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

A SplitButton control has two parts that can be invoked separately. One part behaves like a standard button and invokes an immediate action. The other part invokes a flyout that contains additional options that the user can choose from.

> **Note**
>
> When invoked with touch, the split button behaves as a drop down button; both halves of the button invoke the flyout. With other methods of input, a user can invoke either half of the button separately.

The typical behavior for a split button is:

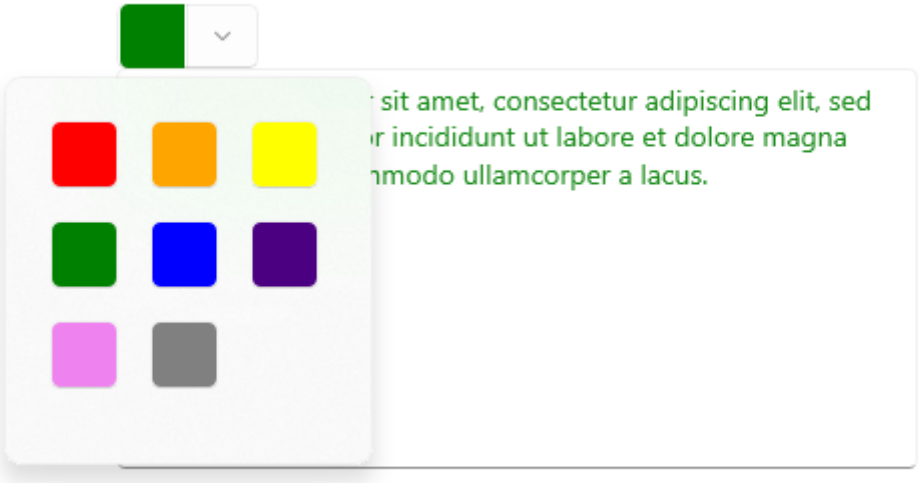
- When the user clicks the button part, handle the **Click** event to invoke the option that's currently selected in the drop down.
- When the drop down is open, handle invocation of the items in the drop down to both change which option is selected, and then invoke it. It's important to invoke the flyout item because the button **Click** event doesn't occur when using touch.

> **Tip**

There are many ways to put items in the drop down and handle their invocation. If you use a **ListView** or **GridView**, one way is to handle the **SelectionChanged** event. If you do this, set SingleSelectionFollowsFocus to **false**. This lets users navigate the options using a keyboard without invoking the item on each change.

Example - Split button

This example shows how to create a split button that is used to change the foreground color of selected text in a **RichEditBox** control. (For more info and code, see Rich edit box). Split button's flyout uses BottomEdgeAlignedLeft as the default value for its Placement property. You can't override this value.



XAML

```
<SplitButton ToolTipService.ToolTip="Foreground color"
             Click="BrushButtonClick">
    <Border x:Name="SelectedColorBorder" Width="20" Height="20"/>
    <SplitButton.Flyout>
        <Flyout x:Name="BrushFlyout">
            <!-- Set SingleSelectionFollowsFocus="False"
                 so that keyboard navigation works correctly. -->
            <GridView ItemsSource="{x:Bind ColorOptions}"
                      SelectionChanged="BrushSelectionChanged"
                      SingleSelectionFollowsFocus="False"
                      SelectedIndex="0" Padding="0">
                <GridView.ItemTemplate>
                    <DataTemplate>
                        <Rectangle Fill="{Binding}" Width="20" Height="20"/>
                    </DataTemplate>
                </GridView.ItemTemplate>
                <GridView.ItemContainerStyle>
```

```
                    <Style TargetType="GridViewItem">
                        <Setter Property="Margin" Value="2"/>
                        <Setter Property="MinWidth" Value="0"/>
                        <Setter Property="MinHeight" Value="0"/>
                    </Style>
                </GridView.ItemContainerStyle>
            </GridView>
        </Flyout>
    </SplitButton.Flyout>
</SplitButton>
```

C#

```
public sealed partial class MainPage : Page
{
    // Color options that are bound to the grid in the split button flyout.
    private List<SolidColorBrush> ColorOptions = new List<SolidColorBrush>
();
    private SolidColorBrush CurrentColorBrush = null;

    public MainPage()
    {
        this.InitializeComponent();

        // Add color brushes to the collection.
        ColorOptions.Add(new SolidColorBrush(Colors.Black));
        ColorOptions.Add(new SolidColorBrush(Colors.Red));
        ColorOptions.Add(new SolidColorBrush(Colors.Orange));
        ColorOptions.Add(new SolidColorBrush(Colors.Yellow));
        ColorOptions.Add(new SolidColorBrush(Colors.Green));
        ColorOptions.Add(new SolidColorBrush(Colors.Blue));
        ColorOptions.Add(new SolidColorBrush(Colors.Indigo));
        ColorOptions.Add(new SolidColorBrush(Colors.Violet));
        ColorOptions.Add(new SolidColorBrush(Colors.White));
    }

    private void BrushButtonClick(object sender, object e)
    {
        // When the button part of the split button is clicked,
        // apply the selected color.
        ChangeColor();
    }

    private void BrushSelectionChanged(object sender,
SelectionChangedEventArgs e)
    {
        // When the flyout part of the split button is opened and the user
selects
        // an option, set their choice as the current color, apply it, then
close the flyout.
        CurrentColorBrush = (SolidColorBrush)e.AddedItems[0];
        SelectedColorBorder.Background = CurrentColorBrush;
        ChangeColor();
```

```
            BrushFlyout.Hide();
        }

        private void ChangeColor()
        {
            // Apply the color to the selected text in a RichEditBox.
            Windows.UI.Text.ITextSelection selectedText =
editor.Document.Selection;
            if (selectedText != null)
            {
                Windows.UI.Text.ITextCharacterFormat charFormatting =
selectedText.CharacterFormat;
                charFormatting.ForegroundColor = CurrentColorBrush.Color;
                selectedText.CharacterFormat = charFormatting;
            }
        }
}
```

Create a toggle split button

✓ **Important APIs**: ToggleSplitButton class, IsCheckedChanged event, IsChecked property

Open the WinUI 3 Gallery app and see the ToggleSplitButton in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

A ToggleSplitButton control has two parts that can be invoked separately. One part behaves like a toggle button that can be on or off. The other part invokes a flyout that contains additional options that the user can choose from.

A toggle split button is typically used to enable or disable a feature when the feature has multiple options that the user can choose from. For example, in a document editor, it could be used to turn lists on or off, while the drop down is used to choose the style of the list.

Note

When invoked with touch, the toggle split button behaves as a drop down button. With other methods of input, a user can toggle and invoke the two halves of the button separately. With touch, both halves of the button invoke the flyout.

Therefore, you must include an option in your flyout content to toggle the button on or off.

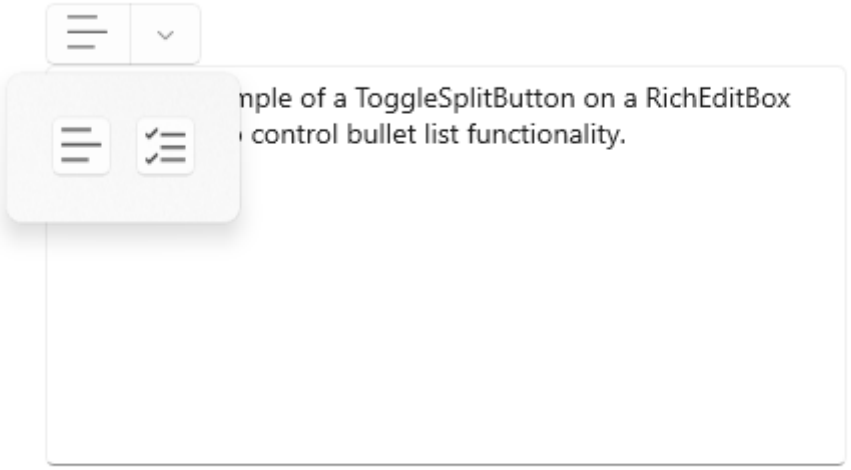
Differences with ToggleButton

Unlike ToggleButton, **ToggleSplitButton** does not have an indeterminate state. As a result, you should keep in mind these differences:

- **ToggleSplitButton** does not have an **IsThreeState** property or **Indeterminate** event.
- The ToggleSplitButton.IsChecked property is just a Boolean, not a **Nullable<bool>**.
- **ToggleSplitButton** has only the IsCheckedChanged event; it does not have separate **Checked** and **Unchecked** events.

Example - Toggle split button

The following example shows how a toggle split button could be used to turn list formatting on or off, and change the style of the list, in a **RichEditBox** control. (For more info and code, see Rich edit box). The flyout of the toggle split button uses BottomEdgeAlignedLeft as the default value for its Placement property. You can't override this value.



XAML

```
<ToggleSplitButton x:Name="ListButton"
                   ToolTipService.ToolTip="List style"
                   Click="ListButton_Click"
                   IsCheckedChanged="ListStyleButton_IsCheckedChanged">
```

```
    <TextBlock FontFamily="Segoe MDL2 Assets" FontSize="14"
Text="&#xE8FD;"/>
    <ToggleSplitButton.Flyout>
        <Flyout>
            <ListView x:Name="ListStylesListView"
                      SelectionChanged="ListStylesListView_SelectionChanged"
                      SingleSelectionFollowsFocus="False">
                <StackPanel Tag="bullet" Orientation="Horizontal">
                    <FontIcon FontFamily="Segoe MDL2 Assets"
Glyph="&#xE7C8;"/>
                    <TextBlock Text="Bullet" Margin="8,0"/>
                </StackPanel>
                <StackPanel Tag="alpha" Orientation="Horizontal">
                    <TextBlock Text="A" FontSize="24" Margin="2,0"/>
                    <TextBlock Text="Alpha" Margin="8"/>
                </StackPanel>
                <StackPanel Tag="numeric" Orientation="Horizontal">
                    <FontIcon FontFamily="Segoe MDL2 Assets"
Glyph="&#xF146;"/>
                    <TextBlock Text="Numeric" Margin="8,0"/>
                </StackPanel>
                <TextBlock Tag="none" Text="None" Margin="28,0"/>
            </ListView>
        </Flyout>
    </ToggleSplitButton.Flyout>
</ToggleSplitButton>
```

C#

```
private void ListStyleButton_IsCheckedChanged(ToggleSplitButton sender,
ToggleSplitButtonIsCheckedChangedEventArgs args)
{
    // Use the toggle button to turn the selected list style on or off.
    if (((ToggleSplitButton)sender).IsChecked == true)
    {
        // On. Apply the list style selected in the drop down to the
selected text.
        var listStyle = ((FrameworkElement)
(ListStylesListView.SelectedItem)).Tag.ToString();
        ApplyListStyle(listStyle);
    }
    else
    {
        // Off. Make the selected text not a list,
        // but don't change the list style selected in the drop down.
        ApplyListStyle("none");
    }
}

private void ListStylesListView_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    var listStyle = ((FrameworkElement)(e.AddedItems[0])).Tag.ToString();
```

```
    if (ListButton.IsChecked == true)
    {
        // Toggle button is on. Turn it off...
        if (listStyle == "none")
        {
            ListButton.IsChecked = false;
        }
        else
        {
            // or apply the new selection.
            ApplyListStyle(listStyle);
        }
    }
    else
    {
        // Toggle button is off. Turn it on, which will apply the selection
        // in the IsCheckedChanged event handler.
        ListButton.IsChecked = true;
    }
}

private void ApplyListStyle(string listStyle)
{
    Windows.UI.Text.ITextSelection selectedText = editor.Document.Selection;
    if (selectedText != null)
    {
        // Apply the list style to the selected text.
        var paragraphFormatting = selectedText.ParagraphFormat;
        if (listStyle == "none")
        {
            paragraphFormatting.ListType = Windows.UI.Text.MarkerType.None;
        }
        else if (listStyle == "bullet")
        {
            paragraphFormatting.ListType =
Windows.UI.Text.MarkerType.Bullet;
        }
        else if (listStyle == "numeric")
        {
            paragraphFormatting.ListType =
Windows.UI.Text.MarkerType.Arabic;
        }
        else if (listStyle == "alpha")
        {
            paragraphFormatting.ListType =
Windows.UI.Text.MarkerType.UppercaseEnglishLetter;
        }
        selectedText.ParagraphFormat = paragraphFormatting;
    }
}
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The DropDownButton, SplitButton, and ToggleSplitButton controls for UWP apps are included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for these controls exist in both the Windows.UI.Xaml.Controls and Microsoft.UI.Xaml.Controls namespaces.

- ✓ **UWP APIs:** Button class, RepeatButton class, HyperlinkButton class, DropDownButton, SplitButton, ToggleSplitButton, ToggleButton, Click event, Command property, Content property
- ✓ **WinUI 2 Apis:** DropDownButton, SplitButton, ToggleSplitButton
- ✓ Open the WinUI 2 Gallery app and see the Button in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for these controls that uses rounded corners. For more info, see Corner radius.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:DropDownButton />
<muxc:SplitButton />
<muxc:ToggleSplitButton />
```

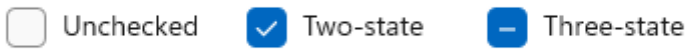
Related articles

- Button class
- Radio buttons
- Check boxes
- Toggle switches

Check boxes

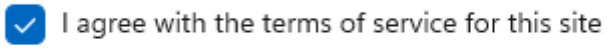
Article • 02/26/2025

A check box is used to select or deselect action items. It can be used for a single item or for a list of multiple items that a user can choose from. The control has three selection states: unselected, selected, and indeterminate. Use the indeterminate state when a collection of sub-choices have both unselected and selected states.



Is this the right control?

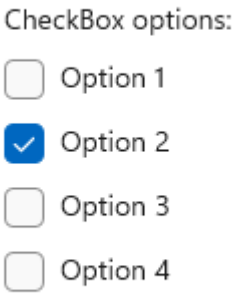
Use a **single check box** for a binary yes/no choice, such as with a "Remember me?" login scenario or with a terms of service agreement.



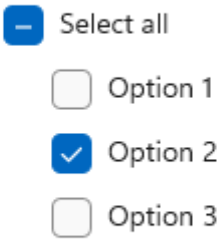
For a binary choice, the main difference between a **check box** and a toggle switch is that the check box is for status and the toggle switch is for action. You can delay committing a check box interaction (as part of a form submit, for example), while you should immediately commit a toggle switch interaction. Also, only check boxes allow for multi-selection.

Use **multiple check boxes** for multi-select scenarios in which a user chooses one or more items from a group of choices that are not mutually exclusive.

Create a group of check boxes when users can select any combination of options.



When options can be grouped, you can use an indeterminate check box to represent the whole group. Use the check box's indeterminate state when a user selects some, but not all, sub-items in the group.

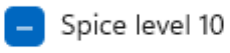


Both **check box** and **radio button** controls let the user select from a list of options. Check boxes let the user select a combination of options. In contrast, radio buttons let the user make a single choice from mutually exclusive options. When there is more than one option but only one can be selected, use a radio button instead.

Recommendations

- Verify that the purpose and current state of the check box is clear.
- Limit check box text content to no more than two lines.
- Word the checkbox label as a statement that the check mark makes true and the absence of a check mark makes false.
- Use the default font unless your brand guidelines tell you to use another.
- If the text content is dynamic, consider how the control will resize and what will happen to visuals around it.

- If there are two or more mutually exclusive options from which to choose, consider using radio buttons.
- Don't put two check box groups next to each other. Use group labels to separate the groups.
- Don't use a check box as an on/off control or to perform a command; instead, use a toggle switch.
- Don't use a check box to display other controls, such as a dialog box.
- Use the indeterminate state to indicate that an option is set for some, but not all, sub-choices.
- When using indeterminate state, use subordinate check boxes to show which options are selected and which are not. Design the UI so that the user can get see the sub-choices.
- Don't use the indeterminate state to represent a third state. The indeterminate state is used to indicate that an option is set for some, but not all, sub-choices. So, don't allow users to set an indeterminate state directly. For an example of what not to do, this check box uses the indeterminate state to indicate medium spiciness:



Instead, use a radio button group that has three options.

Create a checkbox

✓ **Important APIs**: CheckBox class, Checked event, IsChecked property, Content property

Open the WinUI 3 Gallery app and see the CheckBox in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source

> code on GitHub

Create a simple check box

To assign a label to the checkbox, set the Content property. The label displays next to the checkbox.

This XAML creates a single check box that is used to agree to terms of service before a form can be submitted.

XAML

```
<CheckBox x:Name="termsOfServiceCheckBox"
          Content="I agree to the terms of service."/>
```

Here's the same check box created in code.

C#

```
CheckBox termsOfServiceCheckBox = new CheckBox();
termsOfServiceCheckBox.Content = "I agree to the terms of service.";
```

Bind to IsChecked

Use the IsChecked property to determine whether the check box is checked or cleared. You can bind the value of the IsChecked property to another binary value. However, because IsChecked is a nullable boolean value, you must either use a cast or a value converter to bind it to a boolean property. This depends on the actual binding type you are using and you will find examples below for each possible type.

In this example, the **IsChecked** property of the check box to agree to terms of service is bound to the IsEnabled property of a Submit button. The Submit button is enabled only if the terms of service are agreed to.

Using x:Bind

> **Note**
>
> We only show the relevant code here. For more info about data binding see Data binding overview. Specific {x:Bind} info (such as casting) is detailed in {x:Bind} markup extension.

XAML

```
<StackPanel Grid.Column="2" Margin="40">
    <CheckBox x:Name="termsOfServiceCheckBox" Content="I agree to the terms
of service."/>
    <Button Content="Submit"
            IsEnabled="{x:Bind (x:Boolean)termsOfServiceCheckBox.IsChecked,
Mode=OneWay}"/>
</StackPanel>
```

If the check box can also be in the **indeterminate** state, we use the binding's FallbackValue property to specify the boolean value representing this state. In this case, we don't want to have the Submit button enabled as well:

XAML

```
<Button Content="Submit"
        IsEnabled="{x:Bind (x:Boolean)termsOfServiceCheckBox.IsChecked,
Mode=OneWay, FallbackValue=False}"/>
```

Using x:Bind or Binding

> **Note**
>
> We only show the relevant code here using {x:Bind}. In the {Binding} example, we would replace {x:Bind} with {Binding}. For more info about data binding, value converters and differences between the {x:Bind} and {Binding} markup extensions, see Data binding overview.

XAML

```
...
<Page.Resources>
    <local:NullableBooleanToBooleanConverter
x:Key="NullableBooleanToBooleanConverter"/>
</Page.Resources>

...

<StackPanel Grid.Column="2" Margin="40">
    <CheckBox x:Name="termsOfServiceCheckBox" Content="I agree to the terms
of service."/>
    <Button Content="Submit"
            IsEnabled="{x:Bind termsOfServiceCheckBox.IsChecked,
                       Converter={StaticResource
```

```
NullableBooleanToBooleanConverter}, Mode=OneWay}"/>
</StackPanel>
```

C#

```
public class NullableBooleanToBooleanConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter,
string language)
    {
        if (value is bool?)
        {
            return (bool)value;
        }
        return false;
    }

    public object ConvertBack(object value, Type targetType, object
parameter, string language)
    {
        if (value is bool)
            return (bool)value;
        return false;
    }
}
```

Handle Click and Checked events

To perform an action when the check box state changes, you can handle either the Click event, or the Checked and Unchecked events.

The **Click** event occurs whenever the checked state changes. If you handle the Click event, use the **IsChecked** property to determine the state of the check box.

The **Checked** and **Unchecked** events occur independently. If you handle these events, you should handle both of them to respond to state changes in the check box.

In the following examples, we show handling the Click event, and the Checked and Unchecked events.

Multiple checkboxes can share the same event handler. This example creates four checkboxes for selecting pizza toppings. The four checkboxes share the same **Click** event handler to update the list of selected toppings.

XAML

```
<StackPanel Margin="40">
    <TextBlock Text="Pizza Toppings"/>
```

```
    <CheckBox Content="Pepperoni" x:Name="pepperoniCheckbox"
              Click="toppingsCheckbox_Click"/>
    <CheckBox Content="Beef" x:Name="beefCheckbox"
              Click="toppingsCheckbox_Click"/>
    <CheckBox Content="Mushrooms" x:Name="mushroomsCheckbox"
              Click="toppingsCheckbox_Click"/>
    <CheckBox Content="Onions" x:Name="onionsCheckbox"
              Click="toppingsCheckbox_Click"/>

    <!-- Display the selected toppings. -->
    <TextBlock Text="Toppings selected:"/>
    <TextBlock x:Name="toppingsList"/>
</StackPanel>
```

Here's the event handler for the Click event. Every time a checkbox is clicked, it examines the checkboxes to see which ones are checked and updates the list of selected toppings.

C#

```
private void toppingsCheckbox_Click(object sender, RoutedEventArgs e)
{
    string selectedToppingsText = string.Empty;
    CheckBox[] checkboxes = new CheckBox[] { pepperoniCheckbox,
beefCheckbox,
                                             mushroomsCheckbox,
onionsCheckbox };
    foreach (CheckBox c in checkboxes)
    {
        if (c.IsChecked == true)
        {
            if (selectedToppingsText.Length > 1)
            {
                selectedToppingsText += ", ";
            }
            selectedToppingsText += c.Content;
        }
    }
    toppingsList.Text = selectedToppingsText;
}
```

Use the indeterminate state

The CheckBox control inherits from ToggleButton and can have three states:

Expand table

State	Property	Value
checked	IsChecked	**true**

State	Property	Value
unchecked	IsChecked	**false**
indeterminate	IsChecked	**null**

For the check box to report the indeterminate state, you must set the IsThreeState property to **true**.

When options can be grouped, you can use an indeterminate check box to represent the whole group. Use the check box's indeterminate state when a user selects some, but not all, sub-items in the group.

In the following example, the "Select all" checkbox has its IsThreeState property set to **true**. The "Select all" checkbox is checked if all child elements are checked, unchecked if all child elements are unchecked, and indeterminate otherwise.

XAML

```
<StackPanel>
    <CheckBox x:Name="OptionsAllCheckBox" Content="Select all"
IsThreeState="True"
              Checked="SelectAll_Checked" Unchecked="SelectAll_Unchecked"
              Indeterminate="SelectAll_Indeterminate"/>
    <CheckBox x:Name="Option1CheckBox" Content="Option 1" Margin="24,0,0,0"
              Checked="Option_Checked" Unchecked="Option_Unchecked" />
    <CheckBox x:Name="Option2CheckBox" Content="Option 2" Margin="24,0,0,0"
              Checked="Option_Checked" Unchecked="Option_Unchecked"
IsChecked="True"/>
    <CheckBox x:Name="Option3CheckBox" Content="Option 3" Margin="24,0,0,0"
              Checked="Option_Checked" Unchecked="Option_Unchecked" />
</StackPanel>
```

C#

```
private void Option_Checked(object sender, RoutedEventArgs e)
{
    SetCheckedState();
}

private void Option_Unchecked(object sender, RoutedEventArgs e)
{
    SetCheckedState();
}

private void SelectAll_Checked(object sender, RoutedEventArgs e)
{
    Option1CheckBox.IsChecked = Option2CheckBox.IsChecked =
Option3CheckBox.IsChecked = true;
}
```

```
private void SelectAll_Unchecked(object sender, RoutedEventArgs e)
{
    Option1CheckBox.IsChecked = Option2CheckBox.IsChecked =
Option3CheckBox.IsChecked = false;
}

private void SelectAll_Indeterminate(object sender, RoutedEventArgs e)
{
    // If the SelectAll box is checked (all options are selected),
    // clicking the box will change it to its indeterminate state.
    // Instead, we want to uncheck all the boxes,
    // so we do this programmatically. The indeterminate state should
    // only be set programmatically, not by the user.

    if (Option1CheckBox.IsChecked == true &&
        Option2CheckBox.IsChecked == true &&
        Option3CheckBox.IsChecked == true)
    {
        // This will cause SelectAll_Unchecked to be executed, so
        // we don't need to uncheck the other boxes here.
        OptionsAllCheckBox.IsChecked = false;
    }
}

private void SetCheckedState()
{
    // Controls are null the first time this is called, so we just
    // need to perform a null check on any one of the controls.
    if (Option1CheckBox != null)
    {
        if (Option1CheckBox.IsChecked == true &&
            Option2CheckBox.IsChecked == true &&
            Option3CheckBox.IsChecked == true)
        {
            OptionsAllCheckBox.IsChecked = true;
        }
        else if (Option1CheckBox.IsChecked == false &&
            Option2CheckBox.IsChecked == false &&
            Option3CheckBox.IsChecked == false)
        {
            OptionsAllCheckBox.IsChecked = false;
        }
        else
        {
            // Set third state (indeterminate) by setting IsChecked to null.
            OptionsAllCheckBox.IsChecked = null;
        }
    }
}
```

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the **Windows App SDK** and **WinUI 3**, but are generally applicable to UWP apps that use **WinUI 2**. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** CheckBox class, Checked event, IsChecked property, Content property
- ✓ Open the WinUI 2 Gallery app and see the CheckBox in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related articles

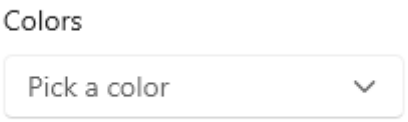
- CheckBox class
- Radio buttons
- Toggle switch

Combo box and list box

Article • 02/26/2025

Use a combo box (also known as a drop-down list) to present a list of items that a user can select from. A combo box starts in a compact state and expands to show a list of selectable items. A list box is similar to a combo box, but is not collapsible/does not have a compact state. You can learn more about list boxes at the end of this article.

When the combo box is closed, it either displays the current selection or is empty if there is no selected item. When the user expands the combo box, it displays the list of selectable items.

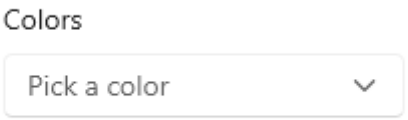


Is this the right control?

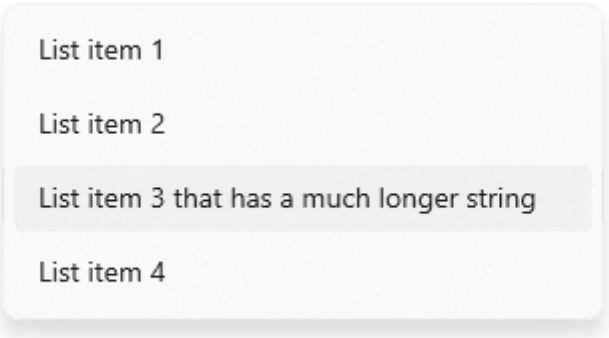
- Use a drop-down list to let users select a single value from a set of items that can be adequately represented with single lines of text.
- Use a list or grid view instead of a combo box to display items that contain multiple lines of text or images.
- When there are fewer than five items, consider using radio buttons (if only one item can be selected) or check boxes (if multiple items can be selected).
- Use a combo box when the selection items are of secondary importance in the flow of your app. If the default option is recommended for most users in most situations, showing all the items by using a list view might draw more attention to the options than necessary. You can save space and minimize distraction by using a combo box.

Examples

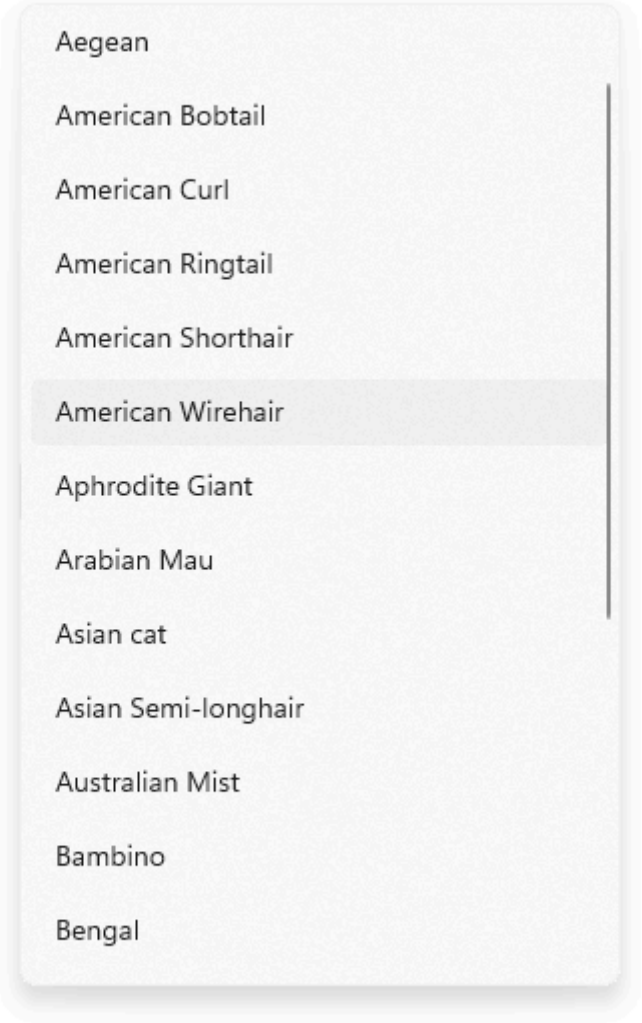
A combo box in its compact state can show a header.



Although combo boxes expand to support longer string lengths, avoid excessively long strings that are difficult to read.



If the collection in a combo box is long enough, a scroll bar will appear to accommodate it. Group items logically in the list.



Recommendations

- Limit the text content of combo box items to a single line.
- Sort items in a combo box in the most logical order. Group together related options and place the most common options at the top. Sort names in alphabetical order, numbers in numerical order, and dates in chronological order.

List boxes

A list box allows the user to choose either a single item or multiple items from a collection. List boxes are similar to drop-down lists, except that list boxes are always open—there is no compact (non-expanded) state for a list box. Items in the list can be scrolled if there isn't space to show everything.

Is a list box the right control?

- A list box can be useful when items in the list are important enough to prominently display, and when there's enough screen real estate, to show the full list.
- A list box should draw the user's attention to the full set of alternatives in an important choice. By contrast, a drop-down list initially draws the user's attention to the selected item.
- Avoid using a list box if:
  - There is a very small number of items for the list. A single-select list box that always has the same 2 options might be better presented as radio buttons. Also consider using radio buttons when there are 3 or 4 static items in the list.
  - The list box is single-select and it always has the same 2 options where one can be implied as not the other, such as "on" and "off." Use a single check box or a toggle switch.
  - There is a very large number of items. A better choice for long lists are grid view and list view. For very long lists of grouped data, semantic zoom is preferred.
  - The items are contiguous numerical values. If that's the case, consider using a slider.
  - The selection items are of secondary importance in the flow of your app or the default option is recommended for most users in most situations. Use a drop-down list instead.

Recommendations for list boxes

- The ideal range of items in a list box is 3 to 9.
- A list box works well when its items can dynamically vary.
- If possible, set the size of a list box so that its list of items don't need to be panned or scrolled.
- Verify that the purpose of the list box, and which items are currently selected, is clear.
- Reserve visual effects and animations for touch feedback, and for the selected state of items.
- Limit the list box item's text content to a single line. If the items are visuals, you can customize the size. If an item contains multiple lines of text or images, instead use a grid view or list view.
- Use the default font unless your brand guidelines indicate to use another.
- Don't use a list box to perform commands or to dynamically show or hide other controls.

Create a combo box

✓ **Important APIs**: ComboBox class, IsEditable property, Text property, TextSubmitted event, ListBox class

Open the WinUI 3 Gallery app and see the ComboBox in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

You populate the combo box by adding objects directly to the Items collection or by binding the ItemsSource property to a data source. Items added to the ComboBox are wrapped in ComboBoxItem containers.

Here's a simple combo box with items added in XAML.

XAML

```
<ComboBox Header="Colors" PlaceholderText="Pick a color" Width="200">
    <x:String>Blue</x:String>
    <x:String>Green</x:String>
    <x:String>Red</x:String>
    <x:String>Yellow</x:String>
</ComboBox>
```

The following example demonstrates binding a combo box to a collection of FontFamily objects.

XAML

```
<ComboBox x:Name="FontsCombo" Header="Fonts" Height="44" Width="296"
          ItemsSource="{x:Bind fonts}" DisplayMemberPath="Source"/>
```

C#

```
ObservableCollection<FontFamily> fonts = new
ObservableCollection<FontFamily>()
{
    fonts.Add(new FontFamily("Arial"));
    fonts.Add(new FontFamily("Courier New"));
    fonts.Add(new FontFamily("Times New Roman"));
};
```

Item selection

Like ListView and GridView, ComboBox is derived from Selector, so you can get the user's selection in the same standard way.

You can get or set the combo box's selected item by using the SelectedItem property, and get or set the index of the selected item by using the SelectedIndex property.

To get the value of a particular property on the selected data item, you can use the SelectedValue property. In this case, set the SelectedValuePath to specify which property on the selected item to get the value from.

> **Tip**
>
> If you set SelectedItem or SelectedIndex to indicate the default selection, an exception occurs if the property is set before the combo box Items collection is populated. Unless you define your Items in XAML, it's best to handle the combo box Loaded event, and set SelectedItem or SelectedIndex in the Loaded event handler.

You can bind to these properties in XAML, or handle the SelectionChanged event to respond to selection changes.

In the event handler code, you can get the selected item from the SelectionChangedEventArgs.AddedItems property. You can get the previously selected item (if any) from the SelectionChangedEventArgs.RemovedItems property. The AddedItems and RemovedItems collections each contain only 1 item because combo box does not support multiple selection.

This example shows how to handle the SelectionChanged event, and also how to bind to the selected item.

XAML

```
<StackPanel>
    <ComboBox x:Name="colorComboBox" Width="200"
              Header="Colors" PlaceholderText="Pick a color"
              SelectionChanged="ColorComboBox_SelectionChanged">
        <x:String>Blue</x:String>
        <x:String>Green</x:String>
        <x:String>Red</x:String>
        <x:String>Yellow</x:String>
    </ComboBox>

    <Rectangle x:Name="colorRectangle" Height="30" Width="100"
               Margin="0,8,0,0" HorizontalAlignment="Left"/>

    <TextBlock Text="{x:Bind colorComboBox.SelectedIndex, Mode=OneWay}"/>
```

```
    <TextBlock Text="{x:Bind colorComboBox.SelectedItem, Mode=OneWay}"/>
</StackPanel>
```

C#

```
private void ColorComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    string colorName = e.AddedItems[0].ToString();
    Color color;
    switch (colorName)
    {
        case "Yellow":
            color = Colors.Yellow;
            break;
        case "Green":
            color = Colors.Green;
            break;
        case "Blue":
            color = Colors.Blue;
            break;
        case "Red":
            color = Colors.Red;
            break;
    }
    colorRectangle.Fill = new SolidColorBrush(color);
}
```

SelectionChanged and keyboard navigation

By default, the SelectionChanged event occurs when a user clicks, taps, or presses Enter on an item in the list to commit their selection, and the combo box closes. Selection doesn't change when the user navigates the open combo box list with the keyboard arrow keys.

To make a combo box that "live updates" while the user is navigating the open list with the arrow keys (like a Font selection drop-down), set SelectionChangedTrigger to Always. This causes the SelectionChanged event to occur when focus changes to another item in the open list.

Selected item behavior change

In Windows 10, version 1809 (SDK 17763) or later, the behavior of selected items is updated to support editable combo boxes.

Prior to SDK 17763, the value of the SelectedItem property (and therefore, SelectedValue and SelectedIndex) was required to be in the combo box's Items collection. Using the previous example, setting `colorComboBox.SelectedItem = "Pink"` results in:

- SelectedItem = null
- SelectedValue = null
- SelectedIndex = -1

In SDK 17763 and later, the value of the SelectedItem property (and therefore, SelectedValue and SelectedIndex) is not required to be in the combo box's Items collection. Using the previous example, setting `colorComboBox.SelectedItem = "Pink"` results in:

- SelectedItem = Pink
- SelectedValue = Pink
- SelectedIndex = -1

Text Search

Combo boxes automatically support search within their collections. As users type characters on a physical keyboard while focused on an open or closed combo box, candidates matching the user's string are brought into view. This functionality is especially helpful when navigating a long list. For example, when interacting with a drop-down containing a list of states, users can press the "w" key to bring "Washington" into view for quick selection. The text search is not case-sensitive.

You can set the IsTextSearchEnabled property to **false** to disable this functionality.

Make a combo box editable

By default, a combo box lets the user select from a pre-defined list of options. However, there are cases where the list contains only a subset of valid values, and the user should be able to enter other values that aren't listed. To support this, you can make the combo box editable.

To make a combo box editable, set the IsEditable property to **true**. Then, handle the TextSubmitted event to work with the value entered by the user.

By default, the SelectedItem value is updated when the user commits custom text. You can override this behavior by setting **Handled** to **true** in the TextSubmitted event args. When the event is marked as handled, the combo box will take no further action after the event and will stay in the editing state. SelectedItem will not be updated.

This example shows a simple editable combo box. The list contains simple strings, and any value entered by the user is used as entered.

A "recently used names" chooser lets the user enter custom strings. The 'RecentlyUsedNames' list contains some values that the user can choose from, but the user can also add a new, custom value. The 'CurrentName' property represents the currently entered name.

XAML

```
<ComboBox IsEditable="true"
          ItemsSource="{x:Bind RecentlyUsedNames}"
          SelectedItem="{x:Bind CurrentName, Mode=TwoWay}"/>
```

Text submitted

You can handle the TextSubmitted event to work with the value entered by the user. In the event handler, you will typically validate that the value entered by the user is valid, then use the value in your app. Depending on the situation, you might also add the value to the combo box's list of options for future use.

The TextSubmitted event occurs when these conditions are met:

- The IsEditable property is **true**
- The user enters text that does not match an existing entry in the combo box list
- The user presses Enter, or moves focus from the combo box.

The TextSubmitted event does not occur if the user enters text and then navigates up or down through the list.

Sample - Validate input and use locally

In this example, a font size chooser contains a set of values corresponding to the font size ramp, but the user may enter font sizes that are not in the list.

When the user adds a value that's not in the list, the font size updates, but the value is not added to the list of font sizes.

If the newly entered value is not valid, you use the SelectedValue to revert the Text property to the last known good value.

XAML

```
<ComboBox x:Name="fontSizeComboBox"
          IsEditable="true"
          ItemsSource="{x:Bind ListOfFontSizes}"
          TextSubmitted="FontSizeComboBox_TextSubmitted"/>
```

C#

```
private void FontSizeComboBox_TextSubmitted(ComboBox sender,
ComboBoxTextSubmittedEventArgs e)
{
    if (byte.TryParse(e.Text, out double newValue))
    {
        // Update the app's font size.
        _fontSize = newValue;
    }
    else
    {
        // If the item is invalid, reject it and revert the text.
        // Mark the event as handled so the framework doesn't update the
selected item.
        sender.Text = sender.SelectedValue.ToString();
        e.Handled = true;
    }
}
```

Sample - Validate input and add to list

Here, a "favorite color chooser" contains the most common favorite colors (Red, Blue, Green, Orange), but the user may enter a favorite color that's not in the list. When the user adds a valid color (like Pink), the newly entered color is added to the list and set as the active "favorite color".

XAML

```
<ComboBox x:Name="favoriteColorComboBox"
          IsEditable="true"
          ItemsSource="{x:Bind ListOfColors}"
          TextSubmitted="FavoriteColorComboBox_TextSubmitted"/>
```

C#

```
private void FavoriteColorComboBox_TextSubmitted(ComboBox sender,
ComboBoxTextSubmittedEventArgs e)
{
    if (IsValid(e.Text))
    {
        FavoriteColor newColor = new FavoriteColor()
```

```
        {
            ColorName = e.Text,
            Color = ColorFromStringConverter(e.Text)
        }
        ListOfColors.Add(newColor);
    }
    else
    {
        // If the item is invalid, reject it but do not revert the text.
        // Mark the event as handled so the framework doesn't update the
selected item.
        e.Handled = true;
    }
}

bool IsValid(string Text)
{
    // Validate that the string is: not empty; a color.
}
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** ComboBox class, IsEditable property, Text property, TextSubmitted event, ListBox class
- ✓ Open the WinUI 2 Gallery app and see the ComboBox in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

> **ⓘ Note**
>
> The IsEditable property requires Windows 10, version 1809 (SDK 17763) or later.

WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related articles

- Text controls
- Guidelines for spell checking
- TextBox class
- PasswordBox class
- String.Length property

Hyperlinks

Article • 02/26/2025

Hyperlinks navigate the user to another part of the app, to another app, or launch a specific uniform resource identifier (URI) using a separate browser app. There are two ways that you can add a hyperlink to a XAML app: the **Hyperlink** text element and **HyperlinkButton** control.

Is this the right control?

Use a hyperlink when you need text that responds when pressed and navigates the user to more information about the text that was pressed.

Choose the right type of hyperlink based on your needs:

- Use an inline **Hyperlink** text element inside of a text control. A Hyperlink element flows with other text elements and you can use it in any InlineCollection. Use a text hyperlink if you want automatic text wrapping and don't necessarily need a large hit target. Hyperlink text can be small and difficult to target, especially for touch.
- Use a **HyperlinkButton** for stand-alone hyperlinks. A HyperlinkButton is a specialized Button control that you can use anywhere that you would use a Button.
- Use a **HyperlinkButton** with an Image as its content to make a clickable image.

Recommendations

- Only use hyperlinks for navigation; don't use them for other actions.
- Use the Body style from the type ramp for text-based hyperlinks. Read about fonts and the Windows type ramp.
- Keep discrete hyperlinks far enough apart so that the user can differentiate between them and has an easy time selecting each one.
- Add tooltips to hyperlinks that indicate to where the user will be directed. If the user will be directed to an external site, include the top-level domain name inside the tooltip, and style the text with a secondary font color.

Create a Hyperlink text element

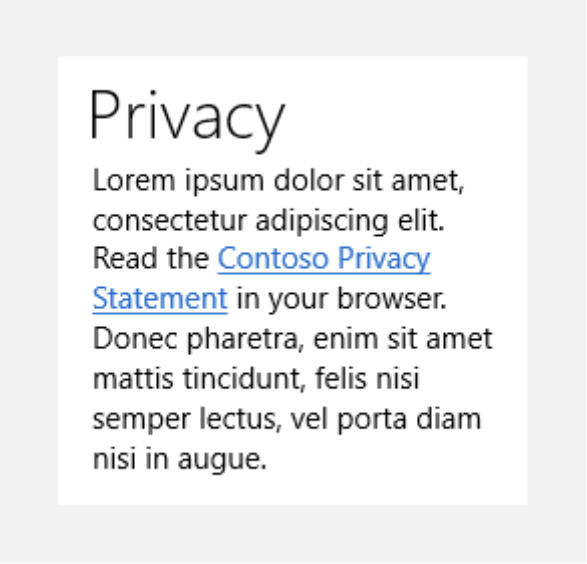
✔ **Important APIs**: Hyperlink text element

This example shows how to use a Hyperlink text element inside of a TextBlock.

XAML

```
<StackPanel Width="200">
    <TextBlock Text="Privacy" Style="{StaticResource
SubheaderTextBlockStyle}"/>
    <TextBlock TextWrapping="WrapWholeWords">
        <Span xml:space="preserve"><Run>Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Read the </Run><Hyperlink
NavigateUri="http://www.contoso.com">Contoso Privacy Statement</Hyperlink>
<Run> in your browser.</Run> Donec pharetra, enim sit amet mattis tincidunt,
felis nisi semper lectus, vel porta diam nisi in augue.</Span>
    </TextBlock>
</StackPanel>
```

The hyperlink appears inline and flows with the surrounding text:



Tip

When you use a Hyperlink in a text control with other text elements in XAML, place the content in a **Span** container and apply the `xml:space="preserve"` attribute to the Span to keep the white space between the Hyperlink and other elements.

Create a HyperlinkButton

✔ **Important APIs**: HyperlinkButton control

Open the WinUI 3 Gallery app and see the Hyperlinks in action

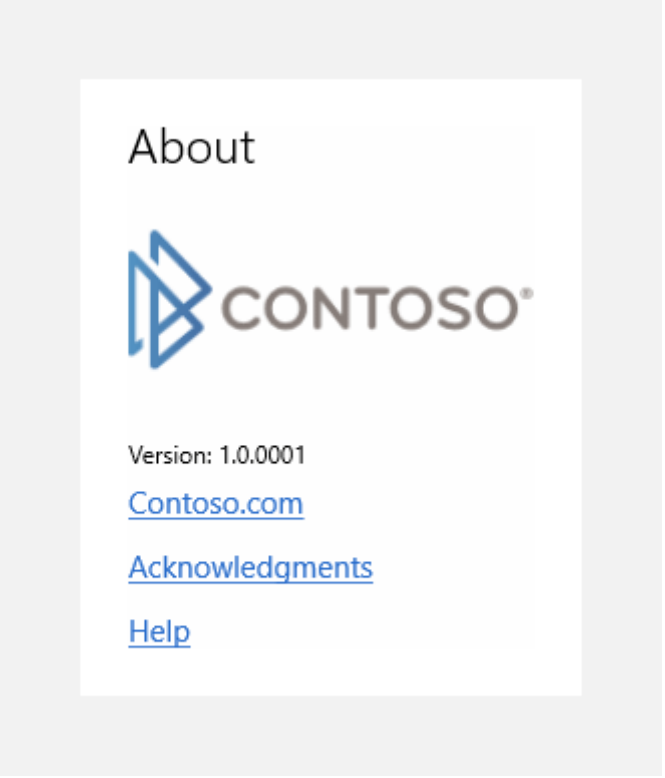
> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Here's how to use a HyperlinkButton, both with text and with an image.

XAML

```
<StackPanel>
    <TextBlock Text="About" Style="{StaticResource TitleTextBlockStyle}"/>
    <HyperlinkButton NavigateUri="http://www.contoso.com">
        <Image Source="Assets/ContosoLogo.png"/>
    </HyperlinkButton>
    <TextBlock Text="Version: 1.0.0001" Style="{StaticResource
CaptionTextBlockStyle}"/>
    <HyperlinkButton Content="Contoso.com"
NavigateUri="http://www.contoso.com"/>
    <HyperlinkButton Content="Acknowledgments"
NavigateUri="http://www.contoso.com"/>
    <HyperlinkButton Content="Help" NavigateUri="http://www.contoso.com"/>
</StackPanel>
```

The hyperlink buttons with text content appear as marked-up text. The Contoso logo image is also a clickable hyperlink:



This example shows how to create a HyperlinkButton in code.

```
C#

HyperlinkButton helpLinkButton = new HyperlinkButton();
helpLinkButton.Content = "Help";
helpLinkButton.NavigateUri = new Uri("http://www.contoso.com");
```

Handle navigation

For both kinds of hyperlinks, you handle navigation the same way; you can set the **NavigateUri** property, or handle the **Click** event.

Navigate to a URI

To use the hyperlink to navigate to a URI, set the NavigateUri property. When a user clicks or taps the hyperlink, the specified URI opens in the default browser. The default browser runs in a separate process from your app.

> ⓘ **Note**
>
> A URI is represented by the Windows.Foundation.Uri class. When programming with .NET, this class is hidden and you should use the System.Uri class. For more info, see the reference pages for these classes.

You don't have to use **http:** or **https:** schemes. You can use schemes such as **ms-appx:**, **ms-appdata:**, or **ms-resources:**, if there's resource content at these locations that's appropriate to load in a browser. However, the **file:** scheme is specifically blocked. For more info, see URI schemes.

When a user clicks the hyperlink, the value of the NavigateUri property is passed to a system handler for URI types and schemes. The system then launches the app that is registered for the scheme of the URI provided for NavigateUri.

If you don't want the hyperlink to load content in a default Web browser (and don't want a browser to appear), then don't set a value for NavigateUri. Instead, handle the Click event, and write code that does what you want.

Handle the Click event

Use the Click event for actions other than launching a URI in a browser, such as navigation within the app. For example, if you want to load a new app page rather than opening a browser, call a Frame.Navigate method within your Click event handler to navigate to the new app page. If you want an external, absolute URI to load within a WebView control that also exists in your app, call WebView.Navigate as part of your Click handler logic.

You don't typically handle the Click event as well as specifying a NavigateUri value, as these represent two different ways of using the hyperlink element. If your intent is to open the URI in the default browser, and you have specified a value for NavigateUri, don't handle the Click event. Conversely, if you handle the Click event, don't specify a NavigateUri.

There's nothing you can do within the Click event handler to prevent the default browser from loading any valid target specified for NavigateUri; that action takes place automatically (asynchronously) when the hyperlink is activated and can't be canceled from within the Click event handler.

Hyperlink underlines

By default, hyperlinks are underlined. This underline is important because it helps meet accessibility requirements. Color-blind users use the underline to distinguish between hyperlinks and other text. If you disable underlines, you should consider adding some other type of formatting difference to distinguish hyperlinks from other text, such as FontWeight or FontStyle.

Hyperlink text elements

You can set the UnderlineStyle property to disable the underline. If you do, consider using FontWeight or FontStyle to differentiate your link text.

HyperlinkButton

By default, the HyperlinkButton appears as underlined text when you set a string as the value for the Content property.

The text does not appear underlined in the following cases:

- You set a TextBlock as the value for the Content property, and set the Text property on the TextBlock.
- You re-template the HyperlinkButton and change the name of the ContentPresenter template part.

If you need a button that appears as non-underlined text, consider using a standard Button control and applying the built-in `TextBlockButtonStyle` system resource to its Style property.

Notes for Hyperlink text element

This section applies only to the Hyperlink text element, not to the HyperlinkButton control.

Input events

Because a Hyperlink is not a UIElement, it does not have the set of UI element input events such as Tapped, PointerPressed, and so on. Instead, a Hyperlink has its own Click event, plus the implicit behavior of the system loading any URI specified as the NavigateUri. The system handles all input actions that should invoke the Hyperlink actions and raises the Click event in response.

Content

Hyperlink has restrictions on the content that can exist in its Inlines collection. Specifically, a Hyperlink only permits Run and other Span types that aren't another Hyperlink. InlineUIContainer can't be in the Inlines collection of a Hyperlink. Attempting to add restricted content throws an invalid argument exception or XAML parse exception.

Hyperlink and theme/style behavior

Hyperlink doesn't inherit from Control, so it doesn't have a Style property or a Template. You can edit the properties that are inherited from TextElement, such as Foreground or FontFamily, to change the appearance of a Hyperlink, but you can't use a common style or template to apply changes. Instead of using a template, consider using common resources for values of Hyperlink properties to provide consistency. Some properties of Hyperlink use defaults from a {ThemeResource} markup extension value provided by the system. This enables the Hyperlink appearance to switch in appropriate ways when the user changes the system theme at run-time.

The default color of the hyperlink is the accent color of the system. You can set the Foreground property to override this.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** Hyperlink text element, HyperlinkButton control
- ✓ Open the WinUI 2 Gallery app and see Hyperlinks in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

Related articles

- Text controls
- Guidelines for tooltips
- Hyperlink class
- HyperlinkButton class

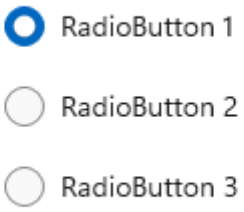
Radio buttons

Article • 02/26/2025

Radio buttons, also called option buttons, let users select one option from a collection of two or more mutually exclusive, but related, options. Radio buttons are always used in groups, and each option is represented by one radio button in the group.

In the default state, no radio button in a RadioButtons group is selected. That is, all radio buttons are cleared. However, once a user has selected a radio button, the user can't deselect the button to restore the group to its initial cleared state.

The singular behavior of a RadioButtons group distinguishes it from check boxes, which support multi-selection and deselection, or clearing.

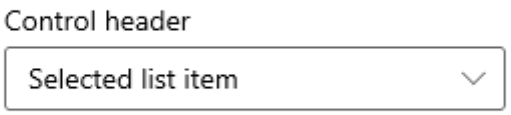


Is this the right control?

Use radio buttons to let users select from two or more mutually exclusive options.

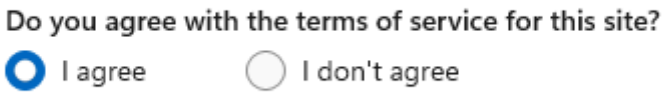
Use radio buttons when users need to see all options before they make a selection. Radio buttons emphasize all options equally, which means that some options might draw more attention than is necessary or desired.

Unless all options deserve equal attention, consider using other controls. For example, to recommend a single best option for most users and in most situations, use a combo box to display that best option as the default option.

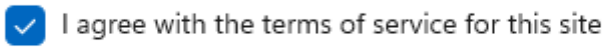


If there are only two possible options that can be expressed clearly as a single binary choice, such as on/off or yes/no, combine them into a single check box or toggle switch control. For example, use a single check box for "I agree" instead of two radio buttons for "I agree" and "I don't agree."

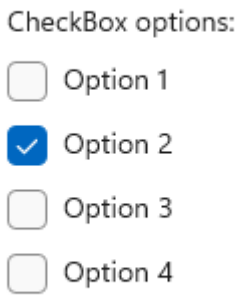
Don't use two radio buttons for a single binary choice:



Use a check box instead:

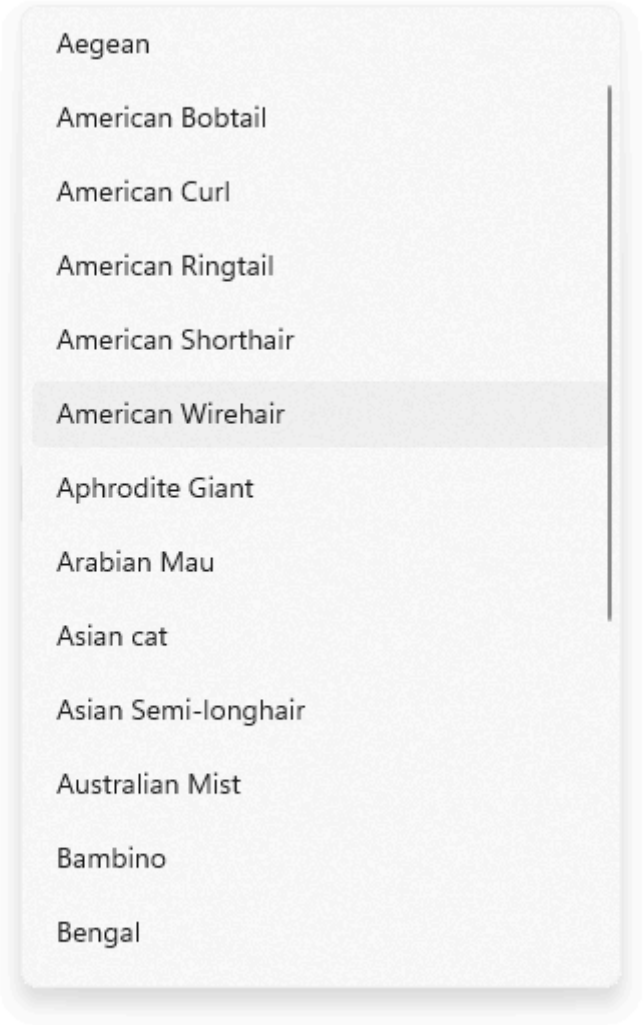


When users can select multiple options, use check boxes.



When users' options lie within a range of values (for example, *10, 20, 30, ... 100*), use a slider control.

If there are more than eight options, use a combo box.



If the available options are based on an app's current context, or they can otherwise vary dynamically, use a list control.

Recommendations

- Make sure that the purpose and current state of a set of radio buttons is explicit.
- Limit the radio button's text label to a single line.
- If the text label is dynamic, consider how the button will automatically resize and what will happen to any visuals around it.

- Use the default font unless your brand guidelines tell you otherwise.
- Don't put two RadioButtons groups side by side. When two RadioButtons groups are right next to each other, it can be difficult for users to determine which buttons belong to which group.

RadioButtons overview

RadioButtons vs RadioButton

There are two ways to create radio button groups: RadioButtons and RadioButton.

- We recommend the **RadioButtons** control. This control simplifies layout, handles keyboard navigation and accessibility, and supports binding to a data source.
- You can use groups of individual **RadioButton** controls.

Keyboard access and navigation behavior have been optimized in the RadioButtons control. These improvements help both accessibility and keyboard power users move through the list of options more quickly and easily.

In addition to these improvements, the default visual layout of individual radio buttons in a RadioButtons group has been optimized through automated orientation, spacing, and margin settings. This optimization eliminates the requirement to specify these properties, as you might have to do when you use a more primitive grouping control, such as StackPanel or Grid.

Navigating a RadioButtons group

The `RadioButtons` control has special navigation behavior that helps keyboard users navigate the list more quickly and more easily.

Keyboard focus

The `RadioButtons` control supports two states:

- No radio button is selected
- One radio button is selected

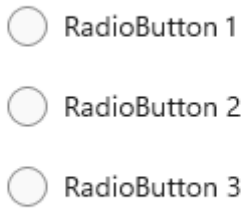
The next sections describe the focus behavior of the control in each state.

No radio button is selected

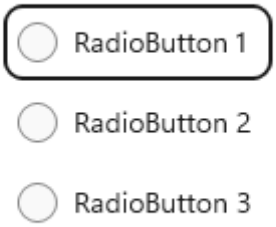
When no radio button is selected, the first radio button in the list gets focus.

> **Note**
>
> The item that receives tab focus from the initial tab navigation is not selected.

List without tab focus, no selection



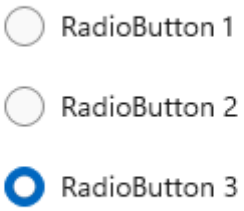
List with initial tab focus, no selection



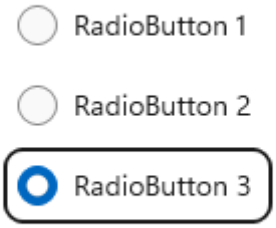
One radio button is selected

When a user tabs into the list where a radio button is already selected, the selected radio button gets focus.

List without tab focus



List with initial tab focus



Keyboard navigation

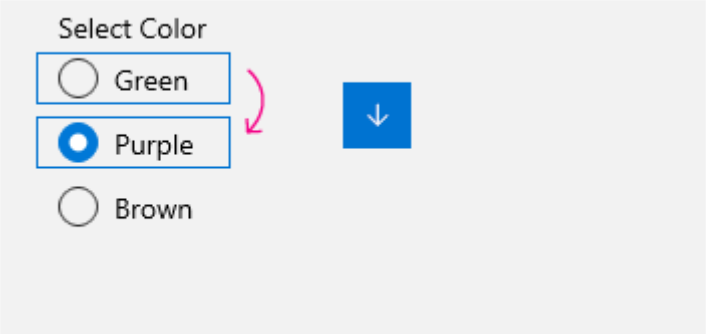
For more information about general keyboard navigation behaviors, see Keyboard interactions - Navigation.

When an item in a `RadioButtons` group already has focus, the user can use arrow keys for "inner navigation" between the items within the group. The Up and Down arrow keys move to the "next" or "previous" logical item, as defined in your XAML markup. The Left and Right arrow keys move spatially.

Navigation within single-column or single-row layouts

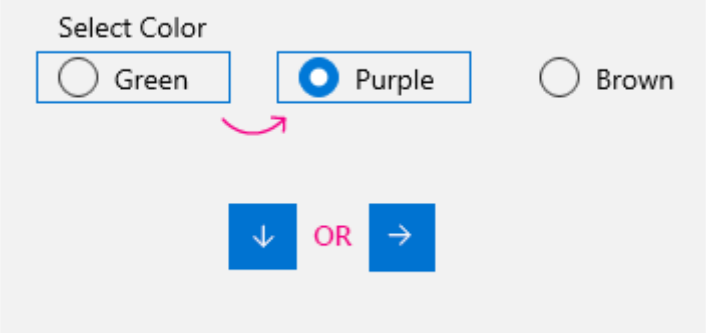
In a single-column or single-row layout, keyboard navigation results in the following behavior:

Single column



The Up arrow and Down arrow keys move between items.
The Left arrow and Right arrow keys do nothing.

Single row



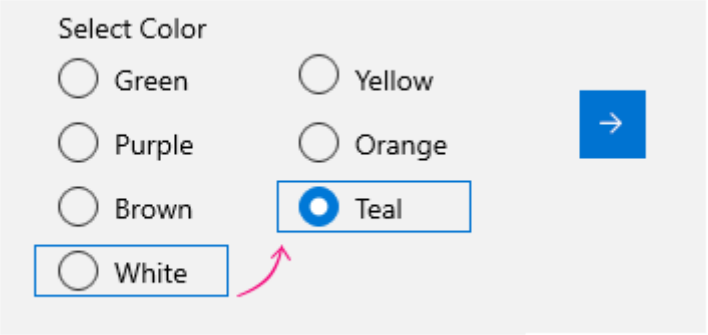
The Left and Up arrow keys move to the previous item, and the Right and Down arrow keys move to the next item.

Navigation within multi-column, multi-row layouts

In a multi-column, multi-row grid layout, keyboard navigation results in this behavior:

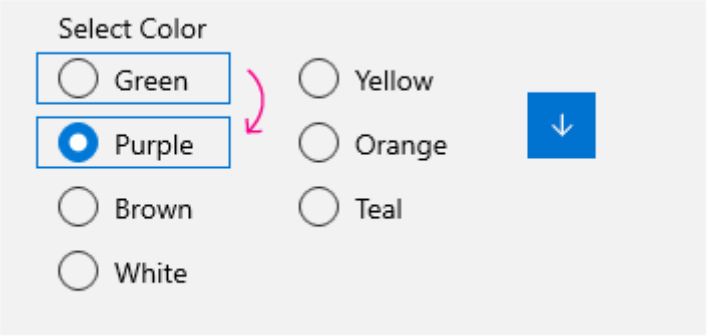
Left/Right arrow keys

The Left and Right arrow keys move focus horizontally between items in a row.



When focus is on the last item in a column and the Right or Left arrow key is pressed, focus moves to the last item in the next or previous column (if any).

Up/Down arrow keys



The Up and Down arrow keys move focus vertically between items in a column.

When focus is on the last item in a column and the Down arrow key is pressed, focus moves to the first item in the next column (if any). When focus is on the first item in a column and the Up arrow key is pressed, focus moves to the last item in the previous column (if any)

For more information, see Keyboard interactions.

Wrapping

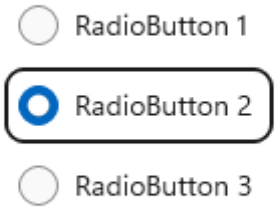
The RadioButtons group doesn't wrap focus from the first row or column to the last, or from the last row or column to the first. This is because, when users use a screen reader, a sense of boundary and a clear indication of beginning and end is lost, which makes it difficult for users with visual impairment to navigate the list.

The `RadioButtons` control also doesn't support enumeration, because the control is intended to contain a reasonable number of items (see Is this the right control?).

Selection follows focus

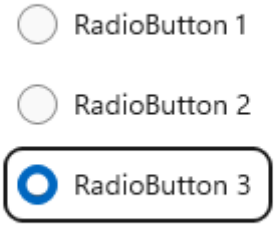
When you use the keyboard to navigate between items in a `RadioButtons` group, as focus moves from one item to the next, the newly focused item gets selected and the previously focused item is cleared.

Before keyboard navigation



Focus and selection before keyboard navigation.

After keyboard navigation



Focus and selection after keyboard navigation, where the Down arrow key moves focus to radio button 3, selects it, and clears radio button 2.

You can move focus without changing selection by using Ctrl+arrow keys to navigate. After focus is moved, you can use the Spacebar to select the item that currently has focus.

Navigating with game pad and remote control

If you use a game pad or remote control to move between radio buttons, the "selection follows focus" behavior is disabled, and the user must press the "A" button to select the radio button that currently has focus.

Accessibility behavior

The following table describes how Narrator handles a `RadioButtons` group and what is announced. This behavior depends on how a user has set the Narrator detail preferences.

Expand table

Action	Narrator announcement
Focus moves to a selected item	"*name*, RadioButton, selected, *x* of *N*"
Focus moves to an unselected item *(If navigating with Ctrl-arrow keys or Xbox gamepad, which indicates selection is not following focus.)*	"*name*, RadioButton, non-selected, *x* of *N*"

> **Note**
>
> The ***name*** that Narrator announces for each item is the value of the AutomationProperties.Name attached property if it is available for the item; otherwise, it is the value returned by the item's ToString method.
>
> ***x*** is the number of the current item. ***N*** is the total number of items in the group.

Create a WinUI RadioButtons group

✓ **Important APIs**: RadioButtons class, SelectedItem property, SelectedIndex property, SelectionChanged event, RadioButton class, IsChecked property, Checked event

Open the WinUI 3 Gallery app and see the RadioButton in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

The `RadioButtons` control uses a content model similar to an ItemsControl. This means that you can:

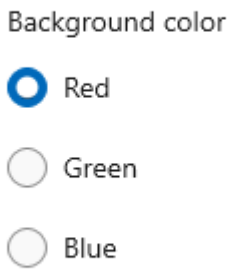
- Populate it by adding items directly to the Items collection or by binding data to its ItemsSource property.
- Use the SelectedIndex or SelectedItem properties to get and set which option is selected.
- Handle the SelectionChanged event to take action when an option is chosen.

Here, you declare a simple `RadioButtons` control with three options. The Header property is set to give the group a label, and the `SelectedIndex` property is set to provide a default option.

XAML

```
<RadioButtons Header="Background color"
              SelectedIndex="0"
              SelectionChanged="BackgroundColor_SelectionChanged">
    <x:String>Red</x:String>
    <x:String>Green</x:String>
    <x:String>Blue</x:String>
</RadioButtons>
```

The result looks like this:



To take an action when the user selects an option, handle the SelectionChanged event. Here, you change the background color of a Border element named "ExampleBorder" (`<Border x:Name="ExampleBorder" Width="100" Height="100"/>`).

C#

```
private void BackgroundColor_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (ExampleBorder != null && sender is RadioButtons rb)
    {
        string colorName = rb.SelectedItem as string;
        switch (colorName)
        {
            case "Red":
                ExampleBorder.Background = new SolidColorBrush(Colors.Red);
                break;
            case "Green":
                ExampleBorder.Background = new
SolidColorBrush(Colors.Green);
                break;
            case "Blue":
                ExampleBorder.Background = new SolidColorBrush(Colors.Blue);
                break;
        }
    }
}
```

Tip

You can also get the selected item from the SelectionChangedEventArgs.AddedItems property. There will only be one selected item, at index 0, so you could get the selected item like this: `string colorName = e.AddedItems[0] as string;`.

Selection states

A radio button has two states: selected or cleared. When an option is selected in a `RadioButtons` group, you can get its value from the SelectedItem property, and its location in the collection from the SelectedIndex property. A radio button can be cleared if a user selects another radio button in the same group, but it can't be cleared if the user selects it again. However, you can clear a radio button group programmatically by setting it `SelectedItem = null`, or `SelectedIndex = -1`. (Setting `SelectedIndex` to any value outside the range of the `Items` collection results in no selection.)

RadioButtons content

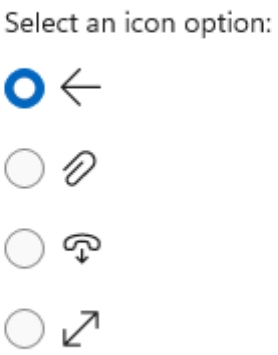
In the previous example, you populated the `RadioButtons` control with simple strings. The control provided the radio buttons, and used the strings as the label for each one.

However, you can populate the `RadioButtons` control with any object. Typically, you want the object to provide a string representation that can be used as a text label. In some cases, an image might be appropriate in place of text.

Here, SymbolIcon elements are used to populate the control.

XAML

```
<RadioButtons Header="Select an icon option:">
    <SymbolIcon Symbol="Back"/>
    <SymbolIcon Symbol="Attach"/>
    <SymbolIcon Symbol="HangUp"/>
    <SymbolIcon Symbol="FullScreen"/>
</RadioButtons>
```



You can also use individual RadioButton controls to populate the `RadioButtons` items. This is a special case that we discuss later. See RadioButton controls in a RadioButtons group.

A benefit of being able to use any object is that you can bind the `RadioButtons` control to a custom type in your data model. The next section demonstrates this.

Data binding

The `RadioButtons` control supports data binding to its ItemsSource property. This example shows how you can bind the control to a custom data source. The appearance and functionality of this example is the same as the previous background color example, but here, the color brushes are stored in the data model instead of being created in the `SelectionChanged` event handler.

XAML

```
<RadioButtons Header="Background color"
              SelectedIndex="0"
              SelectionChanged="BackgroundColor_SelectionChanged"
              ItemsSource="{x:Bind colorOptionItems}"/>
```

C#

```
public sealed partial class MainPage : Page
{
    // Custom data item.
    public class ColorOptionDataModel
    {
        public string Label { get; set; }
        public SolidColorBrush ColorBrush { get; set; }

        public override string ToString()
        {
            return Label;
        }
    }

    List<ColorOptionDataModel> colorOptionItems;

    public MainPage1()
    {
        this.InitializeComponent();

        colorOptionItems = new List<ColorOptionDataModel>();
        colorOptionItems.Add(new ColorOptionDataModel()
            { Label = "Red", ColorBrush = new SolidColorBrush(Colors.Red)
});
        colorOptionItems.Add(new ColorOptionDataModel()
            { Label = "Green", ColorBrush = new
SolidColorBrush(Colors.Green) });
        colorOptionItems.Add(new ColorOptionDataModel()
            { Label = "Blue", ColorBrush = new SolidColorBrush(Colors.Blue)
});
    }

    private void BackgroundColor_SelectionChanged(object sender,
SelectionChangedEventArgs e)
    {
        var option = e.AddedItems[0] as ColorOptionDataModel;
        ExampleBorder.Background = option?.ColorBrush;
    }
}
```

RadioButton controls in a RadioButtons group

You can use individual RadioButton controls to populate the `RadioButtons` items. You might do this to get access to certain properties, like `AutomationProperties.Name`; or you might have existing `RadioButton` code, but want to take advantage of the layout and navigation of `RadioButtons`.

XAML

```
<RadioButtons Header="Background color">
    <RadioButton Content="Red" Tag="red" AutomationProperties.Name="red"/>
    <RadioButton Content="Green" Tag="green"
AutomationProperties.Name="green"/>
    <RadioButton Content="Blue" Tag="blue"
AutomationProperties.Name="blue"/>
</RadioButtons>
```

When you use `RadioButton` controls in a `RadioButtons` group, the `RadioButtons` control knows how to present the `RadioButton`, so you won't end up with two selection circles.

However, there are some behaviors you should be aware of. We recommend that you handle state and events on the individual controls or on `RadioButtons`, but not both, to avoid conflicts.

This table shows the related events and properties on both controls.

Expand table

RadioButton	RadioButtons
Checked, Unchecked, Click	SelectionChanged
IsChecked	SelectedItem, SelectedIndex

If you handle events on an individual `RadioButton`, such as `Checked` or `Unchecked`, and also handle the `RadioButtons.SelectionChanged` event, both events will fire. The `RadioButton` event occurs first, and then the `RadioButtons.SelectionChanged` event occurs, which could result in conflicts.

The `IsChecked`, `SelectedItem`, and `SelectedIndex` properties stay synchronized. A change to one property updates the other two.

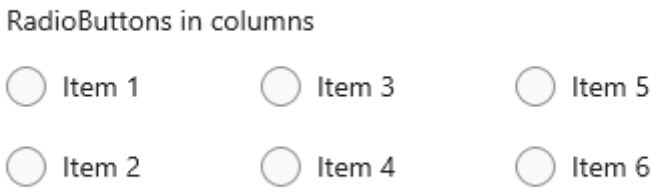
The RadioButton.GroupName property is ignored. The group is created by the `RadioButtons` control.

Defining multiple columns

By default, the `RadioButtons` control arranges its radio buttons vertically in a single column. You can set the MaxColumns property to make the control arrange the radio buttons in multiple columns. (When you do this, they are laid out in column-major order, where items fill in from top to bottom, then left to right.)

XAML

```
<RadioButtons Header="RadioButtons in columns" MaxColumns="3">
    <x:String>Item 1</x:String>
    <x:String>Item 2</x:String>
    <x:String>Item 3</x:String>
    <x:String>Item 4</x:String>
    <x:String>Item 5</x:String>
    <x:String>Item 6</x:String>
</RadioButtons>
```



> **Tip**
>
> To have items arranged in a single horizontal row, set `MaxColumns` equal to the number of items in the group.

Create your own RadioButton group

> **Important**
>
> We recommend using the `RadioButtons` control to group `RadioButton` elements.

Radio buttons work in groups. You can group individual RadioButton controls in either of two ways:

- Put them inside the same parent container.
- Set the GroupName property on each radio button to the same value.

In this example, the first group of radio buttons is implicitly grouped by being in the same stack panel. The second group is divided between two stack panels, so `GroupName` is used to explicitly group them into a single group.

XAML

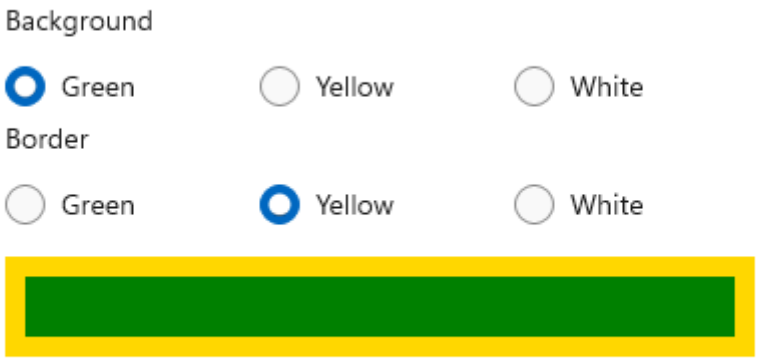
```
<StackPanel>
    <StackPanel>
        <TextBlock Text="Background" Style="{ThemeResource
BaseTextBlockStyle}"/>
        <!-- Group 1 - implicit grouping -->
        <StackPanel Orientation="Horizontal">
            <RadioButton Content="Green" Tag="green"
Checked="BGRadioButton_Checked"/>
            <RadioButton Content="Yellow" Tag="yellow"
Checked="BGRadioButton_Checked"/>
            <RadioButton Content="White" Tag="white"
Checked="BGRadioButton_Checked"
                         IsChecked="True"/>
        </StackPanel>
    </StackPanel>

    <StackPanel>
        <TextBlock Text="BorderBrush" Style="{ThemeResource
BaseTextBlockStyle}"/>
        <!-- Group 2 - grouped by GroupName -->
        <StackPanel Orientation="Horizontal">
            <StackPanel>
                <RadioButton Content="Green" Tag="green"
GroupName="BorderBrush"
                             Checked="BorderRadioButton_Checked"/>
                <RadioButton Content="Yellow" Tag="yellow"
GroupName="BorderBrush"
                             Checked="BorderRadioButton_Checked"
IsChecked="True"/>
                <RadioButton Content="White" Tag="white"
GroupName="BorderBrush"
                             Checked="BorderRadioButton_Checked"/>
            </StackPanel>
        </StackPanel>
    </StackPanel>
    <Border x:Name="ExampleBorder"
            BorderBrush="#FFFFD700" Background="#FFFFFFFF"
            BorderThickness="10" Height="50" Margin="0,10"/>
</StackPanel>
```

C#

```
private void BGRadioButton_Checked(object sender, RoutedEventArgs e)
{
    RadioButton rb = sender as RadioButton;
```

```
    if (rb != null && ExampleBorder != null)
    {
        string colorName = rb.Tag.ToString();
        switch (colorName)
        {
            case "yellow":
                ExampleBorder.Background = new
SolidColorBrush(Colors.Yellow);
                break;
            case "green":
                ExampleBorder.Background = new
SolidColorBrush(Colors.Green);
                break;
            case "white":
                ExampleBorder.Background = new
SolidColorBrush(Colors.White);
                break;
        }
    }
}

private void BorderRadioButton_Checked(object sender, RoutedEventArgs e)
{
    RadioButton rb = sender as RadioButton;

    if (rb != null && ExampleBorder != null)
    {
        string colorName = rb.Tag.ToString();
        switch (colorName)
        {
            case "yellow":
                ExampleBorder.BorderBrush = new
SolidColorBrush(Colors.Gold);
                break;
            case "green":
                ExampleBorder.BorderBrush = new
SolidColorBrush(Colors.DarkGreen);
                break;
            case "white":
                ExampleBorder.BorderBrush = new
SolidColorBrush(Colors.White);
                break;
        }
    }
}
```

These two groups of `RadioButton` controls look like this:



Radio button states

A radio button has two states: selected or cleared. When a radio button is selected, its IsChecked property is `true`. When a radio button is cleared, its `IsChecked` property is `false`. A radio button can be cleared if a user selects another radio button in the same group, but it can't be cleared if the user selects it again. However, you can clear a radio button programmatically by setting its `IsChecked` property to `false`.

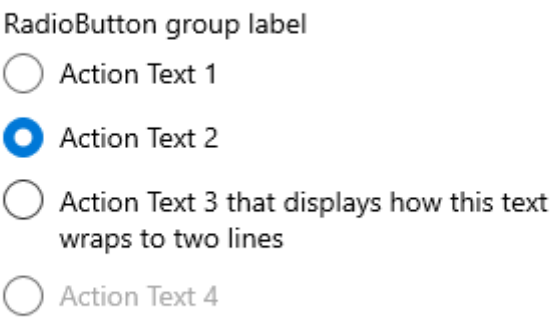
Visuals to consider

The default spacing of individual `RadioButton` controls is different than the spacing provided by a `RadioButtons` group. To apply the `RadioButtons` spacing to individual `RadioButton` controls, use a `Margin` value of `0,0,7,3`, as shown here.

XAML

```
<StackPanel>
    <StackPanel.Resources>
        <Style TargetType="RadioButton">
            <Setter Property="Margin" Value="0,0,7,3"/>
        </Style>
    </StackPanel.Resources>
    <TextBlock Text="Background"/>
    <RadioButton Content="Item 1"/>
    <RadioButton Content="Item 2"/>
    <RadioButton Content="Item 3"/>
</StackPanel>
```

The following images show the preferred spacing of radio buttons in a group.



> **Note**
>
> If you're using a WinUI RadioButtons control, the spacing, margins, and orientation are already optimized.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

The RadioButtons control for UWP apps is included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for these controls exist in both the Windows.UI.Xaml.Controls and Microsoft.UI.Xaml.Controls namespaces.

- ✓ **UWP APIs**: RadioButton class, IsChecked property, Checked event
- ✓ **WinUI 2 Apis**: RadioButtons class, SelectedItem property, SelectedIndex property, SelectionChanged event
- ✓ Open the WinUI 2 Gallery app and see the RadioButton in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

There are two ways to create radio button groups.

- Starting with WinUI 2.3, we recommend the **RadioButtons** control. This control simplifies layout, handles keyboard navigation and accessibility, and supports binding to a data source.
- You can use groups of individual **RadioButton** controls. If your app does not use WinUI 2.3 or later, this is the only option.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:RadioButtons />
```

Related topics

- Buttons
- Toggle switches
- Check boxes
- Lists and combo boxes

- Sliders
- RadioButtons class
- RadioButton class

Rating control

Article • 02/26/2025

The rating control allows users to view and set ratings that reflect degrees of satisfaction with content and services. Users can interact with the rating control with touch, pen, mouse, gamepad, or keyboard. The follow guidance shows how to use the rating control's features to provide flexibility and customization.

Overview

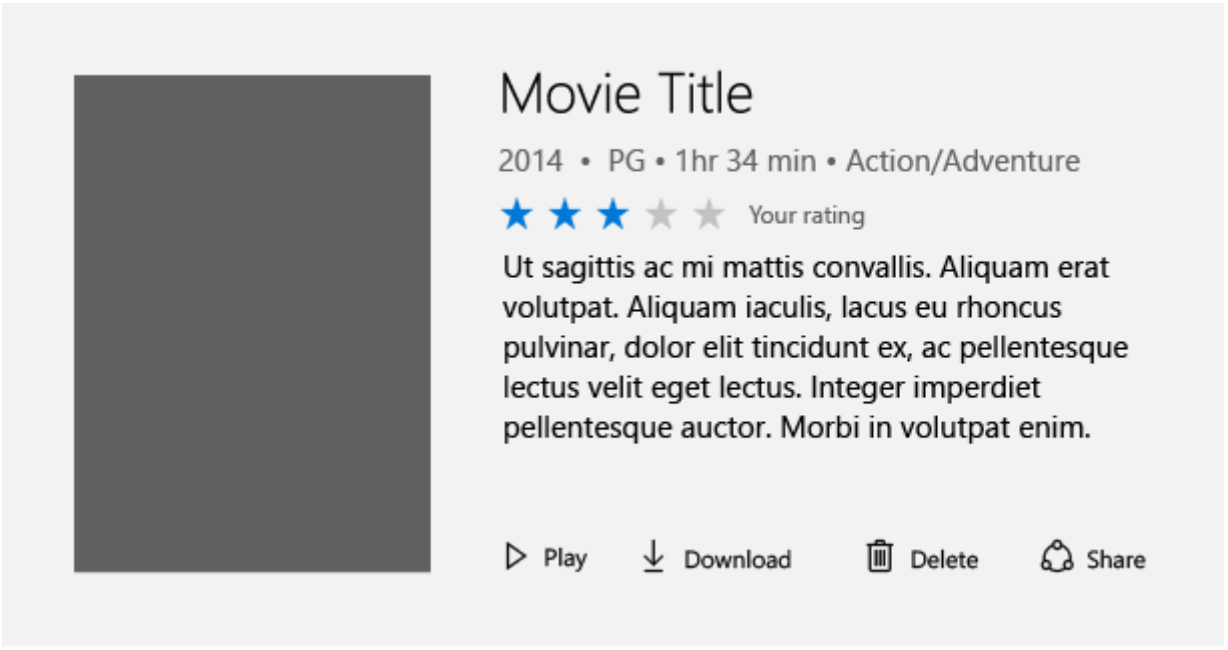
The rating control can be used to enter a rating, or made read-only to display a rating.

Editable rating with placeholder value

Perhaps the most common way to use the rating control is to display an average rating while still allowing the user to enter their own rating value. In this scenario, the rating control is initially set to reflect the average satisfaction rating of all users of a particular service or type of content (such as a music, videos, books, etc.). It remains in this state until a user interacts with the control with the goal of individually rating an item. This interaction changes the state of the ratings control to reflect the user's personal satisfaction rating.

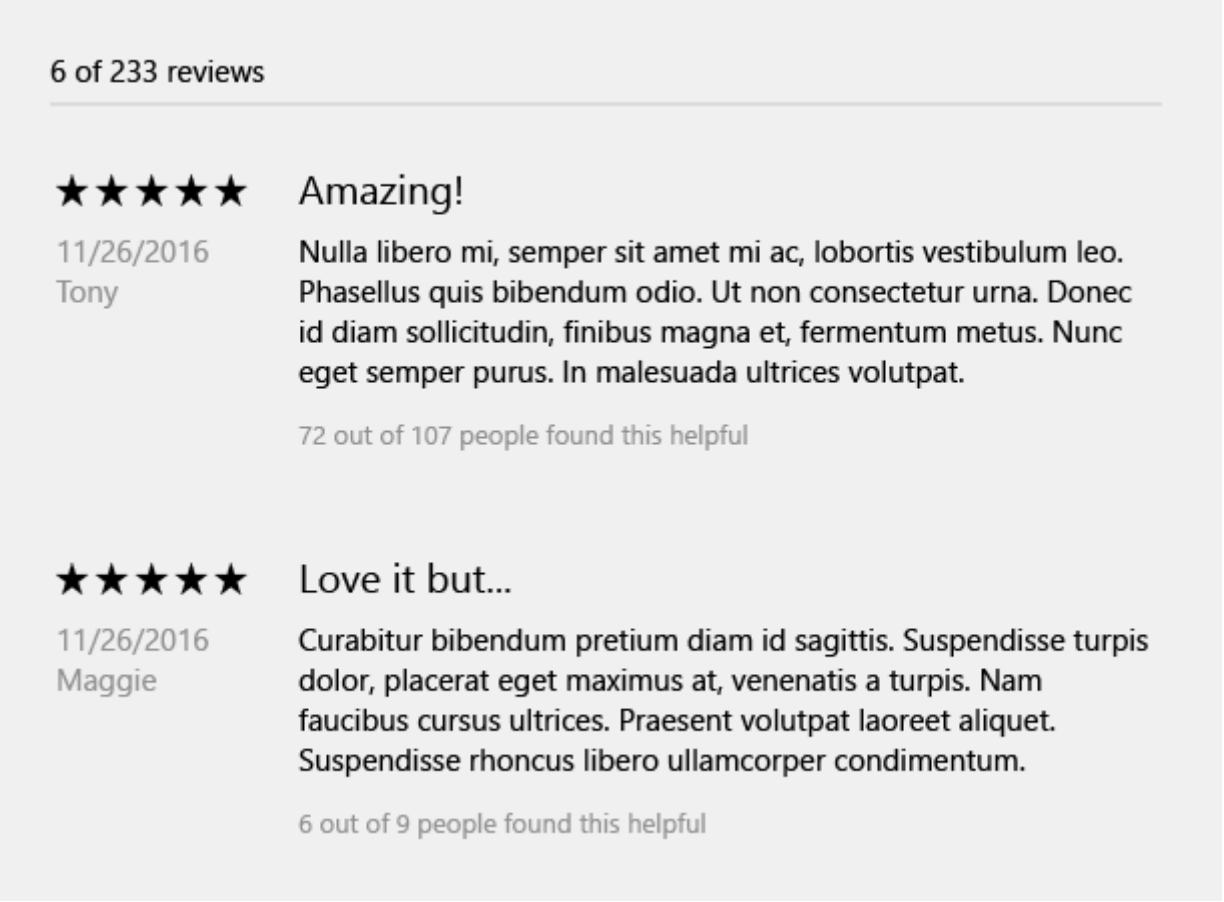
Initial average rating state

Representation of user rating once set



Read-only rating mode

Sometimes you need to show ratings of secondary content, such as that displayed in recommended content or when displaying a list of comments and their corresponding ratings. In this case, the user shouldn't be able to edit the rating, so you can make the control read-only. The read only mode is also the recommended way of using the rating control when it is used in very large virtualized lists of content, for both UI design and performance reasons.



Create a rating control

✓ **Important APIs**: RatingControl class

Open the WinUI 3 Gallery app and see RatingControl in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Editable rating control

This code shows how to create an editable rating control with a placeholder value.

XAML

```
<RatingControl x:Name="MyRating" ValueChanged="RatingChanged"/>
```

C#

```
private void RatingChanged(RatingControl sender, object args)
{
    if (sender.Value == null)
```

```
    {
        MyRating.Caption = "(" + SomeWebService.HowManyPreviousRatings() +
")";
    }
    else
    {
        MyRating.Caption = "Your rating";
    }
}
```

Read-only rating control

This code shows how to create a read-only rating control.

XAML

```
<RatingControl IsReadOnly="True"/>
```

Additional functionality

The rating control has many additional features which can be used. Details for using these features can be found in our reference documentation. Here is a non-comprehensive list of additional functionality:

- Great long list performance
- Compact sizing for tight UI scenarios
- Continuous value fill and rating
- Spacing customization
- Disable growth animations
- Customization of the number of stars

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

The RatingControl for UWP apps is included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in both the Windows.UI.Xaml.Controls and Microsoft.UI.Xaml.Controls namespaces.

- ✓ **UWP APIs:** RatingControl class
- ✓ **WinUI 2 Apis:** RatingControl class
- ✓ Open the WinUI 2 Gallery app and see RatingControl in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles, templates, and features for all controls.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:RatingControl />
```

Sliders

Article • 02/26/2025

A slider is a control that lets the user select from a range of values by moving a thumb control along a track.

Is this the right control?

Use a slider when you want your users to be able to set defined, contiguous values (such as volume or brightness) or a range of discrete values (such as screen resolution settings).

A slider is a good choice when you know that users think of the value as a relative quantity, not a numeric value. For example, users think about setting their audio volume to low or medium—not about setting the value to 2 or 5.

Don't use a slider for binary settings. Use a toggle switch instead.

Here are some additional factors to consider when deciding whether to use a slider:

- **Does the setting seem like a relative quantity?** If not, use radio buttons or a list box.
- **Is the setting an exact, known numeric value?** If so, use a numeric text box.
- **Would a user benefit from instant feedback on the effect of setting changes?** If so, use a slider. For example, users can choose a color more easily by immediately seeing the effect of changes to hue, saturation, or luminosity values.
- **Does the setting have a range of four or more values?** If not, use radio buttons.
- **Can the user change the value?** Sliders are for user interaction. If a user can't ever change the value, use read-only text instead.

If you are deciding between a slider and a numeric text box, use a numeric text box if:

- Screen space is tight.
- The user is likely to prefer using the keyboard.

Use a slider if:

- Users will benefit from instant feedback.

Recommendations

- Size the control so that users can easily set the value they want. For settings with discrete values, make sure the user can easily select any value using the mouse. Make sure the endpoints of the slider always fit within the bounds of a view.
- Give immediate feedback while or after a user makes a selection (when practical). For example, the Windows volume control beeps to indicate the selected audio volume.
- Use labels to show the range of values. Exception: If the slider is vertically oriented and the top label is Maximum, High, More, or equivalent, you can omit the other labels because the meaning is clear.
- Disable all associated labels or feedback visuals when you disable the slider.
- Consider the direction of text when setting the flow direction and/or orientation of your slider. Script flows from left to right in some languages, and from right to left in others.
- Don't use a slider as a progress indicator.
- Don't change the size of the slider thumb from the default size.
- Don't create a continuous slider if the range of values is large and users will most likely select one of several representative values from the range. Instead, use those values as the only steps allowed. For example if time value might be up to 1 month but users only need to pick from 1 minute, 1 hour, 1 day or 1 month, then create a slider with only 4 step points.

Additional usage guidance

Choosing the right layout: horizontal or vertical

You can orient your slider horizontally or vertically. Use these guidelines to determine which layout to use.

- Use a natural orientation. For example, if the slider represents a real-world value that is normally shown vertically (such as temperature), use a vertical orientation.
- If the control is used to seek within media, like in a video app, use a horizontal orientation.
- When using a slider in page that can be panned in one direction (horizontally or vertically), use a different orientation for the slider than the panning direction. Otherwise, users might swipe the slider and change its value accidentally when they try to pan the page.

- If you're still not sure which orientation to use, use the one that best fits your page layout.

Range direction

The range direction is the direction you move the slider when you slide it from its current value to its max value.

- For vertical slider, put the largest value at the top of the slider, regardless of reading direction. For example, for a volume slider, always put the maximum volume setting at the top of the slider. For other types of values (such as days of the week), follow the reading direction of the page.
- For horizontal styles, put the lower value on the left side of the slider for left-to-right page layout, and on the right for right-to-left page layout.
- The one exception to the previous guideline is for media seek bars: always put the lower value on the left side of the slider.

Steps and tick marks

- Use step points if you don't want the slider to allow arbitrary values between min and max. For example, if you use a slider to specify the number of movie tickets to buy, don't allow floating point values. Give it a step value of 1.
- If you specify steps (also known as snap points), make sure that the final step aligns to the slider's max value.
- Use tick marks when you want to show users the location of major or significant values. For example, a slider that controls a zoom might have tick marks for 50%, 100%, and 200%.
- Show tick marks when users need to know the approximate value of the setting.
- Show tick marks and a value label when users need to know the exact value of the setting they choose, without interacting with the control. Otherwise, they can use the value tooltip to see the exact value.
- Always show tick marks when step points aren't obvious. For example, if the slider is 200 pixels wide and has 200 snap points, you can hide the tick marks because users won't notice the snapping behavior. But if there are only 10 snap points, show tick marks.

Labels

- **Slider labels**

  The slider label indicates what the slider is used for.

    - Use a label with no ending punctuation (this is the convention for all control labels).
    - Position labels above the slider when the slider is in a form that places most of its labels above their controls.
    - Position labels to the sides when the slider is in a form that places most of its labels to the side of their controls.
    - Avoid placing labels below the slider because the user's finger might occlude the label when the user touches the slider.

- **Range labels**

  The range, or fill, labels describe the slider's minimum and maximum values.
    - Label the two ends of the slider range, unless a vertical orientation makes this unnecessary.
    - Use only one word, if possible, for each label.
    - Don't use ending punctuation.
    - Make sure these labels are descriptive and parallel. Examples: Maximum/Minimum, More/Less, Low/High, Soft/Loud.

- **Value labels**

  A value label displays the current value of the slider.
    - If you need a value label, display it below the slider.
    - Center the text relative to the control and include the units (such as pixels).
    - Since the slider's thumb is covered during scrubbing, consider showing the current value some other way, with a label or other visual. A slider setting text size could render some sample text of the right size beside the slider.

Appearance and interaction

A slider is composed of a track and a thumb. The track is a bar (which can optionally show various styles of tick marks) representing the range of values that can be input. The thumb is a selector, which the user can position by either tapping the track or by scrubbing back and forth on it.

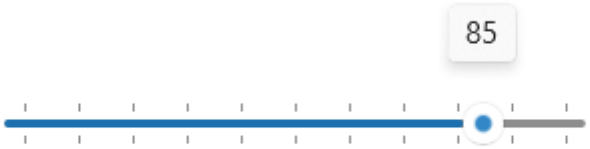
A slider has a large touch target. To maintain touch accessibility, a slider should be positioned far enough away from the edge of the display.

When you're designing a custom slider, consider ways to present all the necessary info to the user with as little clutter as possible. Use a value label if a user needs to know the units in order to understand the setting; find creative ways to represent these values graphically. A slider that controls volume, for example, could display a speaker graphic

without sound waves at the minimum end of the slider, and a speaker graphic with sound waves at the maximum end.

Examples

A slider with tick marks at 10 point intervals from 0 to 100.



Create a slider

✓ **Important APIs**: Slider class, Value property, ValueChanged event

Open the WinUI 3 Gallery app and see the Slider in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Here's how to create a slider in XAML.

XAML

```
<Slider x:Name="volumeSlider" Header="Volume" Width="200"
        ValueChanged="Slider_ValueChanged"/>
```

Here's how to create a slider in code.

C#

```
Slider volumeSlider = new Slider();
volumeSlider.Header = "Volume";
volumeSlider.Width = 200;
volumeSlider.ValueChanged += Slider_ValueChanged;

// Add the slider to a parent container in the visual tree.
stackPanel1.Children.Add(volumeSlider);
```

You get and set the value of the slider from the Value property. To respond to value changes, you can use data binding to bind to the Value property, or handle the ValueChanged event.

C#

```
private void Slider_ValueChanged(object sender,
RangeBaseValueChangedEventArgs e)
{
    Slider slider = sender as Slider;
    if (slider != null)
    {
        media.Volume = slider.Value;
    }
}
```

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** Slider class, Value property, ValueChanged event
- ✓ Open the WinUI 2 Gallery app and see the Slider in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related topics

- Toggle switches

- Slider class

Toggle switches

Article • 02/26/2025

The toggle switch represents a physical switch that allows users to turn things on or off, like a light switch. Use toggle switch controls to present users with two mutually exclusive options (such as on/off), where choosing an option provides immediate results.

To create a toggle switch control, you use the ToggleSwitch class.

Is this the right control?

Use a toggle switch for binary operations that take effect right after the user flips the toggle switch.

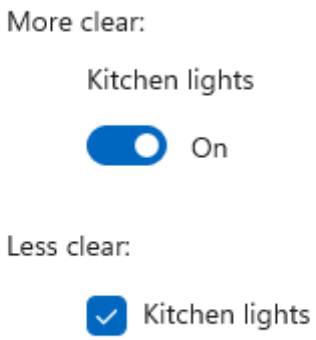
Think of the toggle switch as a physical power switch for a device: you flip it on or off when you want to enable or disable the action performed by the device.

To make the toggle switch easy to understand, label it with one or two words, preferably nouns, that describe the functionality it controls. For example, "WiFi" or "Kitchen lights."

Choosing between toggle switch and check box

For some actions, either a toggle switch or a check box might work. To decide which control would work better, follow these tips:

- Use a toggle switch for binary settings when changes become effective immediately after the user changes them.



In this example, it's clear with the toggle switch that the kitchen lights are set to "On." But with the checkbox, the user needs to think about whether the lights are on now or whether they need to check the box to turn the lights on.

- Use check boxes for optional ("nice to have") items.
- Use a checkbox when the user has to perform extra steps for changes to be effective. For example, if the user must click a "submit" or "next" button to apply changes, use a check box.
- Use check boxes when the user can select multiple items that are related to a single setting or feature.

Recommendations

- Use the default On and Off labels when you can; only replace them when it's necessary for the toggle switch to make sense. If you replace them, use a single word that more accurately describes the toggle. In general, if the words "On" and "Off" don't describe the action tied to a toggle switch, you might need a different control.
- Avoid replacing the On and Off labels unless you must; stick with the default labels unless the situation calls for custom ones.

Create a toggle switch

✓ **Important APIs**: ToggleSwitch class, IsOn property, Toggled event

Open the WinUI 3 Gallery app and see the ToggleSwitch in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source

> code on GitHub

Here's how to create a simple toggle switch. This XAML creates the toggle switch shown previously.

XAML

```
<ToggleSwitch x:Name="lightToggle" Header="Kitchen Lights"/>
```

Here's how to create the same toggle switch in code.

C#

```
ToggleSwitch lightToggle = new ToggleSwitch();
lightToggle.Header = "Kitchen Lights";

// Add the toggle switch to a parent container in the visual tree.
stackPanel1.Children.Add(lightToggle);
```

IsOn

The switch can be either on or off. Use the IsOn property to determine the state of the switch. When the switch is used to control the state of another binary property, you can use a binding as shown here.

XAML

```
<StackPanel Orientation="Horizontal">
    <ToggleSwitch x:Name="ToggleSwitch1" IsOn="True"/>
    <ProgressRing IsActive="{x:Bind ToggleSwitch1.IsOn, Mode=OneWay}"
                  Width="130"/>
</StackPanel>
```

Toggled

In other cases, you can handle the Toggled event to respond to changes in the state.

This example shows how to add a Toggled event handler in XAML and in code. The Toggled event is handled to turn a progress ring on or off, and change its visibility.

XAML

```
<ToggleSwitch x:Name="toggleSwitch1" IsOn="True"
```

```
                Toggled="ToggleSwitch_Toggled"/>
```

Here's how to create the same toggle switch in code.

C#

```
// Create a new toggle switch and add a Toggled event handler.
ToggleSwitch toggleSwitch1 = new ToggleSwitch();
toggleSwitch1.Toggled += ToggleSwitch_Toggled;

// Add the toggle switch to a parent container in the visual tree.
stackPanel1.Children.Add(toggleSwitch1);
```

Here's the handler for the Toggled event.

C#

```
private void ToggleSwitch_Toggled(object sender, RoutedEventArgs e)
{
    ToggleSwitch toggleSwitch = sender as ToggleSwitch;
    if (toggleSwitch != null)
    {
        if (toggleSwitch.IsOn == true)
        {
            progress1.IsActive = true;
            progress1.Visibility = Visibility.Visible;
        }
        else
        {
            progress1.IsActive = false;
            progress1.Visibility = Visibility.Collapsed;
        }
    }
}
```

On/Off labels

By default, the toggle switch includes literal On and Off labels, which are localized automatically. You can replace these labels by setting the OnContent, and OffContent properties.

This example replaces the On/Off labels with Show/Hide labels.

XAML

```
<ToggleSwitch x:Name="imageToggle" Header="Show images"
              OffContent="Show" OnContent="Hide"
```

```
                Toggled="ToggleSwitch_Toggled"/>
```

You can also use more complex content by setting the OnContentTemplate and OffContentTemplate properties.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the **Windows App SDK** and **WinUI 3**, but are generally applicable to UWP apps that use **WinUI 2**. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** ToggleSwitch class, IsOn property, Toggled event
- ✓ Open the WinUI 2 Gallery app and see the Slider in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls.

Related articles

- ToggleSwitch class
- Radio buttons
- Toggle switches
- Check boxes

Collections and lists

Article • 10/31/2023

Collections and lists both refer to the representation of multiple related data items that appear together. Collections can be represented in multiple ways, by different collection controls (also may be referred to as collection views). Collection controls display and enable interactions with collection-based content, such as a list of contacts, a list of dates, a collection of images, and so on.

> **Important APIs**: ItemsView, ListView class, GridView class, FlipView class, TreeView class, ItemsRepeater class

The controls covered in this article include:

Expand table

Control	Primary use
ItemsView	to display collections using a flexible layout system
ListView	to display text-heavy content collections
GridView	to display image-heavy content collections
FlipView	to display image-heavy content collections that require exactly one item to be in focus at a time
TreeView	to display text-heavy content collections in a specific hierarchy
ItemsRepeater	as a customizable building block to create custom collection controls

Design guidelines, features, and examples are given below for each control.

Each of these controls (with the exception of ItemsRepeater) provide built-in styling and interaction. However, to further customize the visual look of your collection view and the items inside it, a DataTemplate is used. Detailed information on data templates and customizing the look of a collection view can be found on the Item containers and templates page.

Each of these controls (with the exception of ItemsRepeater) also have built-in behavior to allow for the selection of single or multiple items. See Selection modes overview to learn more.

One of the scenarios not covered in this article is displaying collections in a table or across multiple columns. If you're looking to display a collection in this format, consider using the DataGrid control from the Windows Community Toolkit.

Examples

Expand table

WinUI 2 Gallery	
	If you have the **WinUI 2 Gallery** app installed, see the ListView, GridView, FlipView, TreeView, and ItemsRepeater in action. • Get the WinUI 2 Gallery app (Microsoft Store) • Get the source code (GitHub)

Items view

You can use an items view to display text- or image-heavy data items in a variety of flexible layouts that can be swapped at run-time.

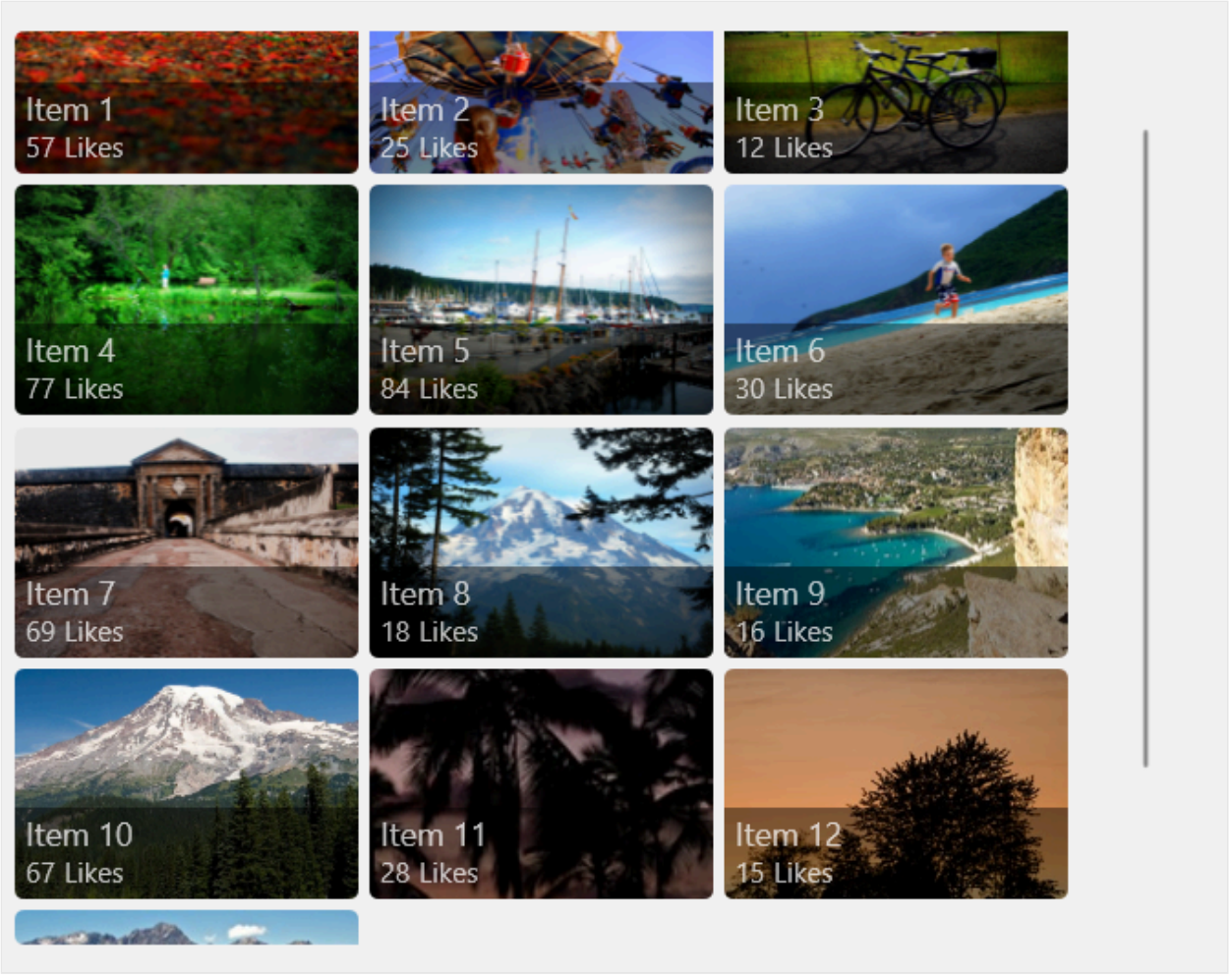
Is this the right control?

Use an items view to:

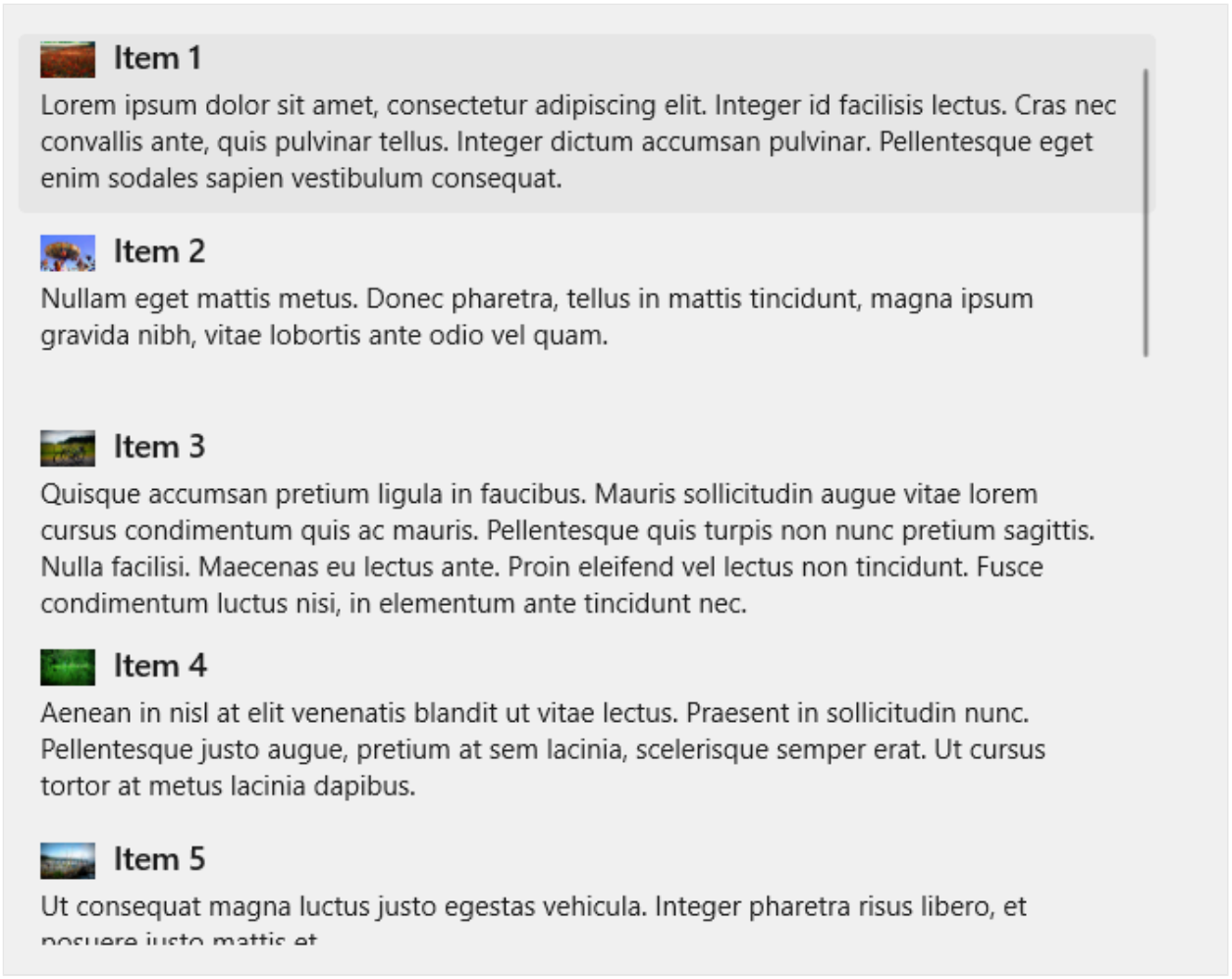
- Display a collection that primarily consists of text-based items, where all of the items should have the same visual and interaction behavior.
- Display a content collection in which the focal point of each item is an image, and each item should have the same visual and interaction behavior.
- Display content libraries.
- Accommodate a variety of use cases, including the following common ones:
  - Create a contacts list.
  - Storefront-type user interface (i.e. browsing apps, songs, products)
  - Interactive photo libraries

Examples

Here's an items view that shows a collection of photos in a uniform grid layout.



This is the same collection shown in a stack layout. The focus here is on the text rather than on the photo.



Related articles

Expand table

Topic	Description
Items view	Learn the essentials of using an items view in your app.
Item containers and templates	The items you display in a list or grid can play a major role in the overall look of your app. Make your app look great by customizing the look of your collection items through modifying control templates and data templates.

List view

List views represent text-heavy items, typically in a single-column, vertically-stacked layout. They let you categorize items and assign group headers, drag and drop items, curate content, and reorder items.

Is this the right control?

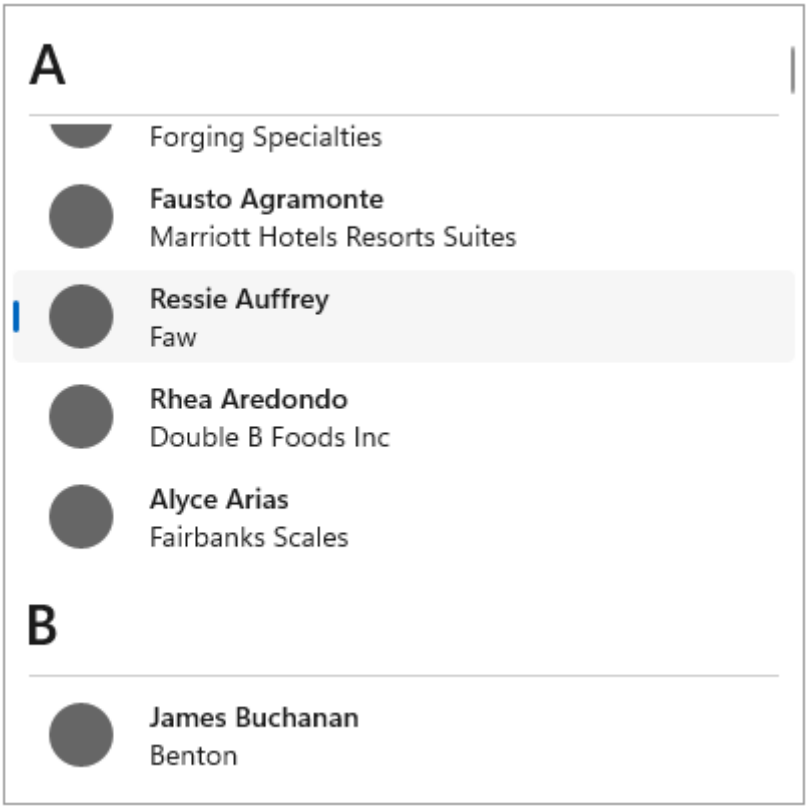
Use a list view to:

- Display a collection that primarily consists of text-based items, where all of the items should have the same visual and interaction behavior.
- Represent a single or categorized collection of content.
- Accommodate a variety of use cases, including the following common ones:
  - Create a list of messages or message log.
  - Create a contacts list.
  - Create the list pane in the list/details pattern. A list/details pattern is often used in email apps, in which one pane has a list of selectable items while the other pane (details) has a detailed view of the selected item.

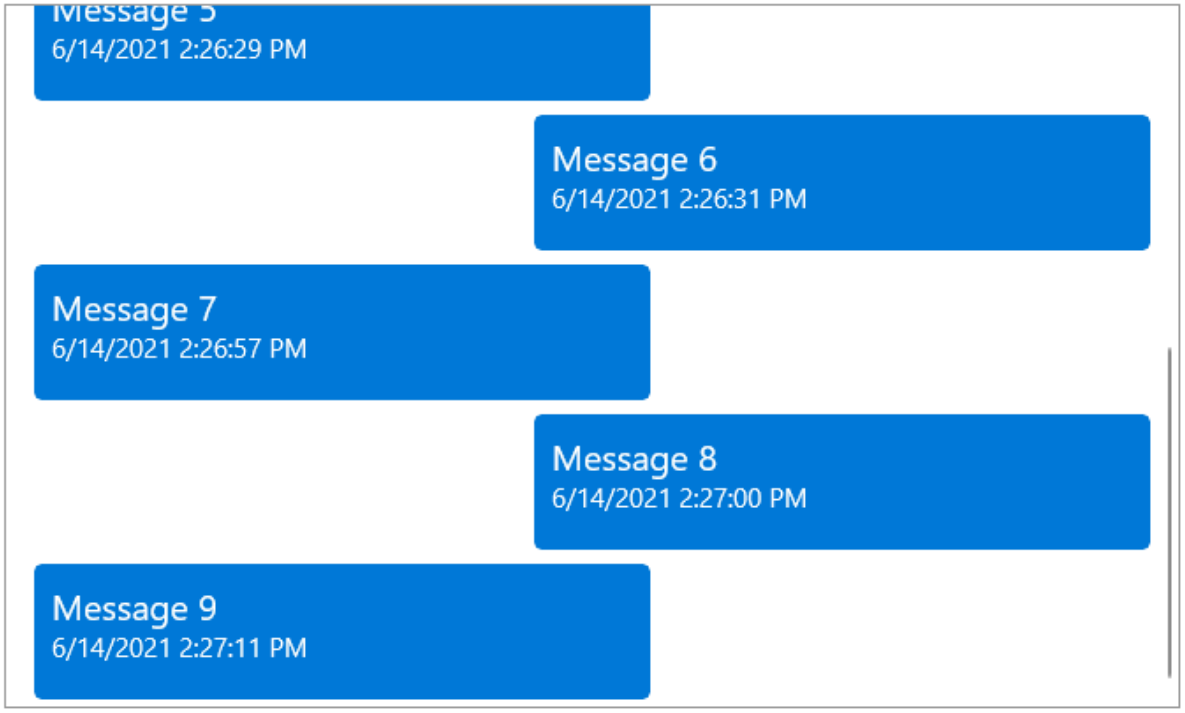
> **Note**
>
> If you need to handle pointer events for a **UIElement** in a scrollable view (such as a ScrollViewer or ListView), you must explicitly disable support for manipulation events on the element in the view by calling **UIElement.CancelDirectmanipulation**. To re-enable manipulation events in the view, call **UIElement.TryStartDirectManipulation**.

Examples

Here's a simple list view that shows a contacts list, and groups the data items alphabetically. The group headers (each letter of the alphabet in this example) can also be customized to stay "sticky", as in they will always appear at the top of the ListView while scrolling.



This is a ListView that has been inverted to display a log of messages, with the newest messages appearing at the bottom. With an inverted ListView, items appear at the bottom of the screen with a built-in animation.



Related articles

Expand table

Topic	Description
List view and grid view	Learn the essentials of using a list view or grid view in your app.
Item containers and templates	The items you display in a list or grid view can play a major role in the overall look of your app. Make your app look great by customizing the look of your collection items through modifying control templates and data templates.
Item templates for list view	Use these example item templates for a ListView to get the look of common app types.
Inverted lists	Inverted lists have new items added at the bottom, like in a chat app. Follow this article's guidance to use an inverted list in your app.
Pull-to-refresh	The pull-to-refresh mechanism lets a user pull down on a list of data using touch in order to retrieve more data. Use this article to implement pull-to-refresh in your list view.
Nested UI	Nested UI is a user interface (UI) that exposes actionable controls enclosed inside a container that a user can also take action on. For example, you might have list view item that contains a button, and the user can select the list item, or press the button nested within it. Follow these best practices to provide the best nested UI experience for your users.

Grid view

Grid views are suited for arranging and browsing image-based content collections. A grid view layout scrolls vertically and pans horizontally. Items are in a wrapped layout, as they appear in a left-to-right, then top-to-bottom reading order.

Is this the right control?

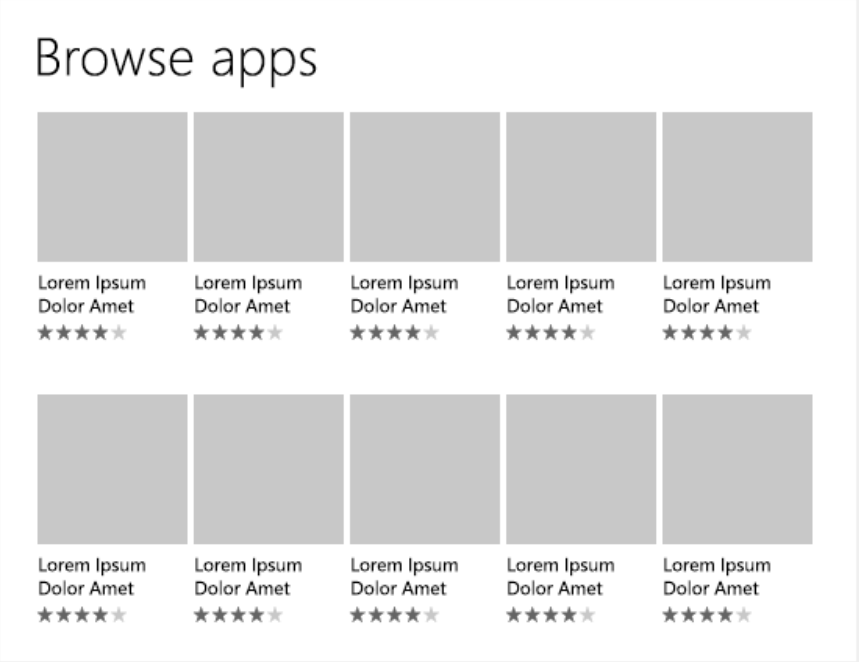
Use a grid view to:

- Display a content collection in which the focal point of each item is an image, and each item should have the same visual and interaction behavior.
- Display content libraries.
- Format the two content views associated with semantic zoom.
- Accommodate a variety of use cases, including the following common ones:
  - Storefront-type user interface (i.e. browsing apps, songs, products)
  - Interactive photo libraries

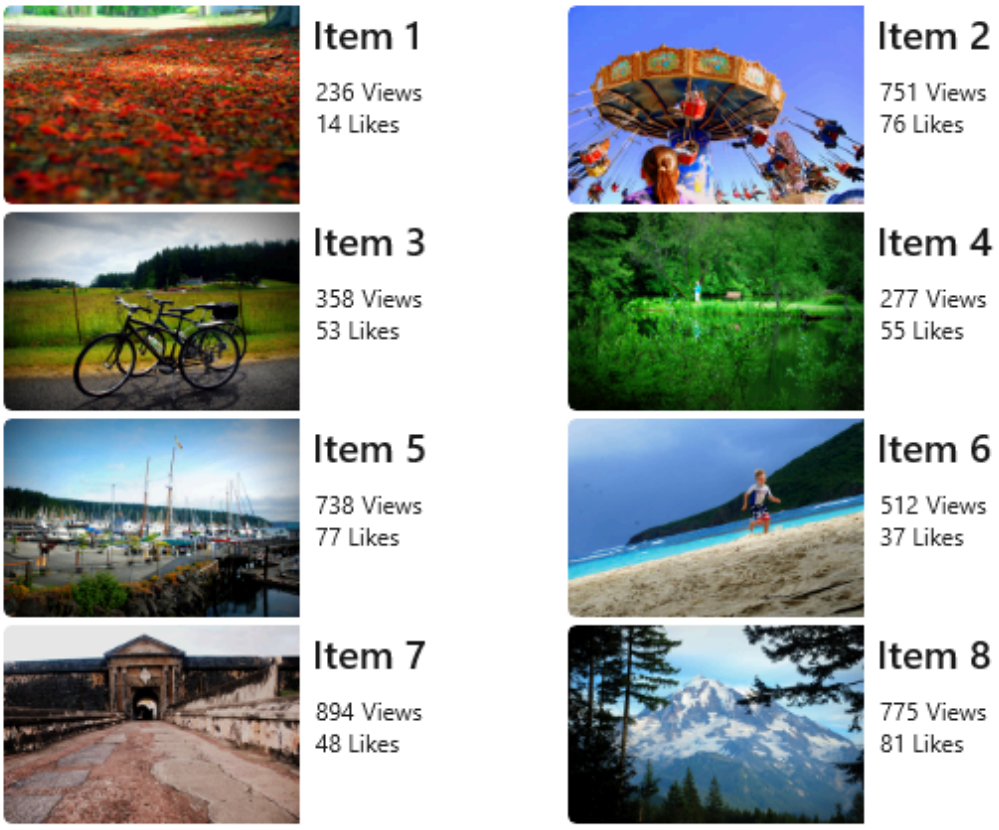
> **Note**
>
> If you need to handle pointer events for a **UIElement** in a scrollable view (such as a ScrollViewer or ListView), you must explicitly disable support for manipulation events on the element in the view by calling **UIElement.CancelDirectmanipulation()**. To re-enable manipulation events in the view, call **UIElement.TryStartDirectManipulation()**.

Examples

This example shows a typical grid view layout, in this case for browsing apps. Metadata for grid view items is usually restricted to a few lines of text and an item rating.



A grid view is an ideal solution for a content library, which is often used to present media such as pictures and videos. In a content library, users expect to be able to tap an item to invoke an action.



Related articles

Expand table

Topic	Description
List view and grid view	Learn the essentials of using a list view or grid view in your app.
Item containers and templates	The items you display in a list or grid view can play a major role in the overall look of your app. Make your app look great by customizing the look of your collection items through modifying control templates and data templates.
Item templates for grid view	Use these example item templates for a ListView to get the look of common app types.
Nested UI	Nested UI is a user interface (UI) that exposes actionable controls enclosed inside a container that a user can also take action on. For example, you might have list view item that contains a button, and the user can select the list item, or press the button nested within it. Follow these best practices to provide the best nested UI experience for your users.

Flip view

Flip views are suited for browsing image-based content collections, specifically where the desired experience is for only one image to be visible at a time. A flip view allows the user to move or "flip" through the collection items (either vertically or horizontally), having each item appear one at a time after the user interaction.

Is this the right control?

Use a flip view to:

- Display a small to medium (less than 25 items) collection, where the collection is made up of images with little to no metadata.
- Display items one at a time, and allow the end-user to flip through the items at their own pace.
- Accommodate a variety of use cases, including the following common ones:
  - Photo galleries
  - Product galleries or showcases

Examples

The following two examples show a FlipView that flips horizontally and vertically, respectively.

Related articles

Expand table

Topic	Description
Flip view	Learn the essentials of using a flip view in your app, along with how to customize the look of your items within a flip view.

Tree view

Tree views are suited for displaying text-based collections that have an important hierarchy that needs to be showcased. Tree view items are collapsible/expandable, are shown in a visual hierarchy, can be supplemented with icons, and can be dragged and dropped between tree views. Tree views allow for N-level nesting.

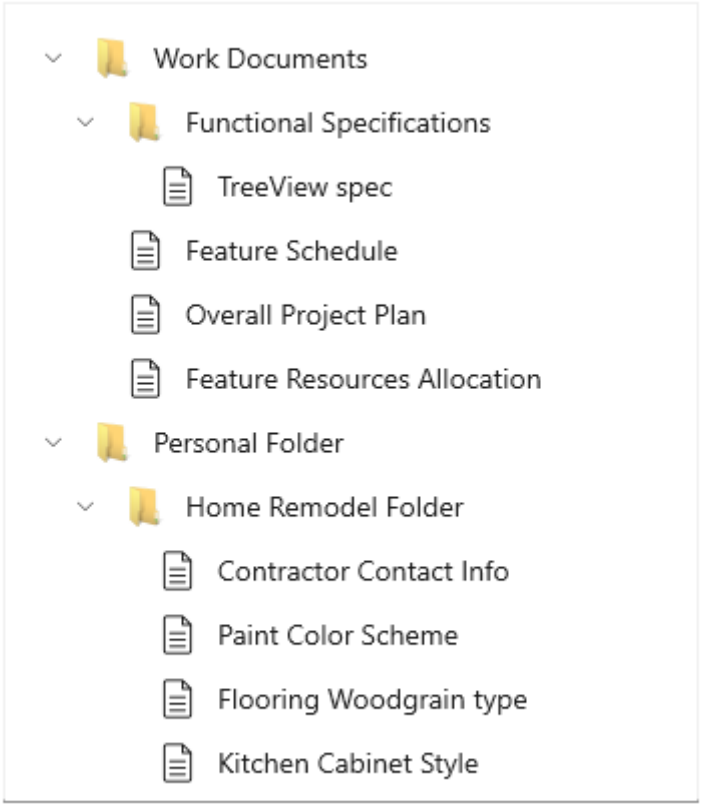
Is this the right control?

Use a tree view to:

- Display a collection of nested items whose context and meaning is dependent on a hierarchy or specific organizational chain.
- Accommodate a variety of use cases, including the following common ones:
  - File browser
  - Company organizational chart

Examples

Here is an example of a tree view that represents a file explorer, and displays many different nested items supplemented by icons.



Related articles

Expand table

Topic	Description
Tree view	Learn the essentials of using a tree view in your app, along with how to customize the look and interaction behavior of your items within a tree view.

ItemsRepeater

ItemsRepeater is different from the rest of the collection controls shown on this page because it does not provide any styling or interaction out-of-box, i.e. when simply placed on a page without defining any properties. ItemsRepeater is rather a building block that you as a developer can use to create your own custom collections control, specifically one that cannot be achieved by using the other controls in this article.

ItemsRepeater is a data-driven and high-performance panel that can be tailored to fit your exact needs.

> **Tip**
>
> The ItemsView control is built on top of ItemsRepeater and provides many of the benefits of ItemsRepeater without the need to create your own custom collection control.

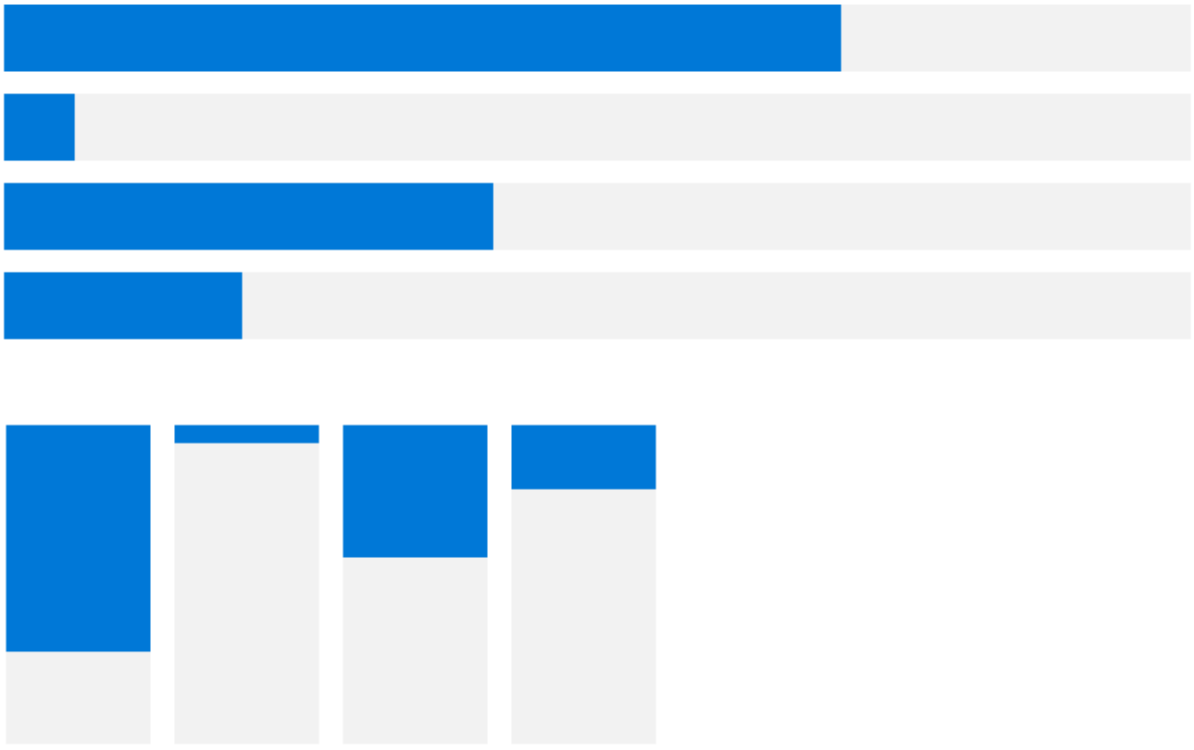
Is this the right control?

Use an ItemsRepeater if:

- You have a specific user interface and user experience in mind that cannot be created using existing collection controls.
- You have an existing data source for your items (such as data pulled from the internet, a database, or a pre-existing collection in your code-behind).

Examples

The following three examples are all ItemsRepeater controls that are bound to the same data source (a collection of numbers). The collection of numbers is represented in three ways, with each of the ItemsRepeaters below using a different custom Layout and a different custom ItemTemplate.



Related articles

Expand table

Topic	Description
ItemsRepeater	Learn the essentials of using an ItemsRepeater in your app, along with how to implement all of the necessary interaction and visual components for your collection view.

Globalization and localization checklist

- **Wrapping**: Allow two lines for the list label.
- **Horizontal expansion**: Make sure fields can accommodate text expansion and are scrollable.
- **Vertical spacing**: Use non-Latin characters for vertical spacing to ensure non-Latin scripts will display properly.

Get the sample code

- WinUI Gallery sample - See all the XAML controls in an interactive format.

Related articles

Design and UX Guidelines

- List/details
- Nav pane
- Semantic zoom
- Drag and drop
- Thumbnail images

API reference

- ListView class

- GridView class
- ComboBox class
- ListBox class

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Items view

Article • 10/31/2023

Use an items view to display a collection of data items, such as photos in an album or items in a product catalog.

The items view is similar to the list view and grid view controls, and you can use it in most cases where you would use those controls. One advantage of the items view is its ability to switch the layout on the fly while preserving item selection.

The items view control is built using the ItemsRepeater, ScrollView, ItemContainer, and ItemCollectionTransitionProvider components, so it offers the unique ability to plug in custom Layout or ItemCollectionTransitionProvider implementations. The item view's inner ScrollView control allows scrolling and zooming of the items. It also offers features unavailable in the ScrollViewer control used by the list view and grid view, like the ability to control the animation during programmatic scrolls.

Like the list view and grid view controls, the items view can use UI and data virtualization; handle keyboard, mouse, pen, and touch input; and has built-in accessibility support.

Is this the right control?

Use an items view to:

- Display a collection where all of the items should have the same visual and interaction behavior.
- Display a content collection with the ability to switch between list, grid, and custom layouts.
- Accommodate a variety of use cases, including the following common ones:
  - Storefront-type user interface (i.e. browsing apps, songs, products)
  - Interactive photo libraries
  - Contacts list

Create an items view

✓ Important APIs: ItemsView class, ItemsSource property, ItemTemplate property, LinedFlowLayout, StackLayout, UniformGridLayout

Open the WinUI 3 Gallery app and see the ItemsView in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

An ItemsView can show a collection of items of any type. To populate the view, set the ItemsSource property to a data source.

> **Note**
>
> Unlike other collection controls (those that derive from ItemsControl), ItemsView does not have an Items property that you can add data items to directly.

Set the items source

You typically use an items view to display data from a source such as a database or the internet. To populate an items view from a data source, you set its ItemsSource property to a collection of data items.

Set ItemsSource in code

Here, the ItemsSource is set in code directly to an instance of a collection.

C#

```
// Data source.
List<String> itemsList = new List<string>();
itemsList.Add("Item 1");
itemsList.Add("Item 2");

// Create a new ItemsView and set the data source.
ItemsView itemsView1 = new ItemsView();
itemsView1.ItemsSource = itemsList;

// Add the ItemsView to a parent container in the visual tree.
rootGrid.Children.Add(itemsView1);
```

Bind ItemsSource in XAML

You can also bind the ItemsSource property to a collection in XAML. For more info, see Data binding with XAML.

> **ⓘ Important**
>
> When you use the x:Bind markup extension in a DataTemplate, you have to specify the data type (`x:DataType`) on the data template.

Here, the ItemsSource is bound to a collection of custom data objects (of type `Photo`).

XAML

```
<ItemsView ItemsSource="{x:Bind Photos}">

</ItemsView>
```

C#

```
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();
        Photos = new ObservableCollection<Photo>();
        PopulatePhotos();
    }

    public ObservableCollection<Photo> Photos
    {
        get; private set;
    }


    private void PopulatePhotos()
    {
        // Populate the Photos collection.
    }
}

public class Photo
{
    public BitmapImage PhotoBitmapImage { get; set; }
    public string Title { get; set; }
    public int Likes { get; set; }
}
```

Specify the look of the items

By default, a data item is displayed in the items view as the string representation of the data object it's bound to. You typically want to show a more rich presentation of your

data. To specify exactly how items in the items view are displayed, you create a DataTemplate. The XAML in the DataTemplate defines the layout and appearance of controls used to display an individual item. The controls in the layout can be bound to properties of a data object, or have static content defined inline. The DataTemplate is assigned to the ItemTemplate property of the ItemsView control.

> **Important**
>
> The root element of the DataTemplate must be an **ItemContainer**; otherwise, an exception is thrown. ItemContainer is an independent primitive control that is used by ItemsView to show the selection states and other visualizations of an individual item in the item collection.

In this example, the DataTemplate is defined in the `Page.Resources` ResourceDictionary. It includes an Image control to show the picture and an overlay that contains the image title and number of likes it's received.

XAML

```
 <Page.Resources>
     <DataTemplate x:Key="PhotoItemTemplate" x:DataType="local:Photo">
         <ItemContainer AutomationProperties.Name="{x:Bind Title}">
             <Grid>
                 <Image Source="{x:Bind PhotoBitmapImage, Mode=OneWay}"
                        Stretch="UniformToFill" MinWidth="70"
                        HorizontalAlignment="Center"
VerticalAlignment="Center"/>
                 <StackPanel Orientation="Vertical" Height="40"
Opacity=".75"
                             VerticalAlignment="Bottom" Padding="5,1,5,1"
                             Background="{ThemeResource
SystemControlBackgroundBaseMediumBrush}">
                     <TextBlock Text="{x:Bind Title}"
                                Foreground="{ThemeResource
SystemControlForegroundAltHighBrush}"/>
                     <StackPanel Orientation="Horizontal">
                         <TextBlock Text="{x:Bind Likes}"
                                    Style="{ThemeResource
CaptionTextBlockStyle}"
                                    Foreground="{ThemeResource
SystemControlForegroundAltHighBrush}"/>
                         <TextBlock Text=" Likes"
                                    Style="{ThemeResource
CaptionTextBlockStyle}"
                                    Foreground="{ThemeResource
SystemControlForegroundAltHighBrush}"/>
                     </StackPanel>
                 </StackPanel>
```

```
            </Grid>
        </ItemContainer>
    </DataTemplate>
</Page.Resources>

<Grid x:Name="rootGrid">
    <ItemsView ItemsSource="{x:Bind Photos}" Width="480" Height="400"
               ItemTemplate="{StaticResource PhotoItemTemplate}">
    </ItemsView>
</Grid>
```

Here's what the item layout defined by the data template looks like.

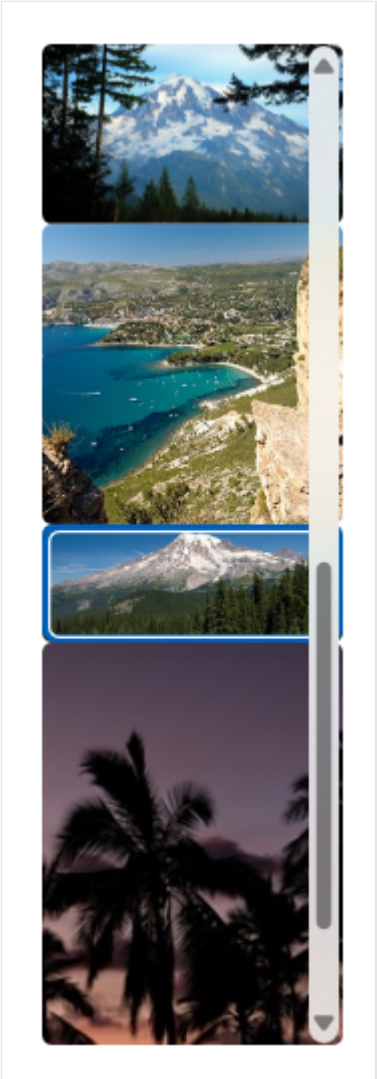
Change the layout of items

When you add items to an ItemsView control, it automatically wraps each item in an ItemContainer and then lays out all of the item containers. How these item containers are laid out depends on the Layout property of the control.

You can modify the layout of items by adjusting the properties on the current layout, or you can completely replace the current layout with a different layout. You can use one of the layouts described next, or you can derive a custom layout from the Layout class.

StackLayout

By default, ItemsView uses a StackLayout, which produces a vertical list, shown here with default property settings and a simple image template.



This XAML sets the spacing between items in the StackLayout to 5px.

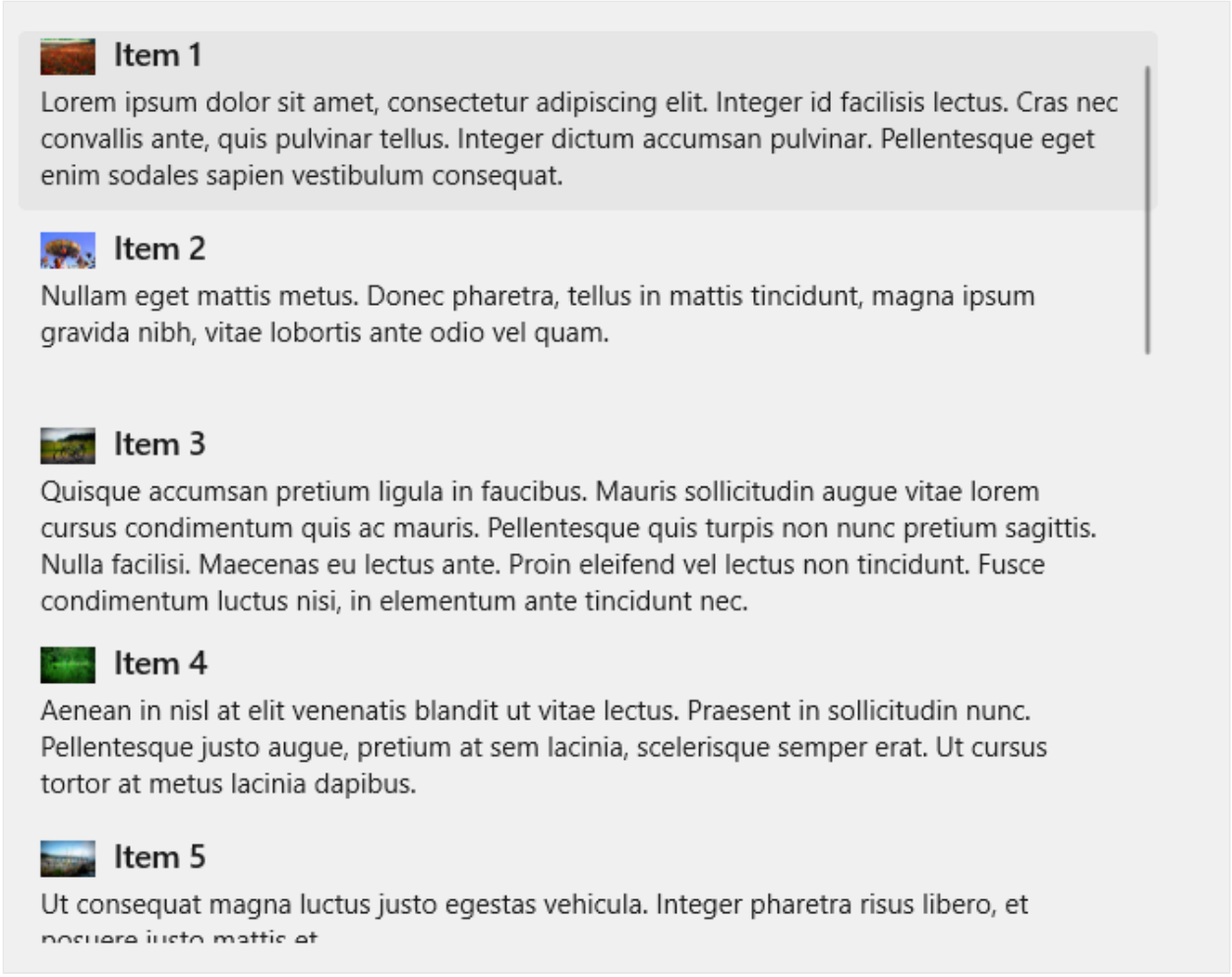
XAML

```
<ItemsView ItemsSource="{x:Bind Photos}" Width="480" Height="400"
           ItemTemplate="{StaticResource PhotoItemTemplate}">
    <ItemsView.Layout>
        <StackLayout Spacing="5"/>
    </ItemsView.Layout>
</ItemsView>
```

StackLayout provides properties to control:

- whether the layout is vertical or horizontal (Orientation)
- the spacing of items (Spacing)

You can use the layout in combination with the ItemTemplate to give your collection a variety of looks to suit your needs. For example, the WinUI Gallery sample changes the ItemTemplate used with the StackLayout to look like this.



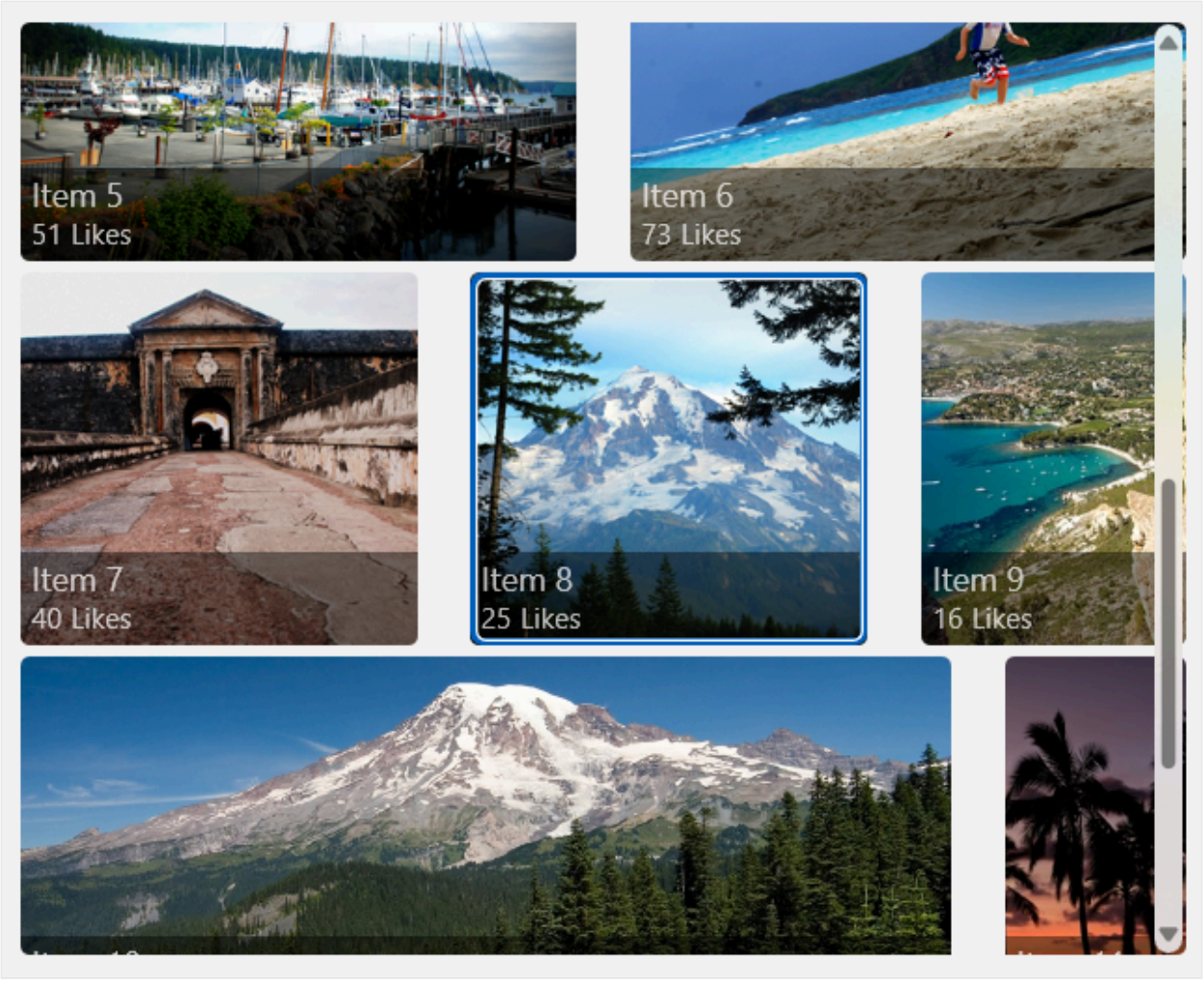
LinedFlowLayout

The LinedFlowLayout positions elements sequentially from left to right, then top to bottom, in a wrapping layout. Use this layout to display an item collection where the items have a fixed height but a variable width. We recommend it for image based collections. This layout also has built-in animations that play when the collection has items added or removed, and when the view is resized.

Here's an items view that shows a collection of photos in a lined flow layout.

XAML

```
<ItemsView Width="500" Height="400" HorizontalAlignment="Left"
           ItemTemplate="{StaticResource LinedFlowLayoutItemTemplate}">
    <ItemsView.Layout>
        <LinedFlowLayout ItemsStretch="Fill"
                         LineHeight="160"
                         LineSpacing="5"
                         MinItemSpacing="5"/>
    </ItemsView.Layout>
</ItemsView>
```



LinedFlowLayout provides properties to control:

- the size and spacing of items (LineHeight, LineSpacing, MinItemSpacing)
- the arrangement of items (ItemsJustification, ItemsStretch)

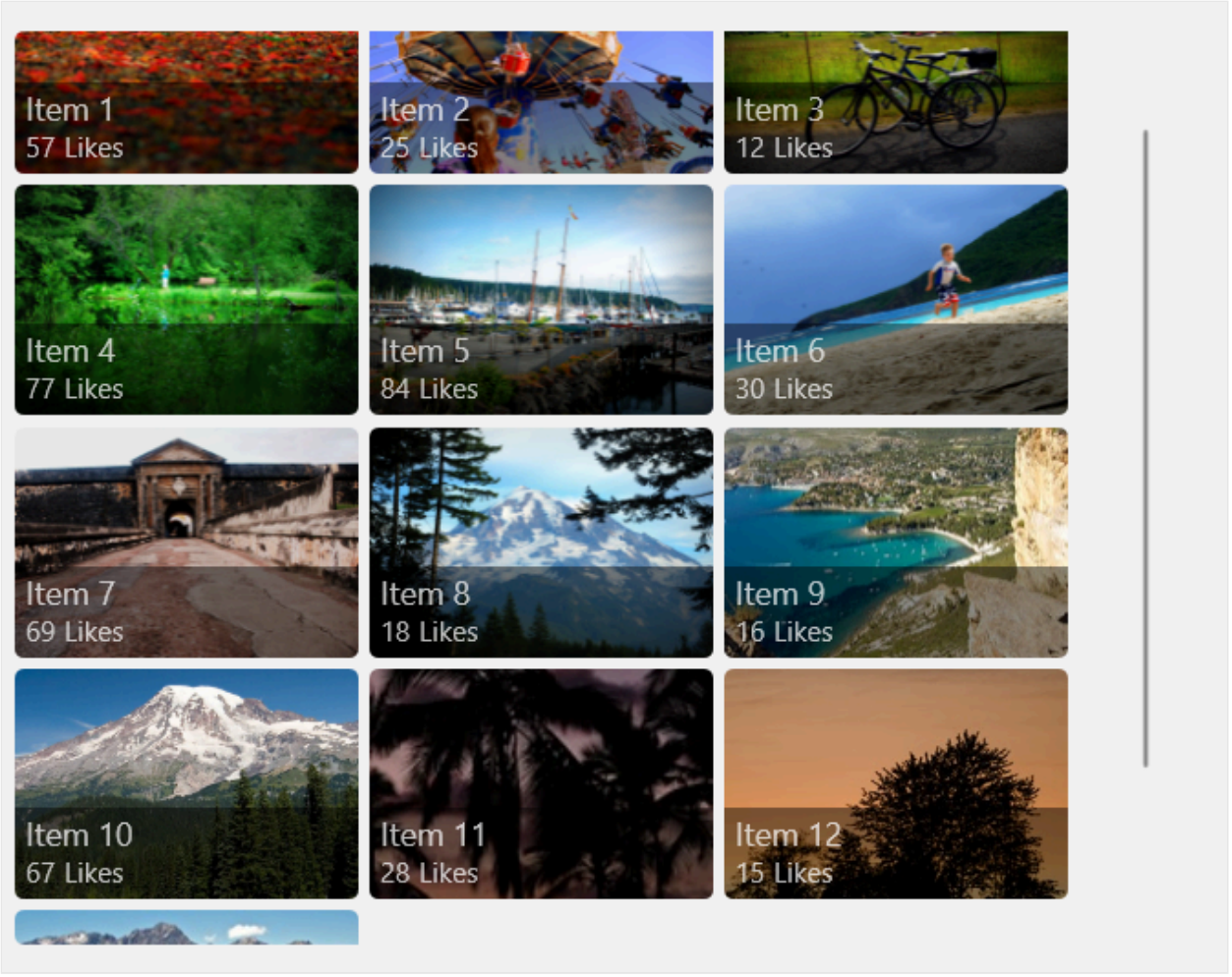
UniformGridLayout

The UniformGridLayout positions elements sequentially from left to right or top to bottom (depending on the Orientation) in a wrapping layout. Each item is the same size.

Here's an items view that shows a collection of photos in a uniform grid layout.

XAML

```
<ItemsView ItemsSource="{x:Bind Photos}" Width="480" Height="400"
           ItemTemplate="{StaticResource PhotoItemTemplate}">
    <ItemsView.Layout>
        <UniformGridLayout MaximumRowsOrColumns="3"
                           MinRowSpacing="5"
                           MinColumnSpacing="5"/>
    </ItemsView.Layout>
</ItemsView>
```



UniformGridLayout provides properties to control:

- whether the layout is row or column based (Orientation)
- the number of rows or columns (MaximumRowsOrColumns)
- the size and spacing of items (MinItemHeight, MinItemWidth, MinColumnSpacing, MinRowSpacing)
- the arrangement of items (ItemsJustification, ItemsStretch)

> **Tip**
>
> Use the interactive demo in the **WinUI 3 Gallery app** to see the effect of these properties in real time.

Item selection and interaction

You can choose from a variety of ways to let users interact with an items view. By default, users can select a single item. You can change the SelectionMode property to enable multi-selection or to disable selection. You can set the IsItemInvokedEnabled property so that users click an item to invoke an action instead of selecting the item.

This table shows the ways a user can interact with an items view, and how you can respond to the interaction.

Expand table

To enable this interaction:	Use these settings:	Handle this event:	Use this property to get the selected item:
No interaction	SelectionMode="None" IsItemInvokedEnabled="False"	N/A	N/A
Single selection	SelectionMode="Single" IsItemInvokedEnabled="False"	SelectionChanged	SelectedItem
Multiple selection	SelectionMode="Multiple" IsItemInvokedEnabled="False"	SelectionChanged	SelectedItems
Extended selection	SelectionMode="Extended" IsItemInvokedEnabled="False"	SelectionChanged	SelectedItems
Click	SelectionMode="None" IsItemInvokedEnabled="True"	ItemInvoked	N/A

> **Note**
>
> You can enable IsItemInvokedEnabled to raise an ItemInvoked event while SelectionMode is also set to `Single`, `Multiple`, or `Extended`. If you do this, the ItemInvoked event is raised first, and then the SelectionChanged event is raised. In some cases (for example, if you go to another page in the ItemInvoked event handler), the SelectionChanged event isn't raised and the item isn't selected.

You can set these properties in XAML or in code, as shown here:

XAML

```
<ItemsView x:Name="myItemsView" SelectionMode="None"
IsItemInvokedEnabled="True"/>
```

C#

```
myItemsView.SelectionMode = ItemsViewSelectionMode.None;
myItemsView.IsItemInvokedEnabled = true;
```

Read-only

You can set the SelectionMode property to ItemsViewSelectionMode.None to disable item selection. This puts the control in read-only mode, so that it's used for displaying data, but not for interacting with it. That is, the item selection is disabled, but the control itself is not.

> **Note**
>
> Items can still be selected and deselected programmatically, but not through user interaction.

Single selection

This table describes keyboard, mouse, and touch interactions when SelectionMode is set to `Single`.

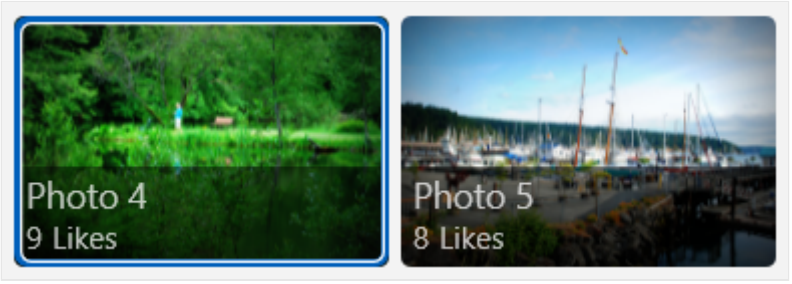
Expand table

Modifier key	Interaction
None	• Users can select a single item by using the space bar, mouse clicks, or taps.
Ctrl	• Users can deselect a single item by using the space bar, mouse clicks, or taps. • By using the arrow keys, users can move the focus independent of the selection.

When SelectionMode is set to `Single`, you can get the selected data item from the SelectedItem property. If no item is selected, SelectedItem is `null`.

If you try to set an item that's not in the items collection as SelectedItem, the operation is ignored and SelectedItem is `null`.

The default selection indicator for `Single` selection looks like this.



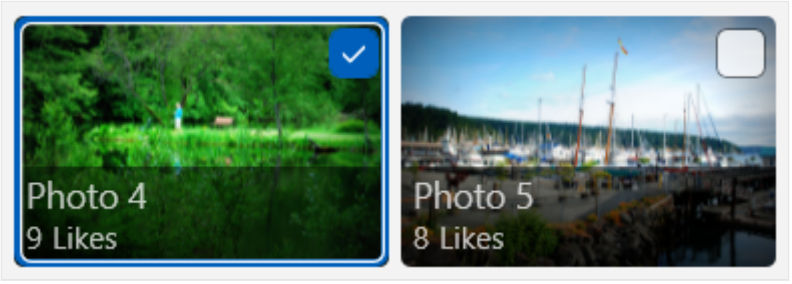
Multiple selection

This table describes the keyboard, mouse, and touch interactions when SelectionMode is set to `Multiple`.

Expand table

Modifier key	Interaction
None	• Users can select multiple items by using the space bar, mouse clicks, or taps to select the focused item. • By using the arrow keys, users can move the focus independent of their selection.
Shift	• Users can select multiple contiguous items by clicking or tapping the first item in the selection and then clicking or tapping the last item in the selection. • By using the arrow keys, users can select contiguous items starting with the item that's selected when they select the Shift key.

The default selection indicator for `Multiple` selection looks like this.



Extended selection

This table describes the keyboard, mouse, and touch interactions when SelectionMode is set to `Extended`.

Expand table

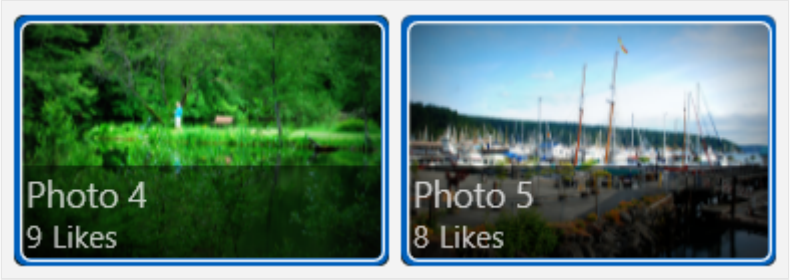
Modifier key	Interaction
None	• The behavior is the same as `Single` selection.
Ctrl	• Users can select multiple items by using the space bar, mouse clicks, or taps to select the focused item. • By using the arrow keys, users can move the focus independent of the selection.
Shift	• Users can select multiple contiguous items by clicking or tapping the first item in the selection and then clicking or tapping the last item in the selection.

Modifier key	Interaction
	• By using the arrow keys, users can select contiguous items starting with the item that's selected when they select the Shift key.

When SelectionMode is set to `Multiple` or `Extended`, you can get the selected data items from the SelectedItems property.

The SelectedItem and SelectedItems properties are synchronized. For example, if you set SelectedItem to `null`, SelectedItems is empty. In multi-select mode, SelectedItem contains the item that was selected first.

The default selection indicator for `Extended` selection is the same as for `Single` selection and looks like this.



Manage item selection programmatically

> **Note**
>
> These selection methods ignore the SelectionMode property and have an effect even when SelectionMode is `Single` or `None`.

Sometimes, you might need to manipulate the ItemsView item selection programmatically. For example, you might display a Select all button to let users select all items in a list. In this case, it's usually not very efficient to add and remove items from the SelectedItems collection one by one. It's more efficient to use the Select, SelectAll, Deselect, and InvertSelection methods to modify the selection than to use the SelectedItems property.

> **Tip**

You can select all items in a collection by calling the SelectAll method. There is no corresponding method to deselect all items. However, you can deselect all items by calling SelectAll followed immediately by InvertSelection.

Get the sample code

- WinUI Gallery sample - See all the XAML controls in an interactive format.

Related articles

- Guidelines for lists
- ItemsView class
- StackLayout
- UniformGridLayout
- LinedFlowLayout

List view and grid view

Article • 02/26/2025

Most applications manipulate and display sets of data, such as a gallery of images or a set of email messages. The XAML UI framework provides ListView and GridView controls that make it easy to display and manipulate data in your app.

> **Note**
>
> ListView and GridView both derive from the **ListViewBase** class, so they have the same functionality but display data differently. In this article, discussions about *list view* apply to both the ListView and GridView controls, unless otherwise specified. We may refer to classes like ListView or ListViewItem, but the *List* prefix can be replaced with *Grid* for the corresponding grid equivalent (GridView or GridViewItem).

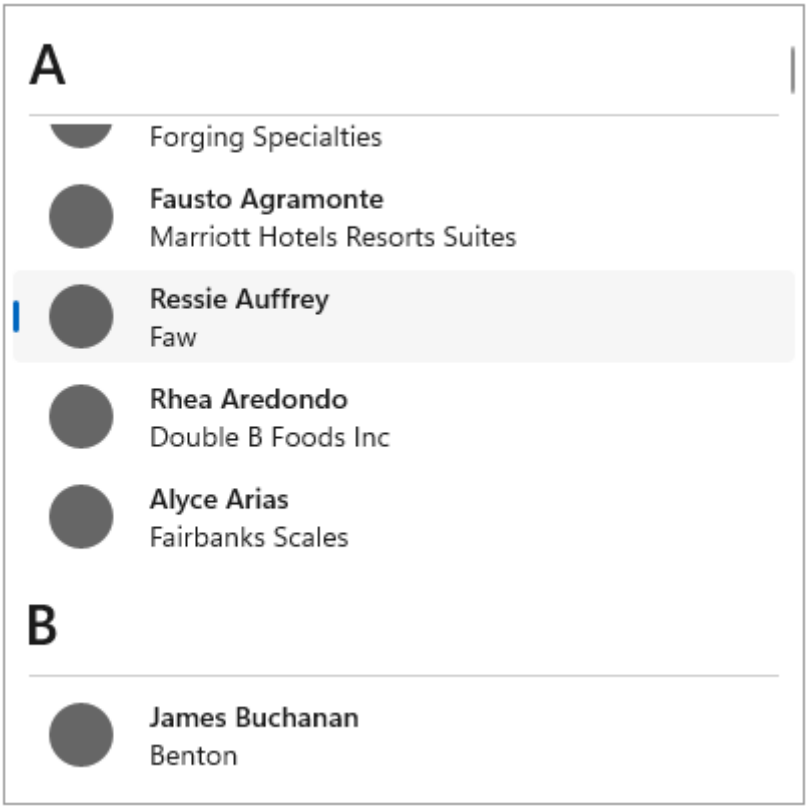
ListView and GridView controls provide many benefits as you work with collections. Both are easy to implement and provide basic UI, interaction, and scrolling while being easily customizable. And both can be bound to existing dynamic data sources or to hard-coded data that's provided in the XAML itself or the code-behind.

Both controls are flexible to use in a variety of scenarios but, overall, they work best with collections in which all items have the same basic structure and appearance, as well as the same interaction behavior. That is, they all should perform the same action when they're clicked (for example, to open a link or browse).

Compare ListView and GridView

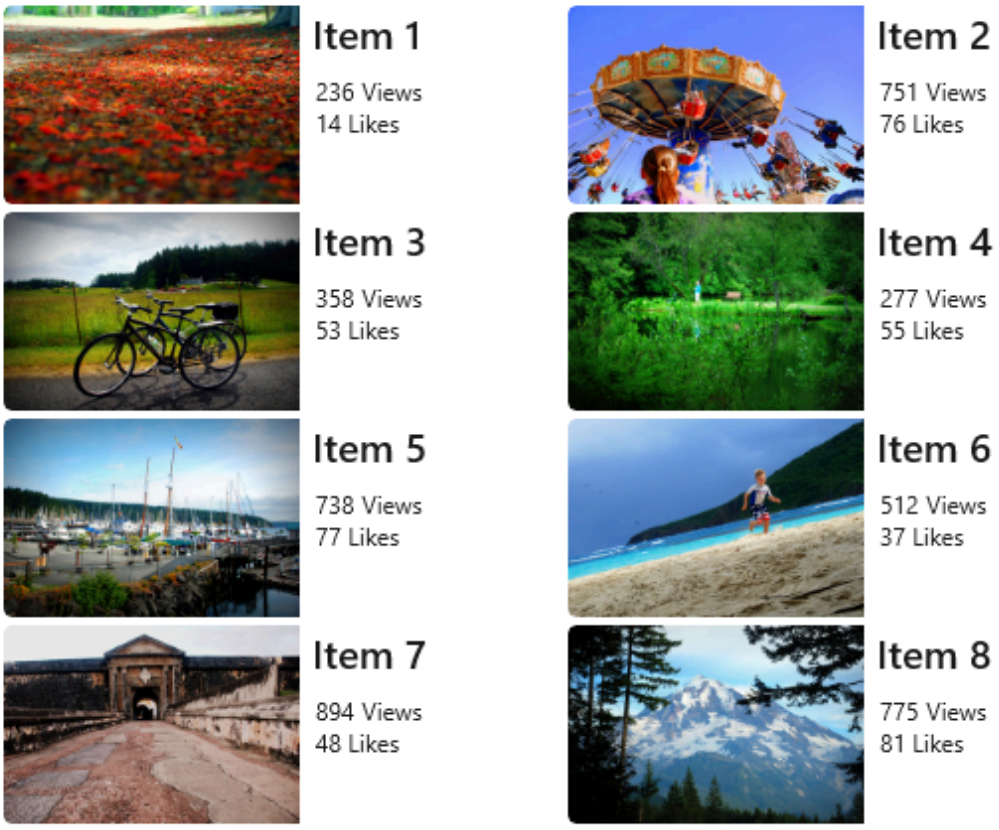
ListView

The ListView control displays data stacked vertically in a single column. ListView works better for items that have text as a focal point, and for collections that are meant to be read top to bottom (for example, alphabetically ordered). A few common use cases for ListView include lists of messages and search results. If you need to display collections in multiple columns or in a table-like format, you should *not* use ListView. Instead, consider using a DataGrid control.



GridView

The GridView control presents a collection of items in rows and columns that can be scrolled vertically. Data is stacked horizontally until it fills a column, and then continues with the next row in the column. GridView works better for collections that have images as their focal point or whose items can be read from side-to-side or are not sorted in a specific order. A common use case for GridView is a photo or product gallery.



Which collection control should you use? A Comparison with ItemsRepeater

It's important to understand the differences between these types of controls before you decide which one to use.

ListView and GridView

The feature-rich ListView and GridView controls work *out of box*. They require no customization, but they can be customized easily. Each has its own built-in UI and UX and is designed to display nearly any type of collection as is.

ItemsRepeater

The ItemsRepeater control also is used to display collections, but it's designed as a building block for creating a custom control to suit your particular UI requirements. It doesn't have the same built-in features and functionality as ListView and GridView, so you'll need to implement any necessary features or interactions. Use ItemsRepeater if

you have a highly customized UI that you can't create by using either ListView or GridView, or if your data source requires different behavior for each item.

Learn more about ItemsRepeater by reading its Guidelines and API documentation.

Create a list view or a grid view

✔ **Important APIs**: ListView class, GridView class, ItemsSource property, Items property

Open the WinUI 3 Gallery app and see the **ListView** or the **GridView** in action.

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

ListView and GridView are both ItemsControl types, so they can contain a collection of items of any type. A ListView or GridView control must have items in its Items collection before it can display anything on the screen. To populate the view, you can add items directly to the collection or set the ItemsSource property to a data source.

> ⊗ **Caution**
>
> You can use either the Items or ItemsSource property to populate the list, but you can't use both at the same time. If you set the ItemsSource property and you add an item in XAML, the added item is ignored. If you set the ItemsSource property and you add an item to the Items collection in code, an exception is thrown.

Many of the examples in this article populate the Items collection directly for the sake of simplicity. However, it's more common for the items in a list to come from a dynamic source, such as a list of books from an online database. You use the ItemsSource property for this purpose.

Add items to a ListView or GridView control

You can add items to the ListView or GridView Items collection by using either XAML or code to yield the same result. You would ordinarily add items through XAML if you have a small number of items that don't change and are easily defined, or if you generate the items in code at runtime.

Method 1: Add items to the Items collection

- Option 1: Add items through XAML

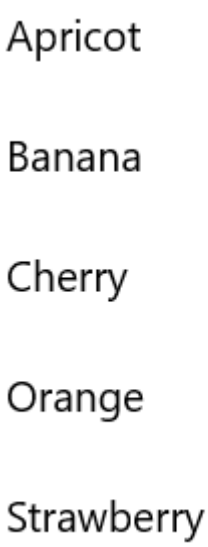
```
XAML

<!-- No corresponding C# code is needed for this example. -->

<ListView x:Name="Fruits">
<x:String>Apricot</x:String>
<x:String>Banana</x:String>
<x:String>Cherry</x:String>
<x:String>Orange</x:String>
<x:String>Strawberry</x:String>
</ListView>
```

- Option 2: Add items through code

```
XAML

<StackPanel Name="FruitsPanel"></StackPanel>
```

```
C#

// Create a new ListView and add content.
ListView Fruits = new ListView();
Fruits.Items.Add("Apricot");
Fruits.Items.Add("Banana");
Fruits.Items.Add("Cherry");
Fruits.Items.Add("Orange");
Fruits.Items.Add("Strawberry");

// Add the ListView to a parent container in the visual tree (which you
created in the corresponding XAML file).
FruitsPanel.Children.Add(Fruits);
```

Both of these options produce the same list view, as shown here:



Method 2: Add items by setting the ItemsSource property

You would ordinarily use a ListView or GridView to display data from a source such as a database or the internet. To populate a ListView or GridView control from a data source, you set its ItemsSource property to a collection of data items. This method works better if ListView or GridView is going to hold custom class objects, as shown in the following examples.

- Option 1: Set ItemsSource in code

  Here, the ListView ItemsSource property is set in code directly to an instance of a collection.

  XAML

  ```
  <StackPanel x:Name="ContactPanel"></StackPanel>
  ```

  C#

  ```
  // Class definition should be provided within the namespace being used,
  outside of any other classes.

  this.InitializeComponent();

  // Instead of adding hard coded items to an ObservableCollection as
  shown here,
  //the data could be pulled asynchronously from a database or the
  internet.
  ObservableCollection<Contact> Contacts = new
  ObservableCollection<Contact>();

  // You create Contact objects by providing a first name, last name, and
  company for the Contact constructor.
  ```

```
// They are then added to the ObservableCollection Contacts.
Contacts.Add(new Contact("John", "Doe", "Contoso, LTD."));
Contacts.Add(new Contact("Jane", "Doe", "Fabrikam, Inc."));
Contacts.Add(new Contact("Santa", "Claus", "Alpine Ski House"));

// Create a new ListView (or GridView) for the UI, and add content by
setting ItemsSource
ListView ContactsLV = new ListView();
ContactsLV.ItemsSource = Contacts;

// Add the ListView to a parent container in the visual tree (which you
created in the corresponding XAML file)
ContactPanel.Children.Add(ContactsLV);
```

- Option 2: Set ItemsSource in XAML

  You can also bind the ItemsSource property to a collection in the XAML. Here, ItemsSource is bound to a public property named Contacts, which exposes the page's private data collection, named *_contacts*.

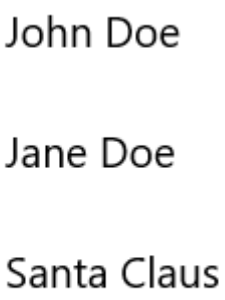
XAML

```
<ListView x:Name="ContactsLV" ItemsSource="{x:Bind Contacts}"/>
```

C#

```
// Provide a class definition within the namespace being used, outside
of any other classes.
// These two declarations belong outside the main page class.
private ObservableCollection<Contact> _contacts = new
ObservableCollection<Contact>();

public ObservableCollection<Contact> Contacts
{
    get { return this._contacts; }
}

// Define this method within your main page class.
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    base.OnNavigatedTo(e);

    // Instead of hard coded items, the data could be pulled
    // asynchronously from a database or the internet.
    Contacts.Add(new Contact("John", "Doe", "Contoso, LTD."));
    Contacts.Add(new Contact("Jane", "Doe", "Fabrikam, Inc."));
    Contacts.Add(new Contact("Santa", "Claus", "Alpine Ski House"));
}
```

Both of these options will produce the same list view, as shown in the following screenshot. (The list view shows the string representation of each item, because a data template isn't defined for this exercise.)



> **Important**
>
> Without a defined data template, custom class objects will appear in the list view with their string value only if they have a defined ToString method.

The next section goes into greater detail about how to visually represent simple and custom class items properly in a ListView or GridView template.

For more info about data binding, see Data binding overview.

> **Note**
>
> If you need to show grouped data in your list view, you must bind to a CollectionViewSource class. CollectionViewSource acts as a proxy for the collection class in XAML and enables grouping support. For more info, see CollectionViewSource.

Customize the look with a data template

By using a data template in a ListView or GridView control, you can define how the items and data are to be visualized. By default, a data item is displayed in the list view as the string representation of the data object it's bound to. You can show the string representation of a particular property of the data item by setting DisplayMemberPath to that property.

However, you might ordinarily want to show a richer presentation of your data. To specify how items in the list view or grid view are to be displayed, you create a

DataTemplate class. The XAML in DataTemplate defines the layout and appearance of the controls that are used to display an individual item. The controls in the layout can be bound to the properties of a data object, or they can have static content that's defined inline.

> **ⓘ Important**
>
> When you use the x:Bind markup extension in DataTemplate, you have to specify the data type (`x:DataType`) on the data template.

A simple ListView data template

In this example, the data item is a simple string. To add an image to the left of the string, and to display the string in teal, you define DataTemplate inline within the ListView definition. This is the same ListView control that you created earlier by using option 1 under method 1.

XAML

```
<!--No corresponding code is needed for this example.-->
<ListView x:Name="FruitsList">
                <ListView.ItemTemplate>
                    <DataTemplate x:DataType="x:String">
                        <Grid>
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="47"/>
                                <ColumnDefinition/>
                            </Grid.ColumnDefinitions>
                            <Image Source="Assets/placeholder.png"
Width="32" Height="32"
                                HorizontalAlignment="Left"
VerticalAlignment="Center"/>
                            <TextBlock Text="{x:Bind}" Foreground="Teal"
FontSize="14"
                                Grid.Column="1" VerticalAlignment="Center"/>
                        </Grid>
                    </DataTemplate>
                </ListView.ItemTemplate>
                <x:String>Apricot</x:String>
                <x:String>Banana</x:String>
                <x:String>Cherry</x:String>
                <x:String>Orange</x:String>
                <x:String>Strawberry</x:String>
            </ListView>
```

Here's how the data items are displayed when you apply a simple ListView data template:

A ListView data template for custom class objects

In the following example, the data item is a Contact object. To add the contact image to the left of the Contact name and company, you define DataTemplate inline within the ListView definition. This ListView data template was created in option 2 under method 2, as shown earlier.

XAML

```
<ListView x:Name="ContactsLV" ItemsSource="{x:Bind Contacts}">
    <ListView.ItemTemplate>
        <DataTemplate x:DataType="local:Contact">
            <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition Height="*"/>
                    <RowDefinition Height="*"/>
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto"/>
                    <ColumnDefinition Width="*"/>
                </Grid.ColumnDefinitions>
                <Image Grid.Column="0" Grid.RowSpan="2" Source="Assets/grey-
placeholder.png" Width="32"
                    Height="32" HorizontalAlignment="Center"
VerticalAlignment="Center"></Image>
                <TextBlock Grid.Column="1" Text="{x:Bind Name}"
Margin="12,6,0,0"
                    Style="{ThemeResource BaseTextBlockStyle}"/>
                <TextBlock  Grid.Column="1" Grid.Row="1" Text="{x:Bind
Company}" Margin="12,0,0,6"
```

```
                    Style="{ThemeResource BodyTextBlockStyle}"/>
            </Grid>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Here's how the data items are displayed when you apply a ListView data template for custom class objects:

Data templates are the primary way you define the look of your ListView. They can also significantly affect performance if your list holds a large number of items.

You can define your data template inline within the ListView or GridView definition, as shown in the preceding code, or separately in a Resources section. If you define it outside the ListView or GridView definition, you must give the data template an x:Key attribute and assign it to the ItemTemplate property of the ListView or GridView by using that key.

For more info and examples of how to use data templates and item containers to define the look of items in your list or grid, see Item containers and templates.

Change the layout of items

When you add items to a ListView or GridView control, it automatically wraps each item in an item container and then lays out all of the item containers. How these item containers are laid out depends on the ItemsPanel property of the control.

- **ListView**, by default, uses ItemsStackPanel, which produces a vertical list:

Item 1

Item 2

Item 3

Item 4

Item 5

- **GridView** uses ItemsWrapGrid, which adds items horizontally, and wraps and scrolls vertically:

Item 1 Item 2 Item 3

Item 4 Item 5

You can modify the layout of items by adjusting the properties on the items panel, or you can replace the default panel with another panel.

> ⓘ **Note**
>
> If you change ItemsPanel, do *not* disable virtualization. Both ItemsStackPanel and ItemsWrapGrid support virtualization, so these classes are safe to use. If you use any other panel, you might disable virtualization and slow the performance of the list view. For more info, see the list view articles under **Performance**.

This example shows how to make a ListView control lay out its item containers in a horizontal list by changing the Orientation property of ItemsStackPanel.

Because the list view scrolls vertically, by default, you also need to adjust some properties on the list view's internal ScrollViewer to make it scroll horizontally.

- Change ScrollViewer.HorizontalScrollMode to **Enabled** or **Auto**.
- Change ScrollViewer.HorizontalScrollBarVisibility to **Auto**.
- Change ScrollViewer.VerticalScrollMode to **Disabled**.
- Change ScrollViewer.VerticalScrollBarVisibility to **Hidden**.

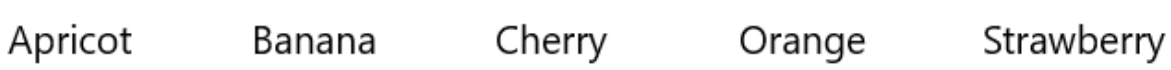
> ⓘ **Important**
>
> The following examples are shown with the list view width unconstrained, so the horizontal scrollbars aren't displayed. If you run this code, you can set `Width="180"`

for ListView to display the scrollbars.

XAML

```
<ListView Height="60"
          ScrollViewer.HorizontalScrollMode="Enabled"
          ScrollViewer.HorizontalScrollBarVisibility="Auto"
          ScrollViewer.VerticalScrollMode="Disabled"
          ScrollViewer.VerticalScrollBarVisibility="Hidden">
    <ListView.ItemsPanel>
        <ItemsPanelTemplate>
            <ItemsStackPanel Orientation="Horizontal"/>
        </ItemsPanelTemplate>
    </ListView.ItemsPanel>
    <x:String>Apricot</x:String>
    <x:String>Banana</x:String>
    <x:String>Cherry</x:String>
    <x:String>Orange</x:String>
    <x:String>Strawberry</x:String>
</ListView>
```

Here's how the list is displayed:



In the next example, ListView lays out items in a vertical wrapping list by using ItemsWrapGrid instead of ItemsStackPanel.

ⓘ Important

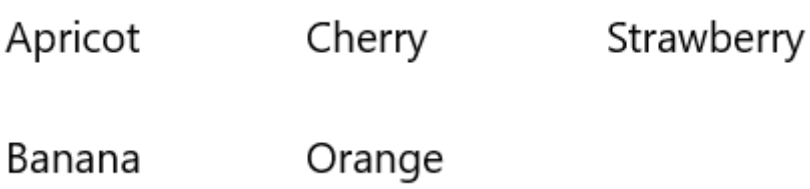
You must constrain the height of the list view to force the control to wrap the containers.

XAML

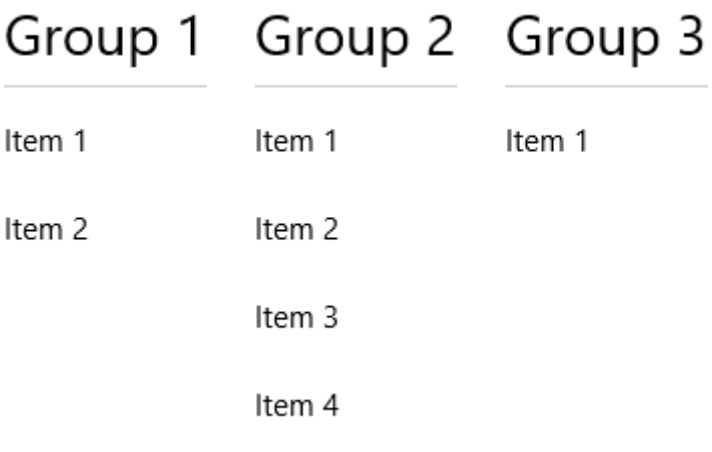
```
<ListView Height="100"
          ScrollViewer.HorizontalScrollMode="Enabled"
          ScrollViewer.HorizontalScrollBarVisibility="Auto"
          ScrollViewer.VerticalScrollMode="Disabled"
          ScrollViewer.VerticalScrollBarVisibility="Hidden">
    <ListView.ItemsPanel>
        <ItemsPanelTemplate>
            <ItemsWrapGrid/>
        </ItemsPanelTemplate>
    </ListView.ItemsPanel>
    <x:String>Apricot</x:String>
```

```
    <x:String>Banana</x:String>
    <x:String>Cherry</x:String>
    <x:String>Orange</x:String>
    <x:String>Strawberry</x:String>
</ListView>
```

Here's how the list is displayed:



If you show grouped data in your list view, ItemsPanel determines how the item groups are laid out, not how the individual items are laid out. For example, if you use the previously shown horizontal ItemsStackPanel to show grouped data, the groups are arranged horizontally, but the items in each group are still stacked vertically, as shown here:



Item selection and interaction

You can choose from a variety of ways to let users interact with a list view. By default, users can select a single item. You can change the SelectionMode property to enable multi-selection or to disable selection. You can set the IsItemClickEnabled property so that users click an item (for example, a button) to invoke an action instead of selecting the item.

> ⓘ **Note**
>
> Both ListView and GridView use the ListViewSelectionMode enumeration for their SelectionMode properties. IsItemClickEnabled is set to **False** by default, so you

need only to set it to enable click mode.

This table shows the ways a user can interact with a list view, and how you can respond to the interaction.

Expand table

To enable this interaction:	Use these settings:	Handle this event:	Use this property to get the selected item:
No interaction	SelectionMode="None" IsItemClickEnabled="False"	N/A	N/A
Single selection	SelectionMode="Single" IsItemClickEnabled="False"	SelectionChanged	SelectedItem SelectedIndex
Multiple selection	SelectionMode="Multiple" IsItemClickEnabled="False"	SelectionChanged	SelectedItems
Extended selection	SelectionMode="Extended" IsItemClickEnabled="False"	SelectionChanged	SelectedItems
Click	SelectionMode="None" IsItemClickEnabled="True"	ItemClick	N/A

> **Note**
>
> You can enable IsItemClickEnabled to raise an ItemClick event while SelectionMode is also set to **Single**, **Multiple**, or **Extended**. If you do this, the ItemClick event is raised first, and then the SelectionChanged event is raised. In some cases (for example, if you go to another page in the ItemClick event handler), the SelectionChanged event isn't raised and the item isn't selected.

You can set these properties in XAML or in code, as shown here:

XAML

```
<ListView x:Name="myListView" SelectionMode="Multiple"/>

<GridView x:Name="myGridView" SelectionMode="None"
IsItemClickEnabled="True"/>
```

C#

```
myListView.SelectionMode = ListViewSelectionMode.Multiple;

myGridView.SelectionMode = ListViewSelectionMode.None;
myGridView.IsItemClickEnabled = true;
```

Read-only

You can set the SelectionMode property to **ListViewSelectionMode.None** to disable item selection. This puts the control in read-only mode, so that it's used for displaying data, but not for interacting with it. That is, the item selection is disabled, but the control itself is not.

Single selection

This table describes keyboard, mouse, and touch interactions when SelectionMode is set to **Single**.

Expand table

Modifier key	Interaction
None	• Users can select a single item by using the space bar, mouse clicks, or taps.
Ctrl	• Users can deselect a single item by using the space bar, mouse clicks, or taps. • By using the arrow keys, users can move the focus independent of the selection.

When SelectionMode is set to **Single**, you can get the selected data item from the SelectedItem property. You can get the index in the collection of the selected item by using the SelectedIndex property. If no item is selected, SelectedItem is **null**, and SelectedIndex is -1.

If you try to set an item that's not in the Items collection as SelectedItem, the operation is ignored and SelectedItem is **null**. However, if you try to set SelectedIndex to an index that's out of the range of the items in the list, a System.ArgumentException exception occurs.

Multiple selection

This table describes the keyboard, mouse, and touch interactions when SelectionMode is set to **Multiple**.

Expand table

Modifier key	Interaction
None	• Users can select multiple items by using the space bar, mouse clicks, or taps to select the focused item. • By using the arrow keys, users can move the focus independent of their selection.
Shift	• Users can select multiple contiguous items by clicking or tapping the first item in the selection and then clicking or tapping the last item in the selection. • By using the arrow keys, users can select contiguous items starting with the item that's selected when they select the Shift key.

Extended selection

This table describes the keyboard, mouse, and touch interactions when SelectionMode is set to **Extended**.

Expand table

Modifier key	Interaction
None	• The behavior is the same as **Single** selection.
Ctrl	• Users can select multiple items by using the space bar, mouse clicks, or taps to select the focused item. • By using the arrow keys, users can move the focus independent of the selection.
Shift	• Users can select multiple contiguous items by clicking or tapping the first item in the selection and then clicking or tapping the last item in the selection. • By using the arrow keys, users can select contiguous items starting with the item that's selected when they select the Shift key.

When SelectionMode is set to **Multiple** or **Extended**, you can get the selected data items from the SelectedItems property.

The SelectedIndex, SelectedItem, and SelectedItems properties are synchronized. For example, if you set SelectedIndex to -1, SelectedItem is set to **null** and SelectedItems is empty. And if you set SelectedItem to **null**, SelectedIndex is set to -1 and SelectedItems is empty.

In multi-select mode, SelectedItem contains the item that was selected first, and Selectedindex contains the index of the item that was selected first.

Respond to selection changes

To respond to selection changes in a list view, handle the SelectionChanged event. In the event handler code, you can get the list of selected items from the SelectionChangedEventArgs.AddedItems property. You can get any items that were deselected from the SelectionChangedEventArgs.RemovedItems property. The AddedItems and RemovedItems collections contain at most one item, unless users select a range of items by holding down the Shift key.

The following example shows how to handle the SelectionChanged event and access the various Item collections:

XAML

```
<StackPanel HorizontalAlignment="Right">
    <ListView x:Name="listView1" SelectionMode="Multiple"
              SelectionChanged="ListView1_SelectionChanged">
        <x:String>Apricot</x:String>
        <x:String>Banana</x:String>
        <x:String>Cherry</x:String>
        <x:String>Orange</x:String>
        <x:String>Strawberry</x:String>
    </ListView>
    <TextBlock x:Name="selectedItem"/>
    <TextBlock x:Name="selectedIndex"/>
    <TextBlock x:Name="selectedItemCount"/>
    <TextBlock x:Name="addedItems"/>
    <TextBlock x:Name="removedItems"/>
</StackPanel>
```

C#

```
private void ListView1_SelectionChanged(object sender,
SelectionChangedEventArgs e)
{
    if (listView1.SelectedItem != null)
    {
        selectedItem.Text =
            "Selected item: " + listView1.SelectedItem.ToString();
    }
    else
    {
        selectedItem.Text =
            "Selected item: null";
    }
    selectedIndex.Text =
        "Selected index: " + listView1.SelectedIndex.ToString();
    selectedItemCount.Text =
        "Items selected: " + listView1.SelectedItems.Count.ToString();
    addedItems.Text =
```

```
        "Added: " + e.AddedItems.Count.ToString();
    removedItems.Text =
        "Removed: " + e.RemovedItems.Count.ToString();
}
```

Click mode

You can change a list view so that users click buttons and other items instead of selecting them. For example, this is useful if your app opens a new page when users click an item in a list or grid.

To enable this behavior:

- Set SelectionMode to **None**.
- Set IsItemClickEnabled to **True**.
- Handle the ItemClick event to do something when users click an item.

Here's a list view with clickable items. The code in the ItemClick event handler opens a new page in the app.

XAML

```
<ListView SelectionMode="None"
          IsItemClickEnabled="True"
          ItemClick="ListView1_ItemClick">
    <x:String>Page 1</x:String>
    <x:String>Page 2</x:String>
    <x:String>Page 3</x:String>
    <x:String>Page 4</x:String>
    <x:String>Page 5</x:String>
</ListView>
```

C#

```
private void ListView1_ItemClick(object sender, ItemClickEventArgs e)
{
    switch (e.ClickedItem.ToString())
    {
        case "Page 1":
            this.Frame.Navigate(typeof(Page1));
            break;

        case "Page 2":
            this.Frame.Navigate(typeof(Page2));
            break;

        case "Page 3":
            this.Frame.Navigate(typeof(Page3));
```

```
            break;

        case "Page 4":
            this.Frame.Navigate(typeof(Page4));
            break;

        case "Page 5":
            this.Frame.Navigate(typeof(Page5));
            break;

        default:
            break;
    }
}
```

Select a range of items programmatically

Sometimes, you might need to manipulate a ListView item selection programmatically. For example, you might display a **Select all** button to let users select all items in a list. In this case, it's usually not very efficient to add and remove items from the SelectedItems collection one by one. Each item change causes a SelectionChanged event, and when you work with the items directly instead of working with index values, the item is de-virtualized.

It's more efficient to use the SelectAll, SelectRange, and DeselectRange methods to modify the selection than to use the SelectedItems property. These methods select (or deselect) items by using ranges of item indexes. Items that are virtualized remain virtualized, because only the index is used. All items in the specified range are selected (or deselected), regardless of their original selection state. The SelectionChanged event occurs only once for each call to these methods.

> **ⓘ Important**
>
> You should call these methods only when the SelectionMode property is set to **Multiple** or **Extended**. If you call SelectRange when SelectionMode is **Single** or **None**, an exception is thrown.

When you select items by using index ranges, use the SelectedRanges property to get all selected ranges in the list.

If the ItemsSource property implements IItemsRangeInfo, and you use these methods to modify the selection, the AddedItems and RemovedItems properties aren't set in SelectionChangedEventArgs. Setting these properties requires de-virtualizing the item object. Use the SelectedRanges property to get the items instead.

You can select all items in a collection by calling the SelectAll method. However, there is no corresponding method to deselect all items. You can deselect all items by calling DeselectRange and passing an ItemIndexRange with a FirstIndex value of 0 and a Length value equal to the number of items in the collection. This is shown in the following example, along with an option to select all items.

XAML

```
<StackPanel Width="160">
    <Button Content="Select all" Click="SelectAllButton_Click"/>
    <Button Content="Deselect all" Click="DeselectAllButton_Click"/>
    <ListView x:Name="listView1" SelectionMode="Multiple">
        <x:String>Apricot</x:String>
        <x:String>Banana</x:String>
        <x:String>Cherry</x:String>
        <x:String>Orange</x:String>
        <x:String>Strawberry</x:String>
    </ListView>
</StackPanel>
```

C#

```
private void SelectAllButton_Click(object sender, RoutedEventArgs e)
{
    if (listView1.SelectionMode == ListViewSelectionMode.Multiple ||
        listView1.SelectionMode == ListViewSelectionMode.Extended)
    {
        listView1.SelectAll();
    }
}

private void DeselectAllButton_Click(object sender, RoutedEventArgs e)
{
    if (listView1.SelectionMode == ListViewSelectionMode.Multiple ||
        listView1.SelectionMode == ListViewSelectionMode.Extended)
    {
        listView1.DeselectRange(new ItemIndexRange(0,
(uint)listView1.Items.Count));
    }
}
```

For info about how to change the look of selected items, see Item containers and templates.

Drag and drop

ListView and GridView controls support dragging and dropping items within their own controls, and between themselves and other ListView and GridView controls. For more

info about implementing drag and drop functionality, see Drag and drop.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for these controls exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** ListView class, GridView class, ItemsSource property, Items property
- ✓ Open the WinUI 2 Gallery app and see the ListView or the GridView in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

Get the sample code

- XAML ListView and GridView sample: Demonstrates the ListView and GridView controls.
- XAML drag and drop sample: Demonstrates drag and drop with the ListView control.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls.

Related articles

- Lists
- Item containers and templates
- Drag and drop

Flip view

Article • 02/26/2025

Use a flip view for browsing images or other items in a collection, such as photos in an album or items in a product details page, one item at a time. For touch devices, swiping across an item moves through the collection. For a mouse, navigation buttons appear on mouse hover. For a keyboard, arrow keys move through the collection.

Is this the right control?

Flip view is best for perusing images in small to medium collections (up to 25 or so items). Examples of such collections include items in a product details page or photos in a photo album. Although we don't recommend flip view for most large collections, the control is common for viewing individual images in a photo album.

Recommendations

- Flip views work best for collections of up to 25 or so items.
- Avoid using a flip view control for larger collections, as the repetitive motion of flipping through each item can be tedious. An exception would be for photo albums, which often have hundreds or thousands of images. Photo albums almost always switch to a flip view once a photo has been selected in the grid view layout. For other large collections, consider a List view or grid view.

Globalization and localization checklist

- **Bi-directional considerations**: Use standard mirroring for RTL languages. The back and forward controls should be based on the language's direction, so for RTL languages, the right button should navigate backwards and the left button should navigate forward.

Examples

Horizontal browsing, starting at the left-most item and flipping right, is the typical layout for a flip view. This layout works well in either portrait or landscape orientation on all devices:

A flip view can also be browsed vertically:

Create a flip view

✓ **Important APIs**: FlipView class, ItemsSource property, ItemTemplate property

Open the WinUI 3 Gallery app and see the FlipView in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

FlipView is an ItemsControl, so it can contain a collection of items of any type. To populate the view, add items to the **Items** collection, or set the **ItemsSource** property to a data source.

By default, a data item is displayed in the flip view as the string representation of the data object it's bound to. To specify exactly how items in the flip view are displayed, you create a **DataTemplate** to define the layout of controls used to display an individual item. The controls in the layout can be bound to properties of a data object, or have content defined inline. You assign the DataTemplate to the **ItemTemplate** property of the FlipView.

Add items to the Items collection

You can add items to the **Items** collection using XAML or code. You typically add items this way if you have a small number of items that don't change and are easily defined in XAML, or if you generate the items in code at run time. Here's a flip view with items defined inline.

XAML

```
<FlipView x:Name="flipView1">
    <Image Source="Assets/Logo.png" />
    <Image Source="Assets/SplashScreen.png" />
    <Image Source="Assets/SmallLogo.png" />
</FlipView>
```

C#

```
// Create a new flip view, add content,
// and add a SelectionChanged event handler.
FlipView flipView1 = new FlipView();
flipView1.Items.Add("Item 1");
flipView1.Items.Add("Item 2");

// Add the flip view to a parent container in the visual tree.
stackPanel1.Children.Add(flipView1);
```

When you add items to a flip view they are automatically placed in a **FlipViewItem** container. To change how an item is displayed you can apply a style to the item container by setting the **ItemContainerStyle** property.

When you define the items in XAML, they are automatically added to the Items collection.

Set the items source

You typically use a flip view to display data from a source such as a database or the Internet. To populate a flip view from a data source, you set its **ItemsSource** property to a collection of data items.

Here, the flip view's ItemsSource is set in code directly to an instance of a collection.

C#

```
// Data source.
List<String> itemsList = new List<string>();
itemsList.Add("Item 1");
itemsList.Add("Item 2");

// Create a new flip view, add content,
// and add a SelectionChanged event handler.
FlipView flipView1 = new FlipView();
flipView1.ItemsSource = itemsList;
flipView1.SelectionChanged += FlipView_SelectionChanged;

// Add the flip view to a parent container in the visual tree.
stackPanel1.Children.Add(flipView1);
```

You can also bind the ItemsSource property to a collection in XAML. For more info, see Data binding with XAML.

Here, the ItemsSource is bound to a **CollectionViewSource** named `itemsViewSource`.

XAML

```
<Page.Resources>
    <!-- Collection of items displayed by this page -->
    <CollectionViewSource x:Name="itemsViewSource" Source="{Binding
Items}"/>
</Page.Resources>

...

<FlipView x:Name="itemFlipView"
          ItemsSource="{Binding Source={StaticResource itemsViewSource}}"/>
```

⊗ Caution

You can populate a flip view either by adding items to its Items collection, or by setting its ItemsSource property, but you can't use both ways at the same time. If you set the ItemsSource property and you add an item in XAML, the added item is

ignored. If you set the ItemsSource property and you add an item to the Items collection in code, an exception is thrown.

Specify the look of the items

By default, a data item is displayed in the flip view as the string representation of the data object it's bound to. You typically want to show a more rich presentation of your data. To specify exactly how items in the flip view are displayed, you create a **DataTemplate**. The XAML in the DataTemplate defines the layout and appearance of controls used to display an individual item. The controls in the layout can be bound to properties of a data object, or have content defined inline. The DataTemplate is assigned to the **ItemTemplate** property of the FlipView control.

In this example, the ItemTemplate of a FlipView is defined inline. An overlay is added to the image to display the image name.

XAML

```
<FlipView MaxWidth="400" Height="180" BorderBrush="Black"
BorderThickness="1"
          ItemsSource="{x:Bind Items, Mode=OneWay}">
    <FlipView.ItemTemplate>
        <DataTemplate x:DataType="data:ControlInfoDataItem">
            <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition Height="*" />
                    <RowDefinition Height="Auto" />
                </Grid.RowDefinitions>
                <Image Width="36" Source="{x:Bind ImagePath}"
Stretch="Uniform"
                       VerticalAlignment="Center" />
                <Border Background="#A5FFFFFF" Height="60" Grid.Row="1">
                    <TextBlock x:Name="Control2Text" Text="{x:Bind Title}"
Foreground="Black"
                               Padding="12,12" Style="{StaticResource
TitleTextBlockStyle}"
                               HorizontalAlignment="Center" />
                </Border>
            </Grid>
        </DataTemplate>
    </FlipView.ItemTemplate>
</FlipView>
```

Here's what the layout defined by the data template looks like.

Set the orientation of the flip view

By default, the flip view flips horizontally. To make the it flip vertically, use a stack panel with a vertical orientation as the flip view's **ItemsPanel**.

This example shows how to use a stack panel with a vertical orientation as the ItemsPanel of a FlipView.

XAML

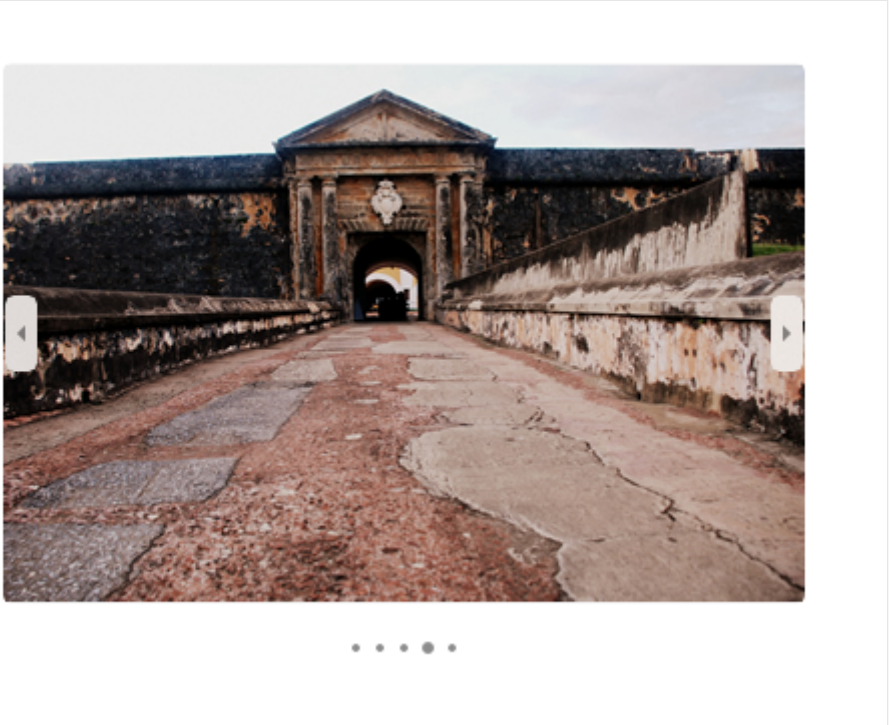
```
<FlipView x:Name="flipViewVertical" Width="480" Height="270"
          BorderBrush="Black" BorderThickness="1">

    <!-- Use a vertical stack panel for vertical flipping. -->
    <FlipView.ItemsPanel>
        <ItemsPanelTemplate>
            <VirtualizingStackPanel Orientation="Vertical"/>
        </ItemsPanelTemplate>
    </FlipView.ItemsPanel>

    <FlipView.ItemTemplate>
        <DataTemplate>
            <Grid>
                <Image Width="480" Height="270" Stretch="UniformToFill"
                       Source="{Binding Image}"/>
                <Border Background="#A5000000" Height="80"
VerticalAlignment="Bottom">
                    <TextBlock Text="{Binding Name}"
                               FontFamily="Segoe UI" FontSize="26.667"
                               Foreground="#CCFFFFFF" Padding="15,20"/>
                </Border>
            </Grid>
        </DataTemplate>
    </FlipView.ItemTemplate>
</FlipView>
```

Here's what the flip view looks like with a vertical orientation.

Adding a context indicator

Use a context indicator (such as a PipsPager or film strip) with a flip view to help provide users with a point of reference within the content.

The following image shows a PipsPager used with a small photo gallery (we recommend centering the PipsPager below the gallery).



This code snippet shows how to bind a PipsPager with a FlipView.

XAML

```
<StackPanel>
  <FlipView x:Name="Gallery" MaxWidth="400" Height="270" ItemsSource="
{x:Bind Pictures}">
      <FlipView.ItemTemplate>
          <DataTemplate x:DataType="x:String">
              <Image Source="{x:Bind Mode=OneWay}"/>
          </DataTemplate>
      </FlipView.ItemTemplate>
  </FlipView>

  <!-- The SelectedPageIndex is bound to the FlipView to keep the two in
sync -->
  <PipsPager x:Name="FlipViewPipsPager"
      HorizontalAlignment="Center"
      Margin="0, 10, 0, 0"
      NumberOfPages="{x:Bind Pictures.Count}"
      SelectedPageIndex="{x:Bind Path=Gallery.SelectedIndex, Mode=TwoWay}"
/>
</StackPanel>
```

For larger collections (10 or more items), we highly recommend using a contextual indicator such as a film strip of thumbnails. Unlike a PipsPager that uses simple dots or glyphs, each thumbnail in the film strip shows a smaller, selectable version of the corresponding image.

For a full example showing how to add a context indicator to a FlipView, see XAML FlipView sample.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the **Windows App SDK** and **WinUI 3**, but are generally applicable to UWP apps that use **WinUI 2**. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** FlipView class, ItemsSource property, ItemTemplate property
- ✓ Open the WinUI 2 Gallery app and see the FlipView in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related articles

- Guidelines for lists
- PipsPager
- **FlipView class**

PipsPager

Article • 02/26/2025

The PipsPager control helps users navigate within linearly paginated content using a configurable collection of glyphs, each of which represents a single "page" within a limitless range. The glyphs highlight the current page, and indicate the availability of both preceding and succeeding pages. The control relies on current context and does not support explicit page numbering or a non-linear organization.

What is a pip?

Pips represent a unit of numerical value, typically rendered as dots. However they can be customized to use other glyphs such as dashes or squares.

By default, each solid dot in the PipsPager control represents a page in the content layout. A user can select a dot to navigate to that page in the content.

Is this the right control?

Use a PipsPager for content organized in a linear structure, is not explicitly numbered, or where a glyph-based representation of numbered pages is desired.

This UI is commonly used in apps such as photo viewers and app lists, where display space is limited and the number of potential pages is infinite.

Recommendations

- Common UI patterns for a PipsPager include photo viewers, app lists, carousels, and layouts where display space is limited.
- For experiences optimized for gamepad input, we recommend against placing UI directly to the left or right of a horizontal PipsPager, and above or below a vertically oriented PipsPager.
- For experiences optimized for touch input, we recommend integrating the PipsPager with a view control, such as a FlipView, to take advantage of on-content pagination with touch (the user can also use touch to select individual pips).

Create a PipsPager

✔ **Important APIs**: PipsPager class

Open the WinUI 3 Gallery app and see the PipsPager in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

A default PipsPager is comprised of five visible pips that can be oriented horizontally (default) or vertically.

A PipsPager also supports navigation buttons (previous, next) to move to an incrementally adjacent page. By default, the navigation buttons are collapsed and do not take up layout space.

Wrapping between the first and last items is not supported.

XAML

```
<PipsPager x:Name="DefaultPipsPager" />
```

Horizontal PipsPager with navigation buttons

The navigation buttons (previous, next) let the user move to an incrementally adjacent page.

By default, the navigation buttons are collapsed. You can control this behavior through the PreviousButtonVisibility and NextButtonVisibility properties.

Possible values for these properties are:

- Collapsed: The button is not visible to the user and **does not** take up layout space. (Default)

- Visible: The button is visible and enabled. Each button is automatically hidden when the PipsPager is at the minimum or maximum extent of the content. For example, if the current page is the first page then the previous button is hidden; if the current page is the last page then the next button is hidden. When hidden, the button is not visible but **does** take up layout space.
- VisibleOnPointerOver: The behavior is the same as Visible *except* that the button is only displayed when the user hovers the pointer cursor over the PipsPager UI, or the user sets keyboard focus on the PipsPager.

XAML

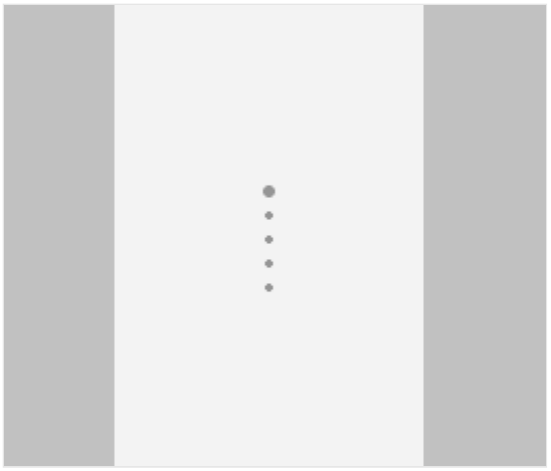
```
<PipsPager x:Name="VisibleButtonPipsPager"
    NumberOfPages="5"
    PreviousButtonVisibility="Visible"
    NextButtonVisibility="Visible" />
```

Vertical PipsPager with navigation buttons visible on pointer over

The PipsPager can be oriented vertically with no change to behavior or interaction experience.

The top button corresponds to the first button and the bottom button corresponds to the last button in the horizontal view.

The following example demonstrates the VisibleOnPointerOver setting for the navigation buttons.



XAML

```
<PipsPager x:Name="VerticalPipsPager"
    NumberOfPages="5"
    Orientation="Vertical"
    PreviousButtonVisibility="VisibleOnPointerOver"
    NextButtonVisibility="VisibleOnPointerOver" />
```

Scrolling pips

If the content consists of a large number of pages (NumberOfPages), you can use the MaxVisiblePips property to set the number of visible, interactive pips.

If the value of NumberOfPages is greater than the value of MaxVisiblePips, the pips automatically scroll in order to center the selected page in the control. If the NumberOfPages is equal to or less than MaxVisiblePips, no scrolling occurs and the number of pips shown is the same as the value of NumberOfPages.

If the value of MaxVisiblePips is greater than the available layout space, the displayed pips are clipped. The number of pips displayed is the lesser of MaxVisiblePips and NumberOfPages.

By default, a maximum of five pips are visible.

XAML

```
<PipsPager x:Name="ScrollingPipsPager"
    NumberOfPages="20"
    MaxVisiblePips="10" />
```

Integrate PipsPager with a Collection control

A PipsPager is often used in conjunction with collection controls.

The following example shows how to bind a PipsPager with a FlipView and provide another way to navigate through content and indicate the current page.

> ⓘ **Note**
>
> To use the PipsPager as a page indicator **only** and disable user interactions, set the control's IsEnabled property to false in the control.

XAML

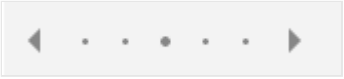
```
<StackPanel>
  <FlipView x:Name="Gallery" MaxWidth="400" Height="270" ItemsSource="
{x:Bind Pictures}">
      <FlipView.ItemTemplate>
          <DataTemplate x:DataType="x:String">
              <Image Source="{x:Bind Mode=OneWay}"/>
          </DataTemplate>
      </FlipView.ItemTemplate>
  </FlipView>
```

```
  <!-- The SelectedPageIndex is bound to the FlipView to keep the two in
sync -->
  <PipsPager x:Name="FlipViewPipsPager"
      HorizontalAlignment="Center"
      Margin="0, 10, 0, 0"
      NumberOfPages="{x:Bind Pictures.Count}"
      SelectedPageIndex="{x:Bind Path=Gallery.SelectedIndex, Mode=TwoWay}"
/>
</StackPanel>
```

Pip and navigation button customization

The navigation buttons and pips can be customized through the PreviousButtonStyle, NextButtonStyle, SelectedPipStyle, and NormalPipStyle properties.

If you set visibility through the PreviousButtonStyle or NextButtonStyle properties, these settings take precedence over the PreviousButtonVisibility or NextButtonVisibility properties, respectively (unless they are set to the PipsPagerButtonVisibility value of Collapsed).



XAML

```
<Page.Resources>
    <Style x:Key="NavButtonBaseStyle" TargetType="Button" BasedOn="
{StaticResource PipsPagerNavigationButtonBaseStyle}">
        <Setter Property="Width" Value="30" />
        <Setter Property="Height" Value="30" />
        <Setter Property="FontSize" Value="12" />
    </Style>

    <Style x:Key="PreviousButtonStyle" BasedOn="{StaticResource
NavButtonBaseStyle}" TargetType="Button">
        <Setter Property="Content" Value="&#xEDDB;" />
    </Style>

    <Style x:Key="NextButtonStyle" BasedOn="{StaticResource
NavButtonBaseStyle}" TargetType="Button">
        <Setter Property="Content" Value="&#xEDDC;" />
    </Style>
</Page.Resources>

<PipsPager x:Name="CustomNavButtonPipsPager"
    PreviousButtonStyle="{StaticResource PreviousButtonStyle}"
    NextButtonStyle="{StaticResource NextButtonStyle}"
    PreviousButtonVisibility="VisibleOnPointerOver"
    NextButtonVisibility="VisibleOnPointerOver" />
```

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The PipsPager for UWP apps requires WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in the Microsoft.UI.Xaml.Controls namespace.

- ✓ **WinUI 2 Apis:** PipsPager class
- ✓ Open the WinUI 2 Gallery app and see the PipsPager in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:PipsPager />
```

Related articles

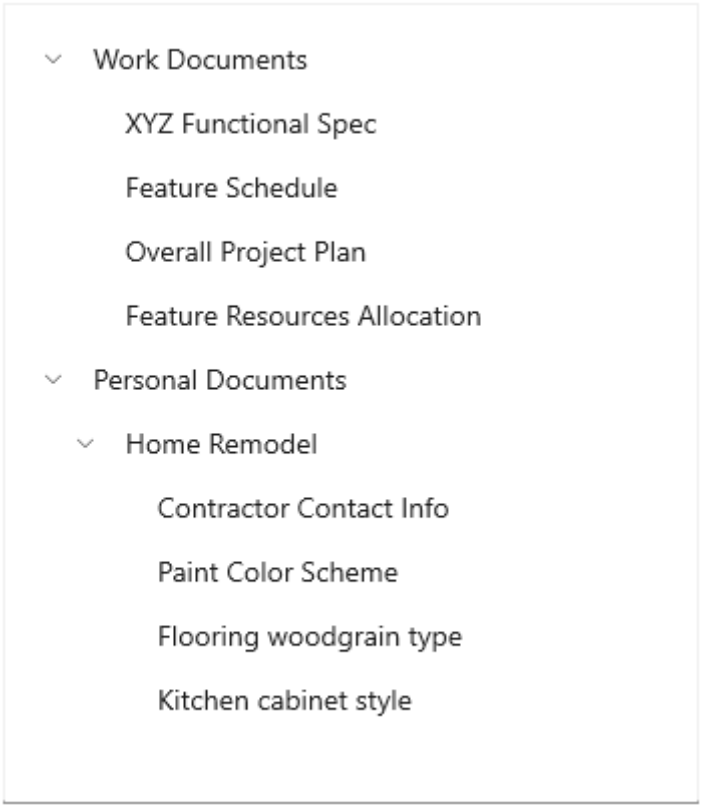
- ScrollViewer
- FlipView
- ItemsRepeater

Tree view

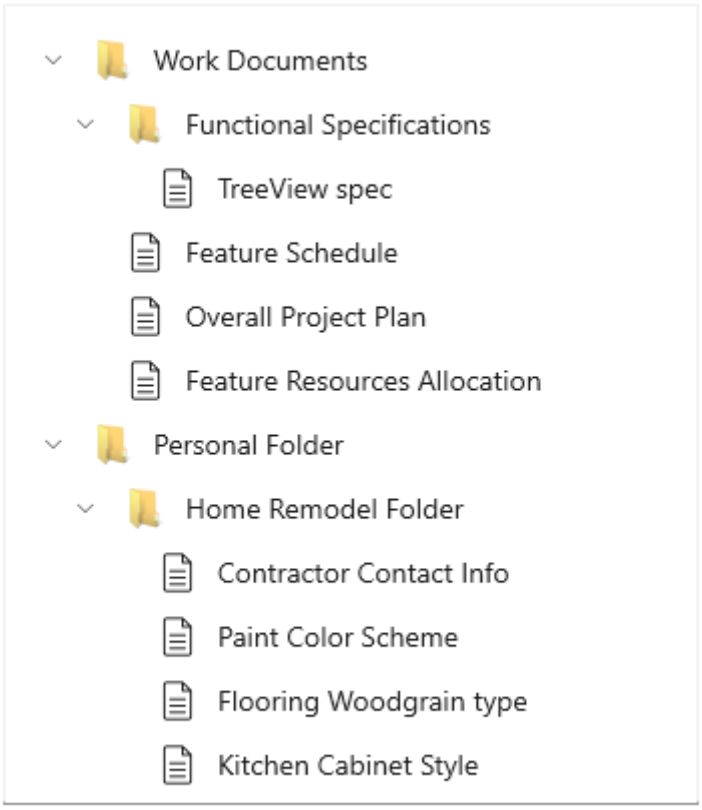
Article • 02/26/2025

The tree view control enables a hierarchical list with expanding and collapsing nodes that contain nested items. It can be used to illustrate a folder structure or nested relationships in your UI.

The tree view uses a combination of indentation and icons to represent the nested relationship between parent nodes and child nodes. Collapsed nodes use a chevron pointing to the right, and expanded nodes use a chevron pointing down.



You can include an icon in the tree view item data template to represent nodes. For example, if you show a file system hierarchy, you could use folder icons for the parent notes and file icons for the leaf nodes.



The **TreeView** APIs support the following features:

- N-level nesting
- Selection of single or multiple nodes
- Data binding to the **ItemsSource** property on **TreeView** and TreeViewItem
- **TreeViewItem** as the root of the **TreeView** item template
- Arbitrary types of content in a **TreeViewItem**
- Drag and drop between tree views

Is this the right control?

- Use a **TreeView** when items have nested list items, and if it is important to illustrate the hierarchical relationship of items to their peers and nodes.
- Avoid using **TreeView** if highlighting the nested relationship of an item is not a priority. For most drill-in scenarios, a regular list view is appropriate.

Create a tree view

You can create a tree view by binding the ItemsSource to a hierarchical data source, or you can create and manage **TreeViewNode** objects yourself.

To create a tree view, you use a TreeView control and a hierarchy of TreeViewNode objects. You create the node hierarchy by adding one or more root nodes to the **TreeView** control's RootNodes collection. Each **TreeViewNode** can then have more nodes added to its Children collection. You can nest tree view nodes to whatever depth you require.

You can bind a hierarchical data source to the ItemsSource property to provide the tree view content, just as you would with ListView's **ItemsSource**. Similarly, use ItemTemplate (and the optional ItemTemplateSelector) to provide a DataTemplate that renders the item.

> **Important**
>
> **ItemsSource** and its related APIs require Windows 10, version 1809 (SDK 17763) or later, or WinUI 2.
>
> **ItemsSource** is an alternative mechanism to **TreeView.RootNodes** for putting content into the **TreeView** control. You cannot set both **ItemsSource** and **RootNodes** at the same time. When you use **ItemsSource**, nodes are created for you, and you can access them from the **TreeView.RootNodes** property.

Here's an example of a simple tree view declared in XAML. You typically add the nodes in code, but we show the XAML hierarchy here because it can be helpful for visualizing how the hierarchy of nodes is created.

XAML

```
<muxc:TreeView>
    <muxc:TreeView.RootNodes>
        <muxc:TreeViewNode Content="Flavors"
                           IsExpanded="True">
            <muxc:TreeViewNode.Children>
                <muxc:TreeViewNode Content="Vanilla"/>
                <muxc:TreeViewNode Content="Strawberry"/>
                <muxc:TreeViewNode Content="Chocolate"/>
            </muxc:TreeViewNode.Children>
        </muxc:TreeViewNode>
    </muxc:TreeView.RootNodes>
</muxc:TreeView>
```

In most cases, your tree view displays data from a data source, so you typically declare the root **TreeView** control in XAML, but add the **TreeViewNode** objects in code or using data binding.

Bind to a hierarchical data source

To create a tree view using data binding, set a hierarchical collection to the **TreeView.ItemsSource** property. Then in the **ItemTemplate**, set the child items collection to the **TreeViewItem.ItemsSource** property.

XAML

```
<muxc:TreeView ItemsSource="{x:Bind DataSource}">
    <muxc:TreeView.ItemTemplate>
        <DataTemplate x:DataType="local:Item">
            <muxc:TreeViewItem ItemsSource="{x:Bind Children}"
                               Content="{x:Bind Name}"/>
        </DataTemplate>
    </muxc:TreeView.ItemTemplate>
</muxc:TreeView>
```

See Tree view using data binding for the full code.

Items and item containers

If you use **TreeView.ItemsSource**, these APIs are available to get the node or data item from the container, and vice versa.

Expand table

TreeViewItem	Description
TreeView.ItemFromContainer	Gets the data item for the specified **TreeViewItem** container.
TreeView.ContainerFromItem	Gets the **TreeViewItem** container for the specified data item.

Expand table

TreeViewNode	Description
TreeView.NodeFromContainer	Gets the **TreeViewNode** for the specified **TreeViewItem** container.
TreeView.ContainerFromNode	Gets the **TreeViewItem** container for the specified **TreeViewNode**.

Manage tree view nodes

This tree view is the same as the one created previously in XAML, but the nodes are created in code instead.

XAML

```
<muxc:TreeView x:Name="sampleTreeView"/>
```

C#

```
private void InitializeTreeView()
{
    muxc.TreeViewNode rootNode = new muxc.TreeViewNode() { Content =
"Flavors" };
    rootNode.IsExpanded = true;
    rootNode.Children.Add(new muxc.TreeViewNode() { Content = "Vanilla" });
    rootNode.Children.Add(new muxc.TreeViewNode() { Content = "Strawberry"
});
    rootNode.Children.Add(new muxc.TreeViewNode() { Content = "Chocolate"
});

    sampleTreeView.RootNodes.Add(rootNode);
}
```

These APIs are available for managing the data hierarchy of your tree view.

Expand table

TreeView	Description
RootNodes	A tree view can have one or more root nodes. Add a **TreeViewNode** object to the **RootNodes** collection to create a root node. The **Parent** of a root node is always **null**. The **Depth** of a root node is 0.

Expand table

TreeViewNode	Description
Children	Add **TreeViewNode** objects to the **Children** collection of a parent node to create your node hierarchy. A node is the **Parent** of all nodes in its **Children** collection.
HasChildren	**true** if the node has realized children. **false** indicates an empty folder or an item.
HasUnrealizedChildren	Use this property if you're filling nodes as they're expanded. See Fill a node when it's expanding later in this article.
Depth	Indicates how far from the root node a child node is.

TreeViewNode	Description
Parent	Gets the **TreeViewNode** that owns the **Children** collection that this node is part of.

The tree view uses the **HasChildren** and **HasUnrealizedChildren** properties to determine whether the expand/collapse icon is shown. If either property is **true**, the icon is shown; otherwise, it's not shown.

Tree view node content

You can store the data item that a tree view node represents in its Content property.

In the previous examples, the content was a simple string value. Here, a tree view node represents the user's **Pictures** folder, so the pictures library StorageFolder is assigned to the node's **Content** property.

C#

```
StorageFolder picturesFolder = KnownFolders.PicturesLibrary;
muxc.TreeViewNode pictureNode = new muxc.TreeViewNode();
pictureNode.Content = picturesFolder;
```

> ⓘ Note
>
> To get access to the **Pictures** folder, you need to specify the **Pictures Library** capability in the app manifest. See App capability declarations for more information.

You can provide a DataTemplate to specify how the data item is displayed in the tree view.

> ⓘ Note
>
> In Windows 10, version 1803, you have to re-template the **TreeView** control and specify a custom **ItemTemplate** if your content is not a string. In later versions, set the **ItemTemplate** property. For more info, see TreeView.ItemTemplate.

Item container style

Whether you use **ItemsSource** or **RootNodes**, the actual element used to display each node – called the "container" – is a TreeViewItem object. You can modify **TreeViewItem** properties to style the container using the **TreeView**'s ItemContainerStyle or ItemContainerStyleSelector properties.

This example shows how to change the expanded/collapsed glyphs to orange +/- signs. In the default **TreeViewItem** template, the glyphs are set to use the `Segoe MDL2 Assets` font. You can set the **Setter.Value** property by providing the Unicode character value in the format used by XAML, like this: `Value=""`.

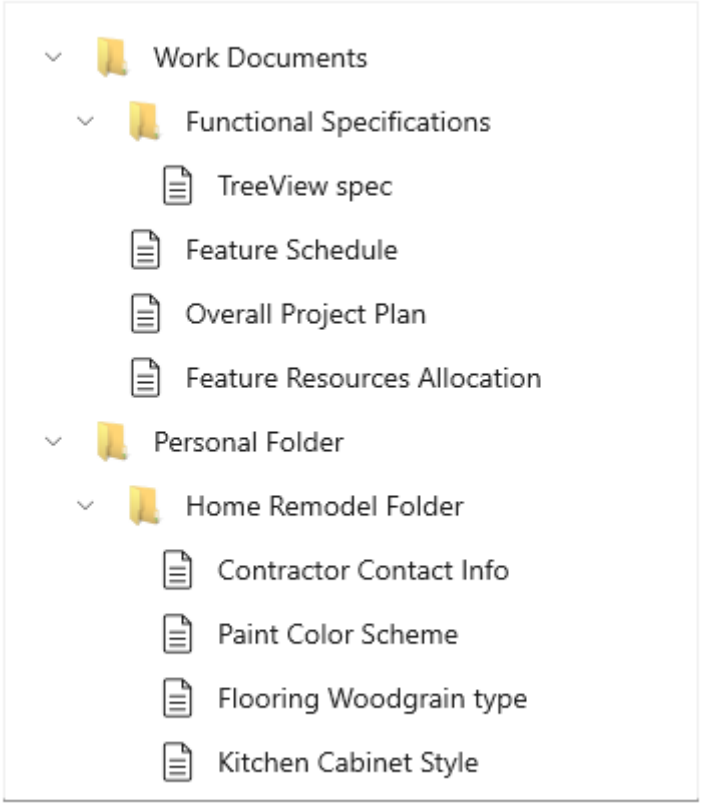
XAML

```
<muxc:TreeView>
    <muxc:TreeView.ItemContainerStyle>
        <Style TargetType="muxc:TreeViewItem">
            <Setter Property="CollapsedGlyph" Value="&#xE948;"/>
            <Setter Property="ExpandedGlyph" Value="&#xE949;"/>
            <Setter Property="GlyphBrush" Value="DarkOrange"/>
        </Style>
    </muxc:TreeView.ItemContainerStyle>
    <muxc:TreeView.RootNodes>
        <muxc:TreeViewNode Content="Flavors"
               IsExpanded="True">
            <muxc:TreeViewNode.Children>
                <muxc:TreeViewNode Content="Vanilla"/>
                <muxc:TreeViewNode Content="Strawberry"/>
                <muxc:TreeViewNode Content="Chocolate"/>
            </muxc:TreeViewNode.Children>
        </muxc:TreeViewNode>
    </muxc:TreeView.RootNodes>
</muxc:TreeView>
```

Item template selectors

By default, the TreeView shows the string representation of the data item for each node. You can set the ItemTemplate property to change what is displayed for all nodes. Or, you can use an ItemTemplateSelector to choose a different DataTemplate for the tree view items based on the type of item or some other criteria you specify.

For example, in a file explorer app, you could use one data template for folders, and another for files.



Here is an example of how to create and use an item template selector. For more info, see the DataTemplateSelector class.

> **Note**
>
> This code is part of a larger example and won't work on its own. To see the full example, including the code that defines `ExplorerItem`, check out the Xaml-Controls-Gallery repository on GitHub. TreeViewPage.xaml and TreeViewPage.xaml.cs contain the relevant code.

XAML

```
<Page.Resources>
    <DataTemplate x:Key="FolderTemplate" x:DataType="local:ExplorerItem">
        <muxc:TreeViewItem ItemsSource="{x:Bind Children}">
            <StackPanel Orientation="Horizontal">
                <Image Width="20" Source="Assets/folder.png"/>
                <TextBlock Text="{x:Bind Name}" />
            </StackPanel>
        </muxc:TreeViewItem>
    </DataTemplate>

    <DataTemplate x:Key="FileTemplate" x:DataType="local:ExplorerItem">
        <muxc:TreeViewItem>
            <StackPanel Orientation="Horizontal">
```

```
                <Image Width="20" Source="Assets/file.png"/>
                <TextBlock Text="{x:Bind Name}"/>
            </StackPanel>
        </muxc:TreeViewItem>
    </DataTemplate>

    <local:ExplorerItemTemplateSelector
            x:Key="ExplorerItemTemplateSelector"
            FolderTemplate="{StaticResource FolderTemplate}"
            FileTemplate="{StaticResource FileTemplate}" />
</Page.Resources>

<Grid>
    <muxc:TreeView
        ItemsSource="{x:Bind DataSource}"
        ItemTemplateSelector="{StaticResource
ExplorerItemTemplateSelector}"/>
</Grid>
```

C#

```
public class ExplorerItemTemplateSelector : DataTemplateSelector
{
    public DataTemplate FolderTemplate { get; set; }
    public DataTemplate FileTemplate { get; set; }

    protected override DataTemplate SelectTemplateCore(object item)
    {
        var explorerItem = (ExplorerItem)item;
        if (explorerItem.Type == ExplorerItem.ExplorerItemType.Folder)
return FolderTemplate;

        return FileTemplate;
    }
}
```

The type of object passed to the SelectTemplateCore method depends on whether you create the tree view by setting the **ItemsSource** property, or by creating and managing **TreeViewNode** objects yourself.

- If **ItemsSource** is set, the object will be of whatever type the data item is. In the previous example, the object was an `ExplorerItem`, so it could be used after a simple cast to `ExplorerItem`: `var explorerItem = (ExplorerItem)item;`.
- If **ItemsSource** is not set and you manage the tree view nodes yourself, the object passed to **SelectTemplateCore** is a **TreeViewNode**. In this case, you can get the data item from the TreeViewNode.Content property.

Here's a data template selector from the Pictures and Music library tree view example shown later. The **SelectTemplateCore** method receives a **TreeViewNode**, which might

have either a StorageFolder or a StorageFile as its content. Based on the content, you can return a default template, or a specific template for the music folder, the picture folder, music files, or picture files.

C#

```
protected override DataTemplate SelectTemplateCore(object item)
{
    var node = (TreeViewNode)item;
    if (node.Content is StorageFolder)
    {
        var content = node.Content as StorageFolder;
        if (content.DisplayName.StartsWith("Pictures")) return
PictureFolderTemplate;
        if (content.DisplayName.StartsWith("Music")) return
MusicFolderTemplate;
    }
    else if (node.Content is StorageFile)
    {
        var content = node.Content as StorageFile;
        if (content.ContentType.StartsWith("image")) return
PictureItemTemplate;
        if (content.ContentType.StartsWith("audio")) return
MusicItemTemplate;
    }
    return DefaultTemplate;
}
```

Interacting with a tree view

You can configure a tree view to let a user interact with it in several different ways:

- Expand or collapse nodes
- Single- or multi-select items
- Click to invoke an item

Expand/collapse

Any tree view node that has children can always be expanded or collapsed by clicking the expand/collapse glyph. You can also expand or collapse a node programmatically, and respond when a node changes state.

Expand/collapse a node programmatically

There are 2 ways you can expand or collapse a tree view node in your code.

- The TreeView class has the Collapse and Expand methods. When you call these methods, you pass in the **TreeViewNode** that you want to expand or collapse.
- Each TreeViewNode has the IsExpanded property. You can use this property to check the state of a node, or set it to change the state. You can also set this property in XAML to set the initial state of a node.

Fill a node when it's expanding

You might need to show a large number of nodes in your tree view, or you might not know ahead of time how many nodes it will have. The **TreeView** control is not virtualized, so you can manage resources by filling each node as it's expanded, and removing the child nodes when it's collapsed.

Handle the Expanding event and use the HasUnrealizedChildren property to add children to a node when it's being expanded. The **HasUnrealizedChildren** property indicates whether the node needs to be filled, or if its **Children** collection has already been populated. It's important to remember that the **TreeViewNode** doesn't set this value, you need to manage it in your app code.

Here's an example of these APIs in use. See the complete example code at the end of this article for context, including the implementation of **FillTreeNode**.

C#

```
private void SampleTreeView_Expanding(muxc.TreeView sender,
muxc.TreeViewExpandingEventArgs args)
{
    if (args.Node.HasUnrealizedChildren)
    {
        FillTreeNode(args.Node);
    }
}
```

It's not required, but you might want to also handle the Collapsed event and remove the child nodes when the parent node is closed. This can be important if your tree view has many nodes, or if the node data uses a lot of resources. You should consider the performance impact of filling a node each time it's opened versus leaving the children on a closed node. The best option will depend on your app.

Here's an example of a handler for the **Collapsed** event.

C#

```
private void SampleTreeView_Collapsed(muxc.TreeView sender,
muxc.TreeViewCollapsedEventArgs args)
{
    args.Node.Children.Clear();
    args.Node.HasUnrealizedChildren = true;
}
```

Invoking an item

A user can invoke an action (treating the item like a button) instead of selecting the item. You handle the ItemInvoked event to respond to this user interaction.

> ⓘ **Note**
>
> Unlike **ListView**, which has the IsItemClickEnabled property, invoking an item is always enabled on the tree view. You can still choose whether to handle the event or not.

TreeViewItemInvokedEventArgs class

The **ItemInvoked** event args give you access to the invoked item. The InvokedItem property has the node that was invoked. You can cast it to a **TreeViewNode** and get the data item from the **TreeViewNode.Content** property.

Here's an example of an **ItemInvoked** event handler. The data item is an IStorageItem, and this example just displays some info about the file and tree. Also, if the node is a folder node, it expands or collapses the node at the same time. Otherwise, the node expands or collapses only when the chevron is clicked.

C#

```
private void SampleTreeView_ItemInvoked(muxc.TreeView sender,
muxc.TreeViewItemInvokedEventArgs args)
{
    var node = args.InvokedItem as muxc.TreeViewNode;
    if (node.Content is IStorageItem item)
    {
        FileNameTextBlock.Text = item.Name;
        FilePathTextBlock.Text = item.Path;
        TreeDepthTextBlock.Text = node.Depth.ToString();

        if (node.Content is StorageFolder)
        {
            node.IsExpanded = !node.IsExpanded;
        }
```

```
    }
}
```

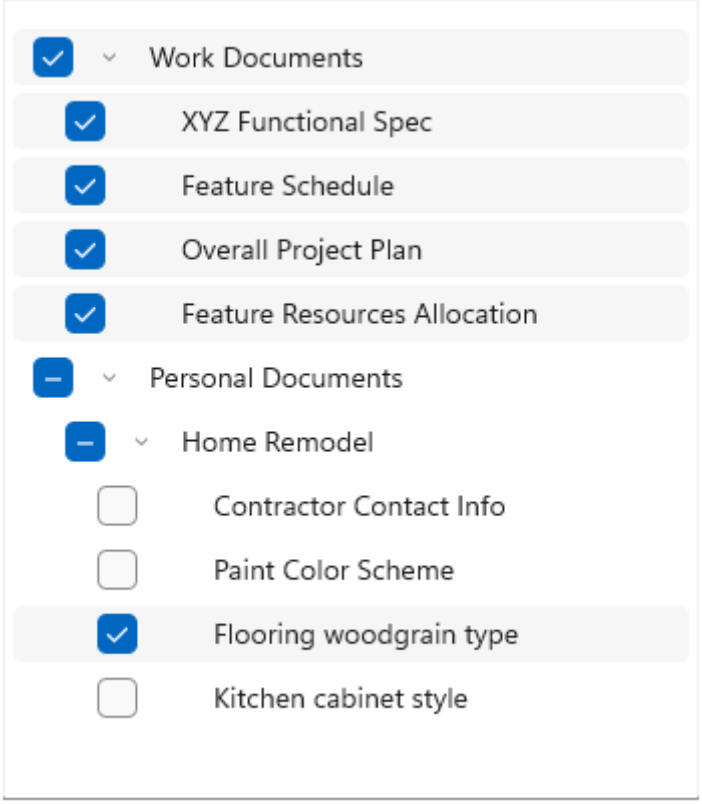
Item selection

The **TreeView** control supports both single-selection and multi-selection. By default, selection of nodes is turned off, but you can set the TreeView.SelectionMode property to allow selection of nodes. The TreeViewSelectionMode values are **None**, **Single**, and **Multiple**.

Multiple selection

When multiple selection is enabled, a checkbox is shown next to each tree view node, and selected items are highlighted. A user can select or de-select an item by using the checkbox; clicking the item still causes it to be invoked.

Selecting or de-selecting a parent node will select or de-select all children under that node. If some, but not all, of the children under a parent node are selected, the checkbox for the parent node is shown in the indeterminate state.



Selected nodes are added to the tree view's SelectedNodes collection. You can call the SelectAll method to select all the nodes in a tree view.

> **Note**
>
> If you call **SelectAll**, all realized nodes are selected, regardless of the **SelectionMode**. To provide a consistent user experience, you should only call **SelectAll** if **SelectionMode** is **Multiple**.

Selection and realized/unrealized nodes

If your tree view has unrealized nodes, they are not taken into account for selection. Here are some things you need to keep in mind regarding selection with unrealized nodes.

- If a user selects a parent node, all the realized children under that parent are also selected. Similarly, if all the child nodes are selected, the parent node also becomes selected.
- The **SelectAll** method only adds realized nodes to the **SelectedNodes** collection.
- If a parent node with unrealized children is selected, the children will be selected as they are realized.

SelectedItem/SelectedItems

TreeView has the SelectedItem and SelectedItems properties. You can use these properties to get the content of selected nodes directly. If multiple selection is enabled, SelectedItem contains the first item in the SelectedItems collection.

SelectionChanged

You can handle the SelectionChanged event to respond when the collection of selected items changes, either programmatically or through user interaction.

XAML

```
<TreeView ItemsSource="{x:Bind Folders}"
          SelectionMode="Multiple"
          SelectionChanged="TreeView_SelectionChanged"/>
```

C#

```
public void TreeView_SelectionChanged(TreeView sender,
TreeViewSelectionChangedEventArgs args)
{
    foreach (object item in args.RemovedItems)
```

```
    {
        this.SelectedFolders.Remove((Folder)item);
    }

    foreach (object item in args.AddedItems)
    {
        this.SelectedFolders.Add((Folder)item);
    }
}
```

Code examples

The following code examples demonstrate various features of the tree view control.

Tree view using XAML

This example shows how to create a simple tree view structure in XAML. The tree view shows ice cream flavors and toppings that the user can choose from, arranged in categories. Multi-selection is enabled, and when the user clicks a button, the selected items are displayed in the main app UI.

XAML

```
<Page
    x:Class="TreeViewTest.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:muxc="using:Microsoft.UI.Xaml.Controls"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
          Padding="100">
        <SplitView IsPaneOpen="True"
               DisplayMode="Inline"
               OpenPaneLength="296">
            <SplitView.Pane>
                <muxc:TreeView x:Name="DessertTree"
SelectionMode="Multiple">
                    <muxc:TreeView.RootNodes>
                        <muxc:TreeViewNode Content="Flavors"
IsExpanded="True">
                            <muxc:TreeViewNode.Children>
                                <muxc:TreeViewNode Content="Vanilla"/>
                                <muxc:TreeViewNode Content="Strawberry"/>
                                <muxc:TreeViewNode Content="Chocolate"/>
                            </muxc:TreeViewNode.Children>
```

```
                        </muxc:TreeViewNode>

                        <muxc:TreeViewNode Content="Toppings">
                            <muxc:TreeViewNode.Children>
                                <muxc:TreeViewNode Content="Candy">
                                    <muxc:TreeViewNode.Children>
                                        <muxc:TreeViewNode
Content="Chocolate"/>
                                        <muxc:TreeViewNode Content="Mint"/>
                                        <muxc:TreeViewNode
Content="Sprinkles"/>
                                    </muxc:TreeViewNode.Children>
                                </muxc:TreeViewNode>
                                <muxc:TreeViewNode Content="Fruits">
                                    <muxc:TreeViewNode.Children>
                                        <muxc:TreeViewNode Content="Mango"/>
                                        <muxc:TreeViewNode Content="Peach"/>
                                        <muxc:TreeViewNode Content="Kiwi"/>
                                    </muxc:TreeViewNode.Children>
                                </muxc:TreeViewNode>
                                <muxc:TreeViewNode Content="Berries">
                                    <muxc:TreeViewNode.Children>
                                        <muxc:TreeViewNode
Content="Strawberry"/>
                                        <muxc:TreeViewNode
Content="Blueberry"/>
                                        <muxc:TreeViewNode
Content="Blackberry"/>
                                    </muxc:TreeViewNode.Children>
                                </muxc:TreeViewNode>
                            </muxc:TreeViewNode.Children>
                        </muxc:TreeViewNode>
                    </muxc:TreeView.RootNodes>
                </muxc:TreeView>
            </SplitView.Pane>

            <StackPanel Grid.Column="1" Margin="12,0">
                <Button Content="Select all" Click="SelectAllButton_Click"/>
                <Button Content="Create order" Click="OrderButton_Click"
Margin="0,12"/>
                <TextBlock Text="Your flavor selections:" Style="
{StaticResource CaptionTextBlockStyle}"/>
                <TextBlock x:Name="FlavorList" Margin="0,0,0,12"/>
                <TextBlock Text="Your topping selections:" Style="
{StaticResource CaptionTextBlockStyle}"/>
                <TextBlock x:Name="ToppingList"/>
            </StackPanel>
        </SplitView>
    </Grid>
</Page>
```

C#

```
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using muxc = Microsoft.UI.Xaml.Controls;

namespace TreeViewTest
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
        }

        private void OrderButton_Click(object sender, RoutedEventArgs e)
        {
            FlavorList.Text = string.Empty;
            ToppingList.Text = string.Empty;

            foreach (muxc.TreeViewNode node in DessertTree.SelectedNodes)
            {
                if (node.Parent.Content?.ToString() == "Flavors")
                {
                    FlavorList.Text += node.Content + "; ";
                }
                else if (node.HasChildren == false)
                {
                    ToppingList.Text += node.Content + "; ";
                }
            }
        }

        private void SelectAllButton_Click(object sender, RoutedEventArgs e)
        {
            if (DessertTree.SelectionMode ==
muxc.TreeViewSelectionMode.Multiple)
            {
                DessertTree.SelectAll();
            }
        }
    }
}
```

Tree view using data binding

This example shows how to create the same tree view as the previous example. However, instead of creating the data hierarchy in XAML, the data is created in code and bound to the tree view's **ItemsSource** property. (The button event handlers shown in the previous example apply to this example also.)

XAML

```
<Page
    x:Class="TreeViewTest.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:muxc="using:Microsoft.UI.Xaml.Controls"
    xmlns:local="using:TreeViewTest"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
          Padding="100">
        <SplitView IsPaneOpen="True"
                   DisplayMode="Inline"
                   OpenPaneLength="296">
            <SplitView.Pane>
                <muxc:TreeView Name="DessertTree"
                                SelectionMode="Multiple"
                                ItemsSource="{x:Bind DataSource}">
                    <muxc:TreeView.ItemTemplate>
                        <DataTemplate x:DataType="local:Item">
                            <muxc:TreeViewItem
                                ItemsSource="{x:Bind Children}"
                                Content="{x:Bind Name}"/>
                        </DataTemplate>
                    </muxc:TreeView.ItemTemplate>
                </muxc:TreeView>
            </SplitView.Pane>

            <StackPanel Grid.Column="1" Margin="12,0">
                <Button Content="Select all"
                        Click="SelectAllButton_Click"/>
                <Button Content="Create order"
                        Click="OrderButton_Click"
                        Margin="0,12"/>
                <TextBlock Text="Your flavor selections:"
                           Style="{StaticResource CaptionTextBlockStyle}"/>
                <TextBlock x:Name="FlavorList" Margin="0,0,0,12"/>
                <TextBlock Text="Your topping selections:"
                           Style="{StaticResource CaptionTextBlockStyle}"/>
                <TextBlock x:Name="ToppingList"/>
            </StackPanel>
        </SplitView>
    </Grid>

</Page>
```

C#

```
using System.Collections.ObjectModel;
using Windows.UI.Xaml;
```

```
using Windows.UI.Xaml.Controls;
using muxc = Microsoft.UI.Xaml.Controls;

namespace TreeViewTest
{
    public sealed partial class MainPage : Page
    {
        private ObservableCollection<Item> DataSource = new
ObservableCollection<Item>();

        public MainPage()
        {
            this.InitializeComponent();
            DataSource = GetDessertData();
        }

        private ObservableCollection<Item> GetDessertData()
        {
            var list = new ObservableCollection<Item>();

            Item flavorsCategory = new Item()
            {
                Name = "Flavors",
                Children =
                {
                    new Item() { Name = "Vanilla" },
                    new Item() { Name = "Strawberry" },
                    new Item() { Name = "Chocolate" }
                }
            };

            Item toppingsCategory = new Item()
            {
                Name = "Toppings",
                Children =
                {
                    new Item()
                    {
                        Name = "Candy",
                        Children =
                        {
                            new Item() { Name = "Chocolate" },
                            new Item() { Name = "Mint" },
                            new Item() { Name = "Sprinkles" }
                        }
                    },
                    new Item()
                    {
                        Name = "Fruits",
                        Children =
                        {
                            new Item() { Name = "Mango" },
                            new Item() { Name = "Peach" },
                            new Item() { Name = "Kiwi" }
                        }
```

```
                    },
                    new Item()
                    {
                        Name = "Berries",
                        Children =
                        {
                            new Item() { Name = "Strawberry" },
                            new Item() { Name = "Blueberry" },
                            new Item() { Name = "Blackberry" }
                        }
                    }
                }
            };

            list.Add(flavorsCategory);
            list.Add(toppingsCategory);
            return list;
        }

        private void OrderButton_Click(object sender, RoutedEventArgs e)
        {
            FlavorList.Text = string.Empty;
            ToppingList.Text = string.Empty;

            foreach (muxc.TreeViewNode node in DessertTree.SelectedNodes)
            {
                if (node.Parent.Content?.ToString() == "Flavors")
                {
                    FlavorList.Text += node.Content + "; ";
                }
                else if (node.HasChildren == false)
                {
                    ToppingList.Text += node.Content + "; ";
                }
            }
        }

        private void SelectAllButton_Click(object sender, RoutedEventArgs e)
        {
            if (DessertTree.SelectionMode ==
muxc.TreeViewSelectionMode.Multiple)
            {
                DessertTree.SelectAll();
            }
        }
    }

    public class Item
    {
        public string Name { get; set; }
        public ObservableCollection<Item> Children { get; set; } = new
ObservableCollection<Item>();

        public override string ToString()
        {
```

```
            return Name;
        }
    }
}
```

Pictures and Music library tree view

This example shows how to create a tree view that shows the contents and structure of the user's **Pictures** and **Music** libraries. The number of items can't be known ahead of time, so each node is filled when it's expanded, and emptied when it's collapsed.

A custom item template is used to display the data items, which are of type IStorageItem.

> **ⓘ Important**
>
> The code in this example requires the **picturesLibrary** and **musicLibrary** capabilities. For more info about file access, see File access permissions, Enumerate and query files and folders, and Files and folders in the Music, Pictures, and Videos libraries.

XAML

```
<Page
    x:Class="TreeViewTest.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:TreeViewTest"
    xmlns:muxc="using:Microsoft.UI.Xaml.Controls"
    xmlns:storage="using:Windows.Storage"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">
    <Page.Resources>
        <DataTemplate x:Key="TreeViewItemDataTemplate"
x:DataType="muxc:TreeViewNode">
            <Grid Height="44">
                <TextBlock Text="{x:Bind
((storage:IStorageItem)Content).Name}"
                           HorizontalAlignment="Left"
                           VerticalAlignment="Center"
                           Style="{ThemeResource BodyTextBlockStyle}"/>
            </Grid>
        </DataTemplate>

        <DataTemplate x:Key="MusicItemDataTemplate"
```

```
x:DataType="muxc:TreeViewNode">
            <StackPanel Height="44" Orientation="Horizontal">
                <SymbolIcon Symbol="Audio" Margin="0,0,4,0"/>
                <TextBlock Text="{x:Bind
((storage:StorageFile)Content).DisplayName}"
                           HorizontalAlignment="Left"
                           VerticalAlignment="Center"
                           Style="{ThemeResource BodyTextBlockStyle}"/>
            </StackPanel>
        </DataTemplate>

        <DataTemplate x:Key="PictureItemDataTemplate"
x:DataType="muxc:TreeViewNode">
            <StackPanel Height="44" Orientation="Horizontal">
                <FontIcon FontFamily="Segoe MDL2 Assets" Glyph="&#xEB9F;"
                          Margin="0,0,4,0"/>
                <TextBlock Text="{x:Bind
((storage:StorageFile)Content).DisplayName}"
                           HorizontalAlignment="Left"
                           VerticalAlignment="Center"
                           Style="{ThemeResource BodyTextBlockStyle}"/>
            </StackPanel>
        </DataTemplate>

        <DataTemplate x:Key="MusicFolderDataTemplate"
x:DataType="muxc:TreeViewNode">
            <StackPanel Height="44" Orientation="Horizontal">
                <SymbolIcon Symbol="MusicInfo" Margin="0,0,4,0"/>
                <TextBlock Text="{x:Bind
((storage:StorageFolder)Content).DisplayName}"
                           HorizontalAlignment="Left"
                           VerticalAlignment="Center"
                           Style="{ThemeResource BodyTextBlockStyle}"/>
            </StackPanel>
        </DataTemplate>

        <DataTemplate x:Key="PictureFolderDataTemplate"
x:DataType="muxc:TreeViewNode">
            <StackPanel Height="44" Orientation="Horizontal">
                <SymbolIcon Symbol="Pictures" Margin="0,0,4,0"/>
                <TextBlock Text="{x:Bind
((storage:StorageFolder)Content).DisplayName}"
                           HorizontalAlignment="Left"
                           VerticalAlignment="Center"
                           Style="{ThemeResource BodyTextBlockStyle}"/>
            </StackPanel>
        </DataTemplate>

        <local:ExplorerItemTemplateSelector
            x:Key="ExplorerItemTemplateSelector"
            DefaultTemplate="{StaticResource TreeViewItemDataTemplate}"
            MusicItemTemplate="{StaticResource MusicItemDataTemplate}"
            MusicFolderTemplate="{StaticResource MusicFolderDataTemplate}"
            PictureItemTemplate="{StaticResource PictureItemDataTemplate}"
            PictureFolderTemplate="{StaticResource
```

```
PictureFolderDataTemplate}"/>
    </Page.Resources>

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <SplitView IsPaneOpen="True"
                   DisplayMode="Inline"
                   OpenPaneLength="296">
            <SplitView.Pane>
                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition/>
                    </Grid.RowDefinitions>
                    <Button Content="Refresh tree"
Click="RefreshButton_Click" Margin="24,12"/>
                    <muxc:TreeView x:Name="sampleTreeView" Grid.Row="1"
SelectionMode="Single"
                                   ItemTemplateSelector="{StaticResource
ExplorerItemTemplateSelector}"
                                   Expanding="SampleTreeView_Expanding"
                                   Collapsed="SampleTreeView_Collapsed"
                                   ItemInvoked="SampleTreeView_ItemInvoked"/>
                </Grid>
            </SplitView.Pane>

            <StackPanel Grid.Column="1" Margin="12,72">
                <TextBlock Text="File name:" Style="{StaticResource
CaptionTextBlockStyle}"/>
                <TextBlock x:Name="FileNameTextBlock" Margin="0,0,0,12"/>

                <TextBlock Text="File path:" Style="{StaticResource
CaptionTextBlockStyle}"/>
                <TextBlock x:Name="FilePathTextBlock" Margin="0,0,0,12"/>

                <TextBlock Text="Tree depth:" Style="{StaticResource
CaptionTextBlockStyle}"/>
                <TextBlock x:Name="TreeDepthTextBlock" Margin="0,0,0,12"/>
            </StackPanel>
        </SplitView>
    </Grid>
</Page>
```

C#

```
using System;
using System.Collections.Generic;
using Windows.Storage;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using muxc = Microsoft.UI.Xaml.Controls;

namespace TreeViewTest
{
```

```
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            InitializeTreeView();
        }

        private void InitializeTreeView()
        {
            // A TreeView can have more than 1 root node. The Pictures
library
            // and the Music library will each be a root node in the tree.
            // Get Pictures library.
            StorageFolder picturesFolder = KnownFolders.PicturesLibrary;
            muxc.TreeViewNode pictureNode = new muxc.TreeViewNode();
            pictureNode.Content = picturesFolder;
            pictureNode.IsExpanded = true;
            pictureNode.HasUnrealizedChildren = true;
            sampleTreeView.RootNodes.Add(pictureNode);
            FillTreeNode(pictureNode);

            // Get Music library.
            StorageFolder musicFolder = KnownFolders.MusicLibrary;
            muxc.TreeViewNode musicNode = new muxc.TreeViewNode();
            musicNode.Content = musicFolder;
            musicNode.IsExpanded = true;
            musicNode.HasUnrealizedChildren = true;
            sampleTreeView.RootNodes.Add(musicNode);
            FillTreeNode(musicNode);
        }

        private async void FillTreeNode(muxc.TreeViewNode node)
        {
            // Get the contents of the folder represented by the current
tree node.
            // Add each item as a new child node of the node that's being
expanded.

            // Only process the node if it's a folder and has unrealized
children.
            StorageFolder folder = null;

            if (node.Content is StorageFolder && node.HasUnrealizedChildren
== true)
            {
                folder = node.Content as StorageFolder;
            }
            else
            {
                // The node isn't a folder, or it's already been filled.
                return;
            }

            IReadOnlyList<IStorageItem> itemsList = await
```

```
folder.GetItemsAsync();

            if (itemsList.Count == 0)
            {
                // The item is a folder, but it's empty. Leave
HasUnrealizedChildren = true so
                // that the chevron appears, but don't try to process
children that aren't there.
                return;
            }

            foreach (var item in itemsList)
            {
                var newNode = new muxc.TreeViewNode();
                newNode.Content = item;

                if (item is StorageFolder)
                {
                    // If the item is a folder, set HasUnrealizedChildren to
true.
                    // This makes the collapsed chevron show up.
                    newNode.HasUnrealizedChildren = true;
                }
                else
                {
                    // Item is StorageFile. No processing needed for this
scenario.
                }

                node.Children.Add(newNode);
            }

            // Children were just added to this node, so set
HasUnrealizedChildren to false.
            node.HasUnrealizedChildren = false;
        }

        private void SampleTreeView_Expanding(muxc.TreeView sender,
muxc.TreeViewExpandingEventArgs args)
        {
            if (args.Node.HasUnrealizedChildren)
            {
                FillTreeNode(args.Node);
            }
        }

        private void SampleTreeView_Collapsed(muxc.TreeView sender,
muxc.TreeViewCollapsedEventArgs args)
        {
            args.Node.Children.Clear();
            args.Node.HasUnrealizedChildren = true;
        }

        private void SampleTreeView_ItemInvoked(muxc.TreeView sender,
muxc.TreeViewItemInvokedEventArgs args)
```

```
        {
            var node = args.InvokedItem as muxc.TreeViewNode;

            if (node.Content is IStorageItem item)
            {
                FileNameTextBlock.Text = item.Name;
                FilePathTextBlock.Text = item.Path;
                TreeDepthTextBlock.Text = node.Depth.ToString();

                if (node.Content is StorageFolder)
                {
                    node.IsExpanded = !node.IsExpanded;
                }
            }
        }

        private void RefreshButton_Click(object sender, RoutedEventArgs e)
        {
            sampleTreeView.RootNodes.Clear();
            InitializeTreeView();
        }
    }

    public class ExplorerItemTemplateSelector : DataTemplateSelector
    {
        public DataTemplate DefaultTemplate { get; set; }
        public DataTemplate MusicItemTemplate { get; set; }
        public DataTemplate PictureItemTemplate { get; set; }
        public DataTemplate MusicFolderTemplate { get; set; }
        public DataTemplate PictureFolderTemplate { get; set; }

        protected override DataTemplate SelectTemplateCore(object item)
        {
            var node = (muxc.TreeViewNode)item;

            if (node.Content is StorageFolder)
            {
                var content = node.Content as StorageFolder;
                if (content.DisplayName.StartsWith("Pictures")) return
PictureFolderTemplate;
                if (content.DisplayName.StartsWith("Music")) return
MusicFolderTemplate;
            }
            else if (node.Content is StorageFile)
            {
                var content = node.Content as StorageFile;
                if (content.ContentType.StartsWith("image")) return
PictureItemTemplate;
                if (content.ContentType.StartsWith("audio")) return
MusicItemTemplate;

            }
            return DefaultTemplate;
        }
```

```
    }
}
```

Drag and drop items between tree views

The following example demonstrates how to create two tree views whose items can be dragged and dropped between each other. When an item is dragged to the other tree view, it is added to the end of the list. However, items can be re-ordered within a tree view. This example also only takes into account tree views with one root node.

XAML

```
<Page
    x:Class="TreeViewTest.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>

        <TreeView x:Name="treeView1"
                  AllowDrop="True"
                  CanDragItems="True"
                  CanReorderItems="True"
                  DragOver="TreeView_DragOver"
                  Drop="TreeView_Drop"
                  DragItemsStarting="TreeView_DragItemsStarting"
                  DragItemsCompleted="TreeView_DragItemsCompleted"/>
        <TreeView x:Name="treeView2"
                  AllowDrop="True"
                  Grid.Column="1"
                  CanDragItems="True"
                  CanReorderItems="True"
                  DragOver="TreeView_DragOver"
                  Drop="TreeView_Drop"
                  DragItemsStarting="TreeView_DragItemsStarting"
                  DragItemsCompleted="TreeView_DragItemsCompleted"/>

    </Grid>

</Page>
```

C#

```
using System;
using Windows.ApplicationModel.DataTransfer;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

namespace TreeViewTest
{
    public sealed partial class MainPage : Page
    {
        private TreeViewNode deletedItem;
        private TreeView sourceTreeView;

        public MainPage()
        {
            this.InitializeComponent();
            InitializeTreeView();
        }

        private void InitializeTreeView()
        {
            TreeViewNode parentNode1 = new TreeViewNode() { Content = "tv1"
};
            TreeViewNode parentNode2 = new TreeViewNode() { Content = "tv2"
};

            parentNode1.Children.Add(new TreeViewNode() { Content =
"tv1FirstChild" });
            parentNode1.Children.Add(new TreeViewNode() { Content =
"tv1SecondChild" });
            parentNode1.Children.Add(new TreeViewNode() { Content =
"tv1ThirdChild" });
            parentNode1.Children.Add(new TreeViewNode() { Content =
"tv1FourthChild" });
            parentNode1.IsExpanded = true;
            treeView1.RootNodes.Add(parentNode1);

            parentNode2.Children.Add(new TreeViewNode() { Content =
"tv2FirstChild" });
            parentNode2.Children.Add(new TreeViewNode() { Content =
"tv2SecondChild" });
            parentNode2.IsExpanded = true;
            treeView2.RootNodes.Add(parentNode2);
        }

        private void TreeView_DragOver(object sender, DragEventArgs e)
        {
            if (e.DataView.Contains(StandardDataFormats.Text))
            {
                e.AcceptedOperation = DataPackageOperation.Move;
            }
        }

        private async void TreeView_Drop(object sender, DragEventArgs e)
        {
```

```
            if (e.DataView.Contains(StandardDataFormats.Text))
            {
                string text = await e.DataView.GetTextAsync();
                TreeView destinationTreeView = sender as TreeView;

                if (destinationTreeView.RootNodes != null)
                {
                    TreeViewNode newNode = new TreeViewNode() { Content =
text };
                    destinationTreeView.RootNodes[0].Children.Add(newNode);
                    deletedItem = newNode;
                }
            }
        }

        private void TreeView_DragItemsStarting(TreeView sender,
TreeViewDragItemsStartingEventArgs args)
        {
            if (args.Items.Count == 1)
            {
                args.Data.RequestedOperation = DataPackageOperation.Move;
                sourceTreeView = sender;

                foreach (var item in args.Items)
                {
                    args.Data.SetText(item.ToString());
                }
            }
        }

        private void TreeView_DragItemsCompleted(TreeView sender,
TreeViewDragItemsCompletedEventArgs args)
        {
            var children = sourceTreeView.RootNodes[0].Children;

            if (deletedItem != null)
            {
                for (int i = 0; i < children.Count; i++)
                {
                    if (children[i].Content.ToString() ==
deletedItem.Content.ToString())
                    {
                        children.RemoveAt(i);
                        break;
                    }
                }
            }

            sourceTreeView = null;
            deletedItem = null;
        }
    }
}
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The TreeView for UWP apps is included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in both the Windows.UI.Xaml.Controls (UWP) and Microsoft.UI.Xaml.Controls (WinUI) namespaces.

- ✓ **UWP APIs:** TreeView class, TreeViewNode class, TreeView.ItemsSource property
- ✓ **WinUI 2 Apis:** TreeView class, TreeViewNode class, TreeView.ItemsSource property
- ✓ Open the WinUI 2 Gallery app and see the TreeView in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles, templates, and features for all controls.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:TreeView>
    <muxc:TreeView.RootNodes>
        <muxc:TreeViewNode Content="Flavors">
            <muxc:TreeViewNode.Children>
                <muxc:TreeViewNode Content="Vanilla"/>
            </muxc:TreeViewNode.Children>
        </muxc:TreeViewNode>
    </muxc:TreeView.RootNodes>
</muxc:TreeView>
```

Related articles

- TreeView class
- ListView class
- ListView and GridView

ItemsRepeater

Article • 02/26/2025

Use an ItemsRepeater to create custom collection experiences using a flexible layout system, custom views, and virtualization.

Unlike ListView, ItemsRepeater does not provide a comprehensive end-user experience – it has no default UI and provides no policy around focus, selection, or user interaction. Instead, it's a building block that you can use to create your own unique collection-based experiences and custom controls. While it has no built-in policy, it enables you to attach policy to build the experience you require. For example, you can define the layout to use, the keyboarding policy, the selection policy, etc.

You can think of ItemsRepeater conceptually as a data-driven panel, rather than as a complete control like ListView. You specify a collection of data items to be displayed, an item template that generates a UI element for each data item, and a layout that determines how the elements are sized and positioned. Then, ItemsRepeater produces child elements based on the data source, and displays them as specified by the item template and layout. The items displayed do not need to be homogenous because ItemsRepeater can load content to represent the data items based on criteria you specify in a data template selector.

Is this the right control?

Use an ItemsRepeater to create custom displays for collections of data. While it can be used to present a basic set of items, you might often use it as the display element in the template of a custom control.

If you need an out-of-the-box control to display data in a list or grid with minimal customization, consider using a ListView or GridView.

ItemsRepeater does not have a built-in Items collection. If you need to provide an Items collection directly, rather than binding to a separate data source, then you're likely in need of a more high-policy experience and should use ListView or GridView.

ItemsControl and ItemsRepeater both enable customizable collection experiences, but ItemsRepeater supports virtualizing UI layouts, while ItemsControl does not. We recommend using ItemsRepeater instead of ItemsControl, whether its for just presenting a few items from data or building a custom collection control.

Scrolling with ItemsRepeater

ItemsRepeater does not derive from Control, so it doesn't have a control template. Therefore, it doesn't contain any built-in scrolling like a ListView or other collection controls do.

When you use an **ItemsRepeater**, you should provide scrolling functionality by wrapping it in a ScrollViewer control.

> **Note**
>
> If your app will run on earlier versions of Windows - those released *before* Windows 10, version 1809 - then you also need to host the **ScrollViewer** inside the ItemsRepeaterScrollHost.
>
> XAML
>
> ```
> <muxc:ItemsRepeaterScrollHost>
>     <ScrollViewer>
>         <muxc:ItemsRepeater ... />
>     </ScrollViewer>
> </muxc:ItemsRepeaterScrollHost>
> ```
>
> If your app will only run on recent versions of Windows 10, version 1809 and later - then there is no need to use the ItemsRepeaterScrollHost.
>
> Prior to Windows 10, version 1809, **ScrollViewer** did not implement the IScrollAnchorProvider interface that the **ItemsRepeater** needs. The **ItemsRepeaterScrollHost** enables the **ItemsRepeater** to coordinate with **ScrollViewer** on earlier releases to correctly preserve the visible location of items the user is viewing. Otherwise, the items might appear to move or disappear suddenly when the items in the list are changed or the app is resized.

Create an ItemsRepeater

✓ **Important APIs**: ItemsRepeater class, ScrollViewer class

Open the WinUI 3 Gallery app and see the ItemsRepeater in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

To use an ItemsRepeater, you need to give it the data to display by setting the **ItemsSource** property. Then, tell it how to display the items by setting the ItemTemplate property.

ItemsSource

To populate the view, set the ItemsSource property to a collection of data items. Here, the **ItemsSource** is set in code directly to an instance of a collection.

C#

```
ObservableCollection<string> Items = new ObservableCollection<string>();

ItemsRepeater itemsRepeater1 = new ItemsRepeater();
itemsRepeater1.ItemsSource = Items;
```

You can also bind the **ItemsSource** property to a collection in XAML. For more info about data binding, see Data binding overview.

XAML

```
<ItemsRepeater ItemsSource="{x:Bind Items}"/>
```

ItemTemplate

To specify how a data item is visualized, set the ItemTemplate property to a DataTemplate or DataTemplateSelector you have defined. The data template defines how the data is visualized. By default, the item is displayed in the view with a **TextBlock** the uses the string representation of the data object.

However, you typically want to show a more rich presentation of your data by using a template that defines the layout and appearance of one or more controls that you'll use to display an individual item. The controls you use in the template can be bound to the properties of the data object, or have static content defined inline.

DataTemplate

In this example, the data object is a simple string. The **DataTemplate** includes an image to the left of the text, and styles the **TextBlock** to display the string in a teal color.

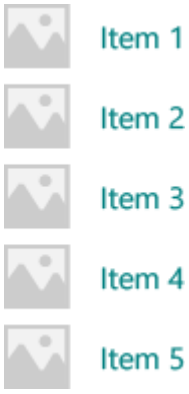
> ⓘ Note

When you use the x:Bind markup extension in a **DataTemplate**, you have to specify the DataType (`x:DataType`) on the DataTemplate.

XAML

```
<DataTemplate x:DataType="x:String">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="47"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Image Source="Assets/placeholder.png" Width="32" Height="32"
               HorizontalAlignment="Left"/>
        <TextBlock Text="{x:Bind}" Foreground="Teal"
                   FontSize="15" Grid.Column="1"/>
    </Grid>
</DataTemplate>
```

Here's how the items would appear when displayed with this **DataTemplate**.



The number of elements used in the **DataTemplate** for an item can have a significant impact on performance if your view displays a large number of items. For more info and examples of how to use **DataTemplate**s to define the look of items in your list, see Item containers and templates.

Tip

For convenience when you want to declare the template inline rather than referenced as a static resource, you can specify the **DataTemplate** or **DataTemplateSelector** as the direct child of the **ItemsRepeater**. It will be assigned as the value of the **ItemTemplate** property. For example, this is valid:

XAML

```
<ItemsRepeater ItemsSource="{x:Bind Items}">
    <DataTemplate>
```

```
        <!-- ... -->
    </DataTemplate>
</ItemsRepeater>
```

> **Tip**
>
> Unlike **ListView** and other collection controls, the **ItemsRepeater** doesn't wrap the elements from a **DataTemplate** with an additional item container that includes default policy such as margins, padding, selection visuals, or a pointer over visual state. Instead, **ItemsRepeater** only presents what is defined in the **DataTemplate**. If you want your items to have the same look as a list view item, you can explicitly include a container, like **ListViewItem**, in your data template. **ItemsRepeater** will show the **ListViewItem** visuals, but doesn't automatically make use of other functionality, like selection or showing the multi-select checkbox.
>
> Similarly, if your data collection is a collection of actual controls, like **Button** (`List<Button>`), you can put a **ContentPresenter** in your **DataTemplate** to display the control.

DataTemplateSelector

The items you display in your view do not need to be of the same type. You can provide the ItemTemplate property with a DataTemplateSelector to select different **DataTemplate**s based on criteria you specify.

This example assumes a **DataTemplateSelector** has been defined that decides between two different **DataTemplate**s to represent a Large and Small item.

XAML

```
<ItemsRepeater ...>
    <ItemsRepeater.ItemTemplate>
        <local:VariableSizeTemplateSelector Large="{StaticResource
LargeItemTemplate}"
                                            Small="{StaticResource
SmallItemTemplate}"/>
    </ItemsRepeater.ItemTemplate>
</ItemsRepeater>
```

When defining a **DataTemplateSelector** to use with **ItemsRepeater** you only need to implement an override for the SelectTemplateCore(Object) method. For more info and examples, see DataTemplateSelector.

> **Note**
>
> An alternative to **DataTemplate**s to manage how elements are created in more advanced scenarios is to implement your own **IElementFactory** to use as the **ItemTemplate**. It will be responsible for generating content when requested.

Configure the data source

Use the ItemsSource property to specify the collection to use to generate the content of items. You can set the ItemsSource to any type that implements **IEnumerable**. Additional collection interfaces implemented by your data source determine what functionality is available to the ItemsRepeater to interact with your data.

This list shows available interfaces and when to consider using each one.

- IEnumerable(.NET) / IIterable
  - Can be used for small, static data sets.

    At a minimum, the data source must implement the IEnumerable / IIterable interface. If this is all that's supported then the control will iterate through everything once to create a copy that it can use to access items via an index value.
- IReadonlyList(.NET) / IVectorView
  - Can be used for static, read-only data sets.

    Enables the control to access items by index and avoids the redundant internal copy.
- IList(.NET) / IVector
  - Can be used for static data sets.

    Enables the control to access items by index and avoids the redundant internal copy.

    Warning: Changes to the list/vector without implementing INotifyCollectionChanged won't be reflected in the UI.
- INotifyCollectionChanged(.NET)
  - Recommended to support change notification.

Enables the control to observe and react to changes in the data source and reflect those changes in the UI.

- IObservableVector
  - Supports change notification

    Like the **INotifyCollectionChanged** interface, this enables the control to observe and react to changes in the data source.

    Warning: The Windows.Foundation.IObservableVector<T> doesn't support a 'Move' action. This can cause the UI for an item to lose its visual state. For example, an item that is currently selected and/or has focus where the move is achieved by a 'Remove' followed by an 'Add' will lose focus and no longer be selected.

    The Platform.Collections.Vector<T> uses IObservableVector<T> and has this same limitation. If support for a 'Move' action is required then use the **INotifyCollectionChanged** interface. The .NET ObservableCollection<T> class uses **INotifyCollectionChanged**.

- IKeyIndexMapping
  - When a unique identifier can be associated with each item. Recommended when using 'Reset' as the collection change action.

    Enables the control to very efficiently recover the existing UI after receiving a hard 'Reset' action as part of an **INotifyCollectionChanged** or **IObservableVector** event. After receiving a reset the control will use the provided unique ID to associate the current data with elements it had already created. Without the key to index mapping the control would have to assume it needs to start over from scratch in creating UI for the data.

The interfaces listed above, other than IKeyIndexMapping, provide the same behavior in ItemsRepeater as they do in ListView and GridView.

The following interfaces on an ItemsSource enable special functionality in the ListView and GridView controls, but currently have no effect on an ItemsRepeater:

- ISupportIncrementalLoading
- IItemsRangeInfo
- ISelectionInfo

> **Tip**

We want your feedback! Let us know what you think on the **WinUI GitHub project**. Consider adding your thoughts on existing proposals such as **#374**: Add incremental loading support for ItemsRepeater.

An alternative approach to incrementally load your data as the user scrolls up or down is to observe the position of the ScrollViewer's viewport and load more data as the viewport approaches the extent.

XAML

```
<ScrollViewer ViewChanged="ScrollViewer_ViewChanged">
    <ItemsRepeater ItemsSource="{x:Bind MyItemsSource}" .../>
</ScrollViewer>
```

C#

```
private async void ScrollViewer_ViewChanged(object sender,
ScrollViewerViewChangedEventArgs e)
{
    if (!e.IsIntermediate)
    {
        var scroller = (ScrollViewer)sender;
        var distanceToEnd = scroller.ExtentHeight - (scroller.VerticalOffset
+ scroller.ViewportHeight);

        // trigger if within 2 viewports of the end
        if (distanceToEnd <= 2.0 * scroller.ViewportHeight
                && MyItemsSource.HasMore && !itemsSource.Busy)
        {
            // show an indeterminate progress UI
            myLoadingIndicator.Visibility = Visibility.Visible;

            await MyItemsSource.LoadMoreItemsAsync(/*DataFetchSize*/);

            loadingIndicator.Visibility = Visibility.Collapsed;
        }
    }
}
```

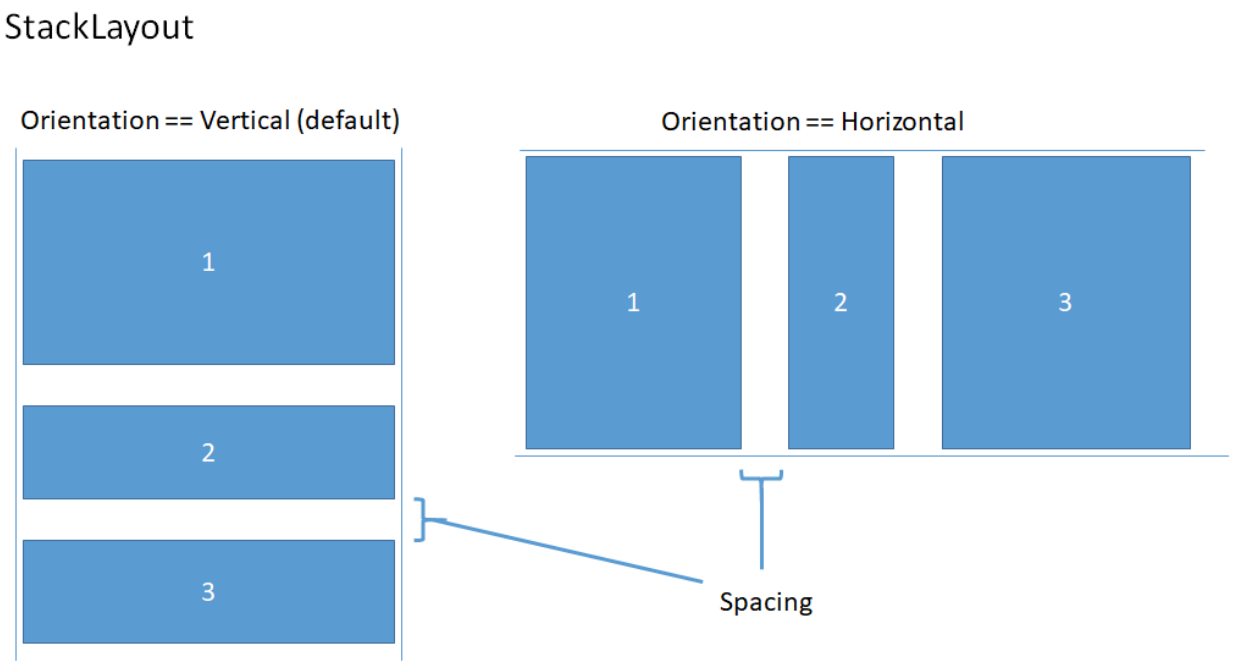
Change the layout of items

Items shown by the ItemsRepeater are arranged by a Layout object that manages the sizing and positioning of its child elements. When used with an ItemsRepeater, the Layout object enables UI virtualization. The layouts provided are StackLayout and UniformGridLayout. By default, ItemsRepeater uses a StackLayout with vertical orientation.

StackLayout

StackLayout arranges elements into a single line that you can orient horizontally or vertically.

You can set the Spacing property to adjust the amount of space between items. Spacing is applied in the direction of the layout's Orientation.



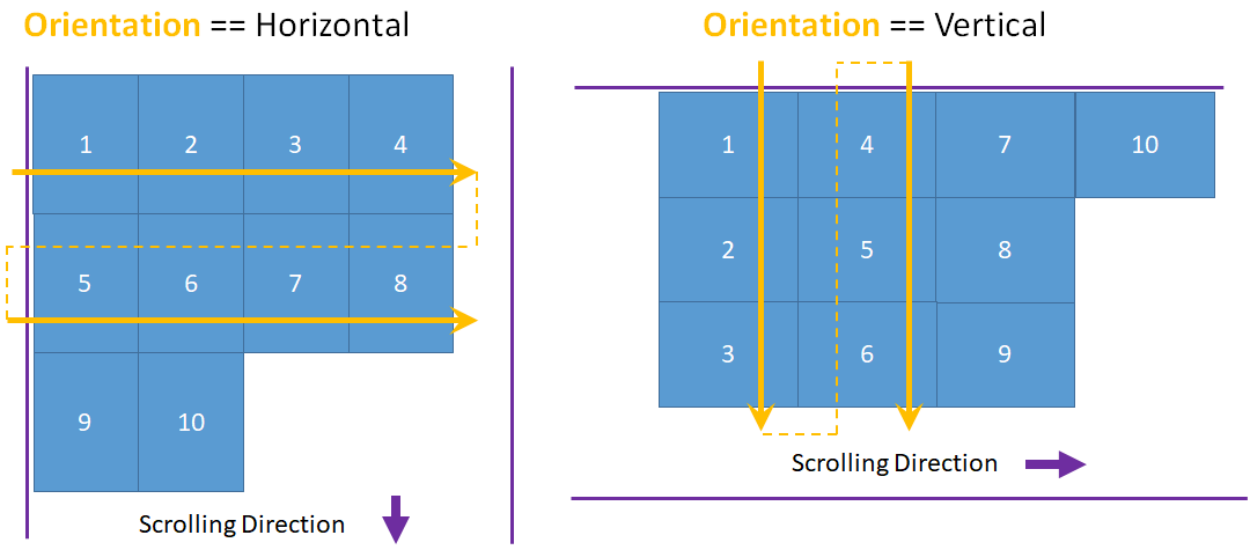
This example shows how to set the ItemsRepeater.Layout property to a StackLayout with horizontal orientation and spacing of 8 pixels.

XAML

```
<!-- xmlns:muxc="using:Microsoft.UI.Xaml.Controls" -->
<muxc:ItemsRepeater ItemsSource="{x:Bind Items}" ItemTemplate="
{StaticResource MyTemplate}">
    <muxc:ItemsRepeater.Layout>
        <muxc:StackLayout Orientation="Horizontal" Spacing="8"/>
    </muxc:ItemsRepeater.Layout>
</muxc:ItemsRepeater>
```

UniformGridLayout

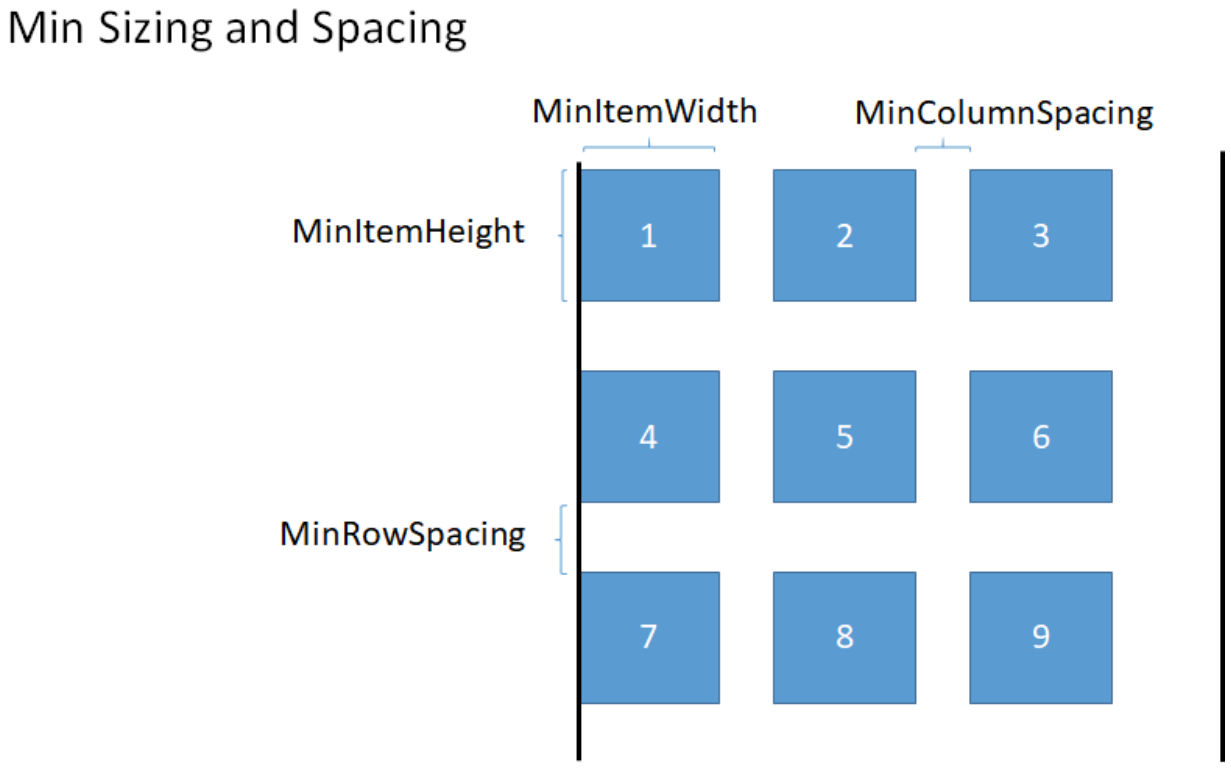
The UniformGridLayout positions elements sequentially in a wrapping layout. Items are layed out in order from left-to-right when the Orientation is **Horizontal**, and layed out top-to-bottom when the Orientation is **Vertical**. Every item is sized equally.



The number of items in each row of a horizontal layout is influenced by the minimum item width. The number of items in each column of a vertical layout is influenced by the minimum item height.

- You can explicitly provide a minimum size to use by setting the MinItemHeight and MinItemWidth properties.
- If you don't specify a minimum size, the measured size of the first item is considered the minimum size per item.

You can also set minimum spacing for the layout to include between rows and columns by setting the MinColumnSpacing and MinRowSpacing properties.



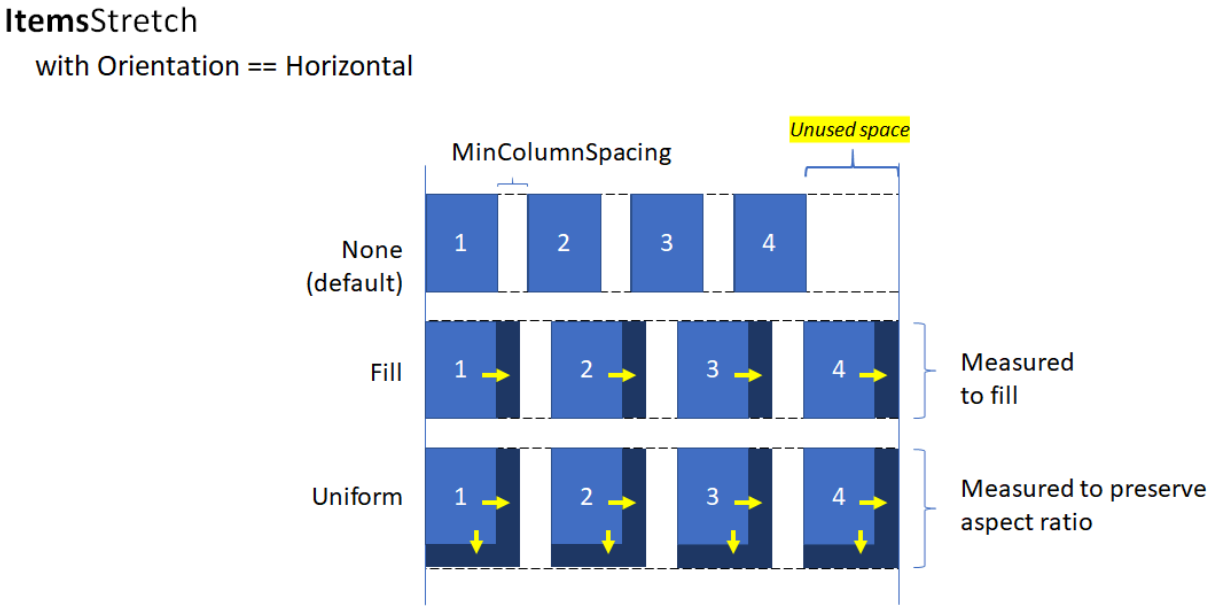
After the number of items in a row or column has been determined based on the item's minimum size and spacing, there might be unused space left after the last item in the row or column (as illustrated in the previous image). You can specify whether any extra space is ignored, used to increase the size of each item, or used to create extra space between the items. This is controlled by the ItemsStretch and ItemsJustification properties.

You can set the ItemsStretch property to specify how the item size is increased to fill the unused space.

This list shows the available values. The definitions assume the default **Orientation** of **Horizontal**.

- **None**: Extra space is left unused at the end of the row. This is the default.
- **Fill**: The items are given extra width to use up the available space (height if vertical).
- **Uniform**: The items are given extra width to use up the available space, and given extra height in order to maintain aspect ratio (height and width are switched if vertical).

This image shows the effect of the **ItemsStretch** values in a horizontal layout.



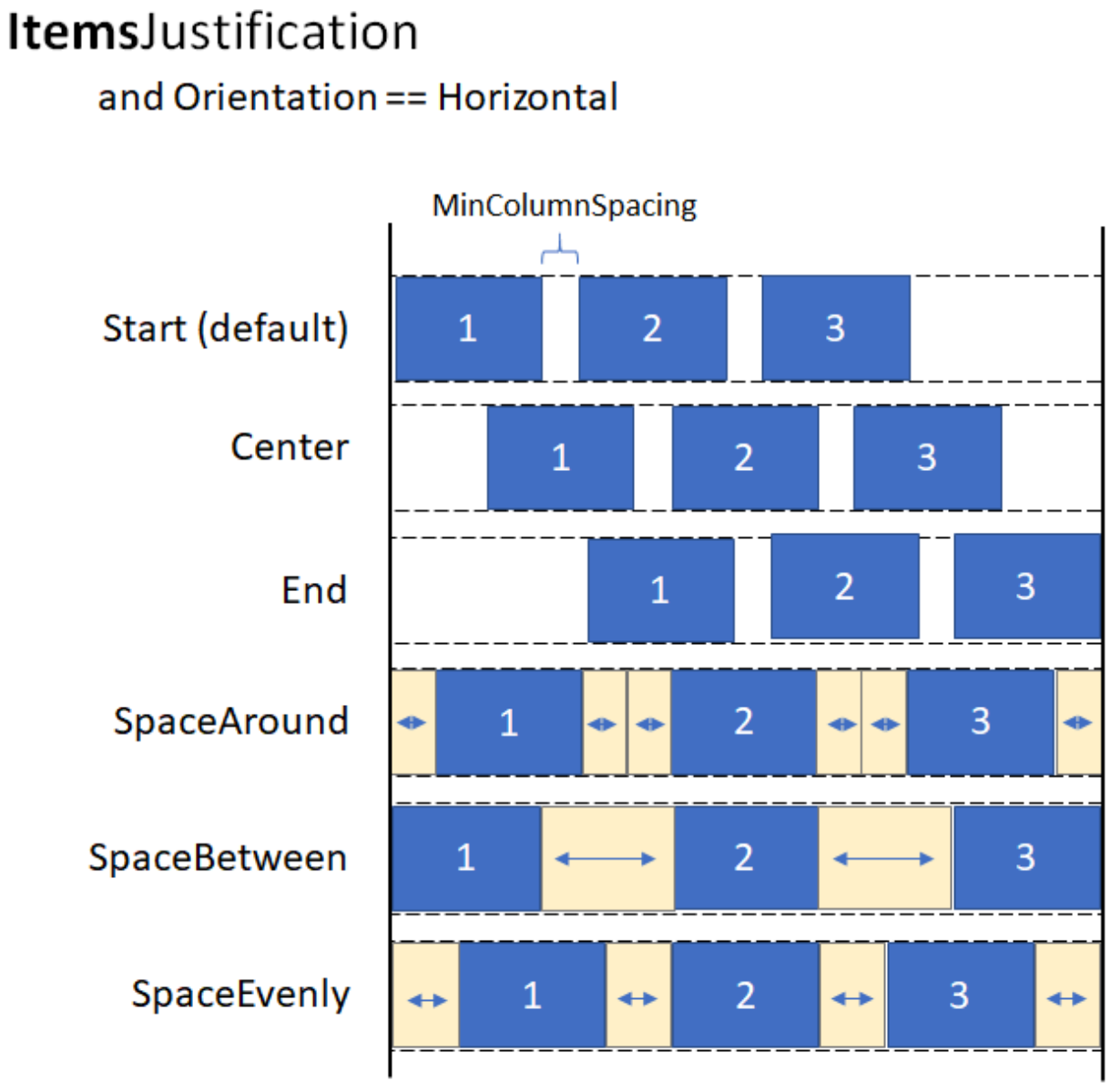
When **ItemsStretch** is **None**, you can set the ItemsJustification property to specify how extra space is used to align the items.

This list shows the available values. The definitions assume the default **Orientation** of **Horizontal**.

- **Start**: Items are aligned with the start of the row. Extra space is left unused at the end of the row. This is the default.

- **Center**: Items are aligned in the center of the row. Extra space is divided evenly at the start and end of the row.
- **End**: Items are aligned with the end of the row. Extra space is left unused at the start of the row.
- **SpaceAround**: Items are distributed evenly. An equal amount of space is added before and after each item.
- **SpaceBetween**: Items are distributed evenly. An equal amount of space is added between each item. No space is added at the start and end of the row.
- **SpaceEvenly**: Items are distributed evenly with an equal amount of space both between each item and at the start and end of the row.

This image shows the effect of the **ItemsStretch** values in a vertical layout (applied to columns instead of rows).



Tip

The **ItemsStretch** property affects the *measure* pass of layout. The **ItemsJustification** property affects the *arrange* pass of layout.

This example shows how to set the **ItemsRepeater.Layout** property to a **UniformGridLayout**.

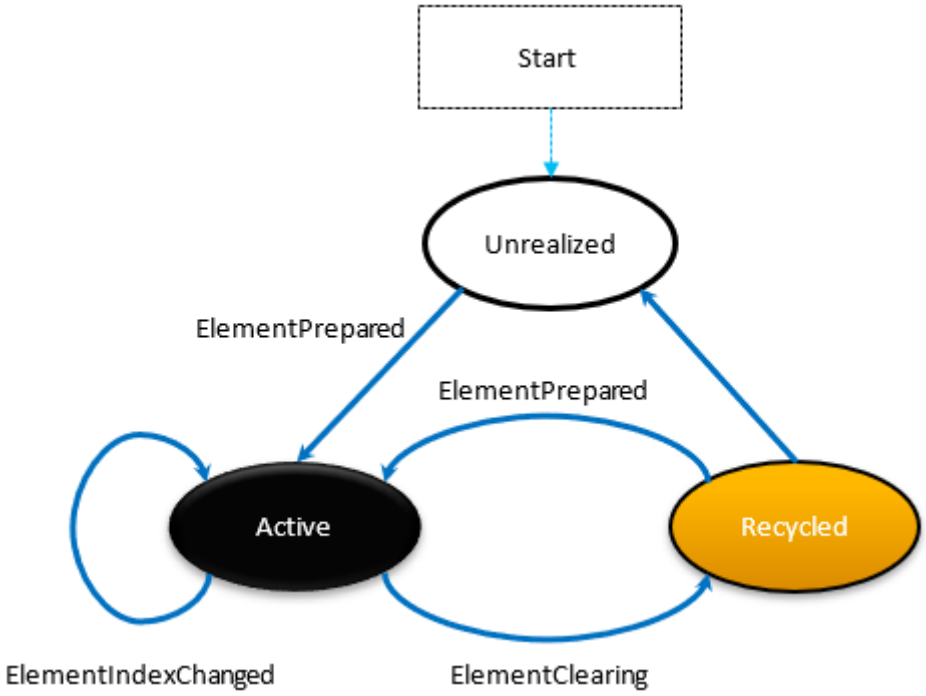
XAML

```
<!-- xmlns:muxc="using:Microsoft.UI.Xaml.Controls" -->
<muxc:ItemsRepeater ItemsSource="{x:Bind Items}"
                    ItemTemplate="{StaticResource MyTemplate}">
    <muxc:ItemsRepeater.Layout>
        <muxc:UniformGridLayout MinItemWidth="200"
                                MinColumnSpacing="28"
                                ItemsJustification="SpaceAround"/>
    </muxc:ItemsRepeater.Layout>
</muxc:ItemsRepeater>
```

Lifecycle events

When you host items in an ItemsRepeater, you might need to take some action when an item is shown or stops being shown, such as start an asynchronous download of some content, associate the element with a mechanism to track selection, or stop some background task.

In a virtualizing control, you cannot rely on Loaded/Unloaded events because the element might not be removed from the live visual tree when it's recycled. Instead, other events are provided to manage the lifecycle of elements. This diagram shows the lifecycle of an element in an ItemsRepeater, and when the related events are raised.



- **ElementPrepared** occurs each time an element is made ready for use. This occurs for both a newly created element as well as an element that already exists and is being re-used from the recycle queue.
- **ElementClearing** occurs immediately each time an element has been sent to the recycle queue, such as when it falls outside the range of realized items.
- **ElementIndexChanged** occurs for each realized UIElement where the index for the item it represents has changed. For example, when another item is added or removed in the data source, the index for items that come after in the ordering receive this event.

This example shows how you could use these events to attach a custom selection service to track item selection in a custom control that uses ItemsRepeater to display items.

XAML

```
<!-- xmlns:muxc="using:Microsoft.UI.Xaml.Controls" -->
<UserControl ...>
    ...
    <ScrollViewer>
        <muxc:ItemsRepeater ItemsSource="{x:Bind Items}"
                            ItemTemplate="{StaticResource MyTemplate}"
                            ElementPrepared="OnElementPrepared"
                            ElementIndexChanged="OnElementIndexChanged"
                            ElementClearing="OnElementClearing">
        </muxc:ItemsRepeater>
    </ScrollViewer>
    ...
</UserControl>
```

C#

```
interface ISelectable
{
    int SelectionIndex { get; set; }
    void UnregisterSelectionModel(SelectionModel selectionModel);
    void RegisterSelectionModel(SelectionModel selectionModel);
}

private void OnElementPrepared(ItemsRepeater sender,
ElementPreparedEventArgs args)
{
    var selectable = args.Element as ISelectable;
    if (selectable != null)
    {
        // Wire up this item to recognize a 'select' and listen for
programmatic
        // changes to the selection model to know when to update its visual
state.
        selectable.SelectionIndex = args.Index;
        selectable.RegisterSelectionModel(this.SelectionModel);
    }
}

private void OnElementIndexChanged(ItemsRepeater sender,
ElementIndexChangedEventArgs args)
{
    var selectable = args.Element as ISelectable;
    if (selectable != null)
    {
        // Sync the ID we use to notify the selection model when the item
        // we represent has changed location in the data source.
        selectable.SelectionIndex = args.NewIndex;
    }
}

private void OnElementClearing(ItemsRepeater sender,
ElementClearingEventArgs args)
{
    var selectable = args.Element as ISelectable;
    if (selectable != null)
    {
        // Disconnect handlers to recognize a 'select' and stop
        // listening for programmatic changes to the selection model.
        selectable.UnregisterSelectionModel(this.SelectionModel);
        selectable.SelectionIndex = -1;
    }
}
```

Sorting, Filtering and Resetting the Data

When you perform actions such as filtering or sorting your data set, you traditionally may have compared the previous set of data to the new data, then issued granular change notifications via INotifyCollectionChanged. However, it is often easier to completely replace the old data with the new data and trigger a collection change notification using the Reset action instead.

Typically, a reset causes a control to release existing child elements and start over, building the UI from the beginning at scroll position 0 as it has no awareness of exactly how the data has changed during the reset.

However, if the collection assigned as the ItemsSource supports unique identifiers by implementing the IKeyIndexMapping interface, then the ItemsRepeater can quickly identify:

- reusable UIElements for data that existed both before and after the reset
- previously visible items that were removed
- new items added that will be visible

This lets the ItemsRepeater avoid starting over from scroll position 0. It also lets it quickly restore the UIElements for data that didn't change in a reset, resulting in better performance.

This example shows how to display a list of items in a vertical stack where *MyItemsSource* is a custom data source that wraps an underlying list of items. It exposes a *Data* property that can be used to re-assign a new list to use as the items source, which then triggers a reset.

XAML

```
<ScrollViewer x:Name="sv">
    <ItemsRepeater x:Name="repeater"
                   ItemsSource="{x:Bind MyItemsSource}"
                   ItemTemplate="{StaticResource MyTemplate}">
       <ItemsRepeater.Layout>
           <StackLayout ItemSpacing="8"/>
       </ItemsRepeater.Layout>
   </ItemsRepeater>
</ScrollViewer>
```

C#

```
public MainPage()
{
    this.InitializeComponent();

    // Similar to an ItemsControl, a developer sets the ItemsRepeater's
```

```
ItemsSource.
    // Here we provide our custom source that supports unique IDs which
enables
    // ItemsRepeater to be smart about handling resets from the data.
    // Unique IDs also make it easy to do things apply sorting/filtering
    // without impacting any state (i.e. selection).
    MyItemsSource myItemsSource = new MyItemsSource(data);

    repeater.ItemsSource = myItemsSource;

    // ...

    // We can sort/filter the data using whatever mechanism makes the
    // most sense (LINQ, database query, etc.) and then reassign
    // it, which in our implementation triggers a reset.
    myItemsSource.Data = someNewData;
}

// ...


public class MyItemsSource : IReadOnlyList<ItemBase>, IKeyIndexMapping,
INotifyCollectionChanged
{
    private IList<ItemBase> _data;

    public MyItemsSource(IEnumerable<ItemBase> data)
    {
        if (data == null) throw new ArgumentNullException();

        this._data = data.ToList();
    }

    public IList<ItemBase> Data
    {
        get { return _data; }
        set
        {
            _data = value;

            // Instead of tossing out existing elements and re-creating
them,
            // ItemsRepeater will reuse the existing elements and match them
up
            // with the data again.
            this.CollectionChanged?.Invoke(
                this,
                new
NotifyCollectionChangedEventArgs(NotifyCollectionChangedAction.Reset));
        }
    }

    #region IReadOnlyList<T>

    public ItemBase this[int index] => this.Data != null
```

```
            ? this.Data[index]
            : throw new IndexOutOfRangeException();

        public int Count => this.Data != null ? this.Data.Count : 0;
        public IEnumerator<ItemBase> GetEnumerator() =>
this.Data.GetEnumerator();
        IEnumerator IEnumerable.GetEnumerator() => this.GetEnumerator();

        #endregion

        #region INotifyCollectionChanged

        public event NotifyCollectionChangedEventHandler CollectionChanged;

        #endregion

        #region IKeyIndexMapping

        private int lastRequestedIndex = IndexNotFound;
        private const int IndexNotFound = -1;

        // When UniqueIDs are supported, the ItemsRepeater caches the unique ID
for each item
        // with the matching UIElement that represents the item.  When a reset
occurs the
        // ItemsRepeater pairs up the already generated UIElements with items in
the data
        // source.
        // ItemsRepeater uses IndexForUniqueId after a reset to probe the data
and identify
        // the new index of an item to use as the anchor.  If that item no
        // longer exists in the data source it may try using another cached
unique ID until
        // either a match is found or it determines that all the previously
visible items
        // no longer exist.
        public int IndexForUniqueId(string uniqueId)
        {
            // We'll try to increase our odds of finding a match sooner by
starting from the
            // position that we know was last requested and search forward.
            var start = lastRequestedIndex;
            for (int i = start; i < this.Count; i++)
            {
                if (this[i].PrimaryKey.Equals(uniqueId))
                    return i;
            }

            // Then try searching backward.
            start = Math.Min(this.Count - 1, lastRequestedIndex);
            for (int i = start; i >= 0; i--)
            {
                if (this[i].PrimaryKey.Equals(uniqueId))
                    return i;
            }
```

```
        return IndexNotFound;
    }

    public string UniqueIdForIndex(int index)
    {
        var key = this[index].PrimaryKey;
        lastRequestedIndex = index;
        return key;
    }

    #endregion
}
```

Create a custom collection control

You can use the ItemsRepeater to create a custom collection control complete with its own type of control to present each item.

> **Note**
>
> This is similar to using **ItemsControl**, but instead of deriving from **ItemsControl** and putting an **ItemsPresenter** in the control template, you derive from **Control** and insert an **ItemsRepeater** in the control template. The custom collection control "has an" **ItemsRepeater** versus "is an" **ItemsControl**. This implies that you must also explicitly choose which properties to expose, rather than which inherited properties to not support.

This example shows how to place an ItemsRepeater in the template of a custom control named *MediaCollectionView* and expose its properties.

XAML

```
<!-- xmlns:muxc="using:Microsoft.UI.Xaml.Controls" -->
<Style TargetType="local:MediaCollectionView">
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="local:MediaCollectionView">
                <Border
                    Background="{TemplateBinding Background}"
                    BorderBrush="{TemplateBinding BorderBrush}"
                    BorderThickness="{TemplateBinding BorderThickness}">
                    <ScrollViewer x:Name="ScrollViewer">
                        <muxc:ItemsRepeater x:Name="ItemsRepeater"
                                            ItemsSource="{TemplateBinding
ItemsSource}"
```

```
                                                ItemTemplate="{TemplateBinding
ItemTemplate}"
                                                Layout="{TemplateBinding
Layout}"
                                                TabFocusNavigation="
{TemplateBinding TabFocusNavigation}"/>
                    </ScrollViewer>
                </Border>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>
```

C#

```
public sealed class MediaCollectionView : Control
{
    public object ItemsSource
    {
        get { return (object)GetValue(ItemsSourceProperty); }
        set { SetValue(ItemsSourceProperty, value); }
    }

    // Using a DependencyProperty as the backing store for ItemsSource.
This enables animation, styling, binding, etc...
    public static readonly DependencyProperty ItemsSourceProperty =
        DependencyProperty.Register(nameof(ItemsSource), typeof(object),
typeof(MediaCollectionView), new PropertyMetadata(0));

    public DataTemplate ItemTemplate
    {
        get { return (DataTemplate)GetValue(ItemTemplateProperty); }
        set { SetValue(ItemTemplateProperty, value); }
    }

    // Using a DependencyProperty as the backing store for ItemTemplate.
This enables animation, styling, binding, etc...
    public static readonly DependencyProperty ItemTemplateProperty =
        DependencyProperty.Register(nameof(ItemTemplate),
typeof(DataTemplate), typeof(MediaCollectionView), new PropertyMetadata(0));

    public Layout Layout
    {
        get { return (Layout)GetValue(LayoutProperty); }
        set { SetValue(LayoutProperty, value); }
    }

    // Using a DependencyProperty as the backing store for Layout.  This
enables animation, styling, binding, etc...
    public static readonly DependencyProperty LayoutProperty =
        DependencyProperty.Register(nameof(Layout), typeof(Layout),
typeof(MediaCollectionView), new PropertyMetadata(0));
```

```
    public MediaCollectionView()
    {
        this.DefaultStyleKey = typeof(MediaCollectionView);
    }
}
```

Display grouped items

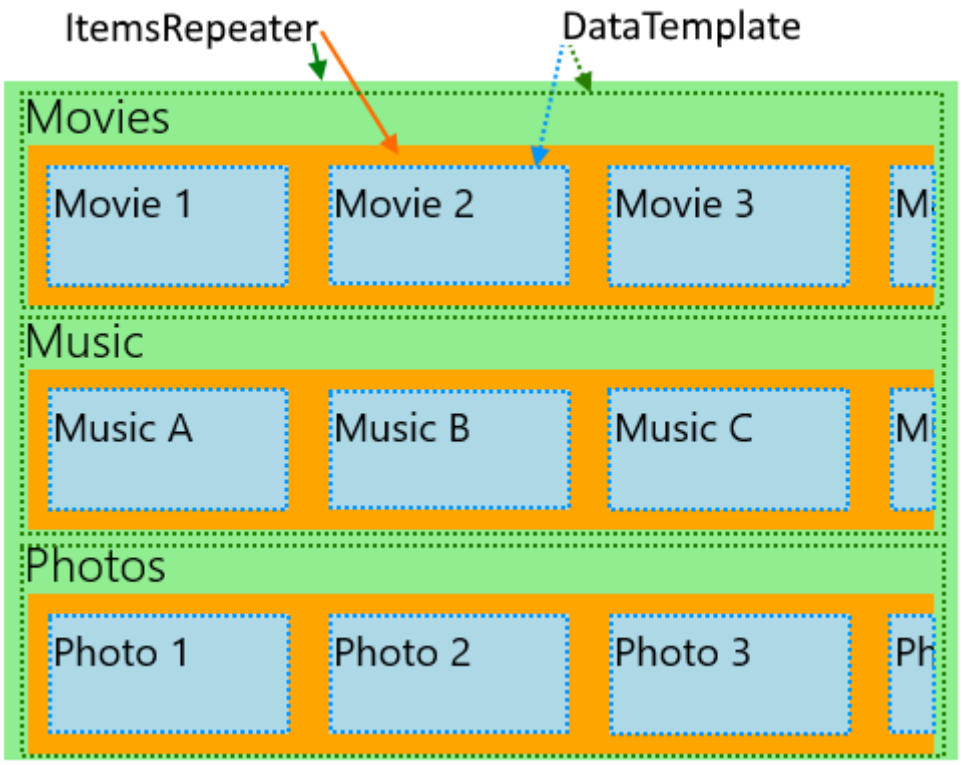
You can nest an ItemsRepeater in the ItemTemplate of another ItemsRepeater to create nested virtualizing layouts. The framework will make efficient use of resources by minimizing unnecessary realization of elements that aren't visible or near the current viewport.

This example shows how you can display a list of grouped items in a vertical stack. The outer ItemsRepeater generates each group. In the template for each group, another ItemsRepeater generates the items.

XAML

```
<!-- xmlns:muxc="using:Microsoft.UI.Xaml.Controls" -->

<Page.Resources>
    <muxc:StackLayout x:Key="MyGroupLayout"/>
    <muxc:StackLayout x:Key="MyItemLayout" Orientation="Horizontal"/>
</Page.Resources>

<ScrollViewer>
  <muxc:ItemsRepeater ItemsSource="{x:Bind AppNotifications}"
                      Layout="{StaticResource MyGroupLayout}">
    <muxc:ItemsRepeater.ItemTemplate>
      <DataTemplate x:DataType="ExampleApp:AppNotifications">
        <!-- Group -->
        <StackPanel>
          <!-- Header -->
          <TextBlock Text="{x:Bind AppTitle}"/>
          <!-- Items -->
          <muxc:ItemsRepeater ItemsSource="{x:Bind Notifications}"
                              Layout="{StaticResource MyItemLayout}"
                              ItemTemplate="{StaticResource MyTemplate}"/>
          <!-- Footer -->
          <Button Content="{x:Bind FooterText}"/>
        </StackPanel>
      </DataTemplate>
    </muxc:ItemsRepeater.ItemTemplate>
  </muxc:ItemsRepeater>
</ScrollViewer>
```

The image below shows the basic layout that's created using the above sample as a guideline.



This next example shows a layout for an app that has various categories that can change with user preference and are presented as horizontally scrolling lists. The layout of this example is also represented by the image above.

XAML

```
<!-- xmlns:muxc="using:Microsoft.UI.Xaml.Controls" -->
<!-- Include the <muxc:ItemsRepeaterScrollHost> if targeting Windows 10
versions earlier than 1809. -->
<ScrollViewer>
  <muxc:ItemsRepeater ItemsSource="{x:Bind Categories}"
                      Background="LightGreen">
    <muxc:ItemsRepeater.ItemTemplate>
      <DataTemplate x:DataType="local:Category">
        <StackPanel Margin="12,0">
          <TextBlock Text="{x:Bind Name}" Style="{ThemeResource
TitleTextBlockStyle}"/>
          <!-- Include the <muxc:ItemsRepeaterScrollHost> if targeting
Windows 10 versions earlier than 1809. -->
          <ScrollViewer HorizontalScrollMode="Enabled"
                                      VerticalScrollMode="Disabled"

HorizontalScrollBarVisibility="Auto" >
            <muxc:ItemsRepeater ItemsSource="{x:Bind Items}"
                                Background="Orange">
              <muxc:ItemsRepeater.ItemTemplate>
                <DataTemplate x:DataType="local:CategoryItem">
                  <Grid Margin="10"
```

```
                        Height="60" Width="120"
                        Background="LightBlue">
                    <TextBlock Text="{x:Bind Name}"
                               Style="{StaticResource
SubtitleTextBlockStyle}"
                               Margin="4"/>
                  </Grid>
                </DataTemplate>
              </muxc:ItemsRepeater.ItemTemplate>
              <muxc:ItemsRepeater.Layout>
                <muxc:StackLayout Orientation="Horizontal"/>
              </muxc:ItemsRepeater.Layout>
            </muxc:ItemsRepeater>
          </ScrollViewer>
        </StackPanel>
      </DataTemplate>
    </muxc:ItemsRepeater.ItemTemplate>
  </muxc:ItemsRepeater>
</ScrollViewer>
```

Bringing an Element Into View

The XAML framework already handles bringing a FrameworkElement into view when it either 1) receives keyboard focus or 2) receives Narrator focus. There may be other cases where you need to explicitly bring an element into view. For example, in response to a user action or to restore the state of the UI after a page navigation.

Bringing a virtualized element into view involves the following:

1. Realize a UIElement for an item
2. Run the layout to ensure the element has a valid position
3. Initiate a request to bring the realized element into view

The example below demonstrates these steps as part of restoring the scroll position of an item in a flat, vertical list after a page navigation. In the case of hierarchical data using nested ItemsRepeaters the approach is essentially the same, but must be done at each level of the hierarchy.

XAML

```
<ScrollViewer x:Name="scrollviewer">
  <ItemsRepeater x:Name="repeater" .../>
</ScrollViewer>
```

C#

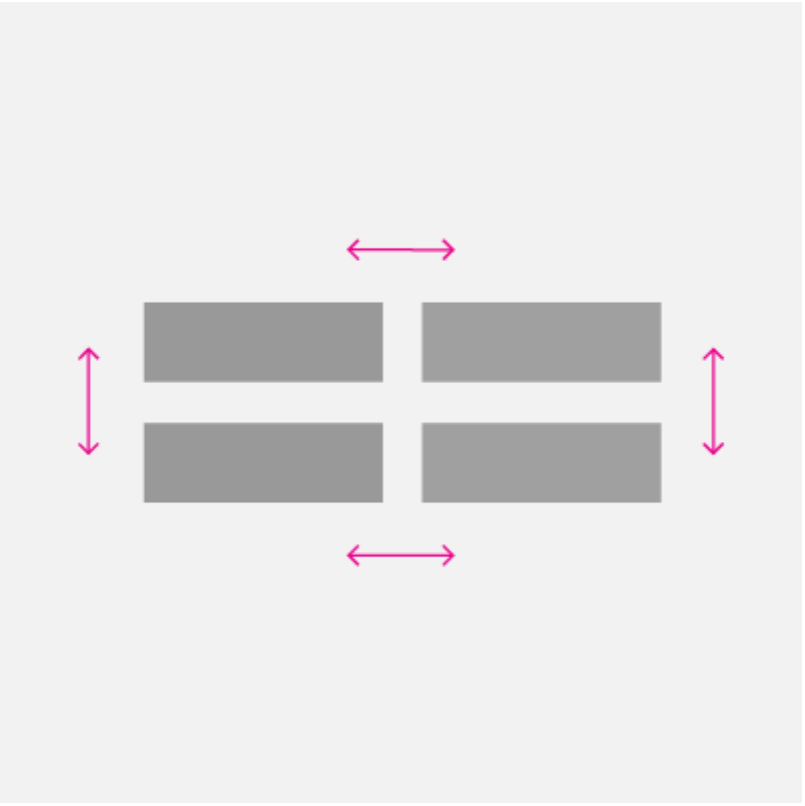
```
public class MyPage : Page
{
    // ...

     protected override void OnNavigatedTo(NavigationEventArgs e)
    {
        base.OnNavigatedTo(e);

        // retrieve saved offset + index(es) of the tracked element and then
bring it into view.
        // ...

        var element = repeater.GetOrCreateElement(index);

        // ensure the item is given a valid position
        element.UpdateLayout();

        element.StartBringIntoView(new BringIntoViewOptions()
        {
            VerticalOffset = relativeVerticalOffset
        });
    }

    protected override void OnNavigatingFrom(NavigatingCancelEventArgs e)
    {
        base.OnNavigatingFrom(e);

        // retrieve and save the relative offset and index(es) of the
scrollviewer's current anchor element ...
        var anchor = this.scrollviewer.CurrentAnchor;
        var index = this.repeater.GetElementIndex(anchor);
        var anchorBounds =
anchor.TransformToVisual(this.scrollviewer).TransformBounds(new Rect(0, 0,
anchor.ActualSize.X, anchor.ActualSize.Y));
        relativeVerticalOffset = this.scrollviewer.VerticalOffset -
anchorBounds.Top;
    }
}
```

Enable Accessibility

ItemsRepeater does not provide a default accessibility experience. The documentation on Usability for Windows apps provides a wealth of information to help you ensure your app provides an inclusive user experience. If you're using the ItemsRepeater to create a custom control then be sure to see the documentation on Custom automation peers.

Keyboarding

The minimal keyboarding support for focus movement that ItemsRepeater provides is based on XAML's 2D Directional Navigation for Keyboarding.



The ItemsRepeater's XYFocusKeyboardNavigation mode is *Enabled* by default. Depending on the intended experience, consider adding support for common Keyboard Interactions such as Home, End, PageUp, and PageDown.

The ItemsRepeater does automatically ensure that the default tab order for its items (whether virtualized or not) follows the same order that the items are given in the data. By default the ItemsRepeater has its TabFocusNavigation property set to Once instead of the common default of *Local*.

> **Note**
>
> The ItemsRepeater does not automatically remember the last focused item. This means that when a user is using Shift+Tab they may be taken to the last realized item.

Announcing "Item *X* of *Y*" in Screen Readers

You need to manage setting the appropriate automation properties, such as values for **PositionInSet** and **SizeOfSet**, and ensure they remain up-to-date when items are added, moved, removed, etc.

In some custom layouts there may not be an obvious sequence to the visual order. Users minimally expect that the values for the PositionInSet and SizeOfSet properties used by screen readers will match the order the items appear in the data (offset by 1 to match natural counting versus being 0-based).

The best way to achieve this is by having the automation peer for the item control implement the GetPositionInSetCore and GetSizeOfSetCore methods and report the position of the item in the data set represented by the control. The value is only computed at run-time when accessed by an assistive technology and keeping it up-to-date becomes a non-issue. The value matches the data order.

This example shows how you could do this when presenting a custom control called *CardControl*.

XAML

```
<ScrollViewer >
    <ItemsRepeater x:Name="repeater" ItemsSource="{x:Bind MyItemsSource}">
       <ItemsRepeater.ItemTemplate>
           <DataTemplate x:DataType="local:CardViewModel">
               <local:CardControl Item="{x:Bind}"/>
           </DataTemplate>
       </ItemsRepeater.ItemTemplate>
   </ItemsRepeater>
</ScrollViewer>
```

C#

```
internal sealed class CardControl : CardControlBase
{
    protected override AutomationPeer OnCreateAutomationPeer() => new
CardControlAutomationPeer(this);

    private sealed class CardControlAutomationPeer :
FrameworkElementAutomationPeer
    {
        private readonly CardControl owner;

        public CardControlAutomationPeer(CardControl owner) : base(owner) =>
this.owner = owner;

        protected override int GetPositionInSetCore()
          => ((ItemsRepeater)owner.Parent)?.GetElementIndex(this.owner) + 1
?? base.GetPositionInSetCore();

        protected override int GetSizeOfSetCore()
          => ((ItemsRepeater)owner.Parent)?.ItemsSourceView?.Count ??
base.GetSizeOfSetCore();
```

```
    }
}
```

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The ItemsRepeater for UWP apps requires WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in the Microsoft.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** ScrollViewer class
- ✓ **WinUI 2 Apis:** ItemsRepeater class
- ✓ Open the WinUI 2 Gallery app and see the ItemsRepeater in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:ItemsRepeater />
```

Related articles

- Lists
- ItemsRepeater
- ScrollViewer

Item containers and templates

Article • 06/09/2022

ListView and **GridView** controls manage how their items are arranged (horizontal, vertical, wrapping, etc...) and how a user interacts with the items, but not how the individual items are shown on the screen. Item visualization is managed by item containers. When you add items to a list view they are automatically placed in a container. The default item container for ListView is ListViewItem; for GridView, it's GridViewItem.

> **Important APIs**: ListView class, GridView class, ListViewItem class, GridViewItem class, ItemTemplate property, ItemContainerStyle property

> **Note**
>
> ListView and GridView both derive from the **ListViewBase** class, so they have the same functionality, but display data differently. In this article, when we talk about list view, the info applies to both the ListView and GridView controls unless otherwise specified. We may refer to classes like ListView or ListViewItem, but the *List* prefix can be replaced with *Grid* for the corresponding grid equivalent (GridView or GridViewItem).

ListView items and GridView items

As mentioned above, ListView items are automatically placed into the ListViewItem container, and GridView items are placed into the GridViewItem container. These item containers are controls that have their own built-in styling and interaction, but can also be highly customized. However, before customization, make sure to familizarize yourself with the recommended styling and guidelines for ListViewItem and GridViewItem:

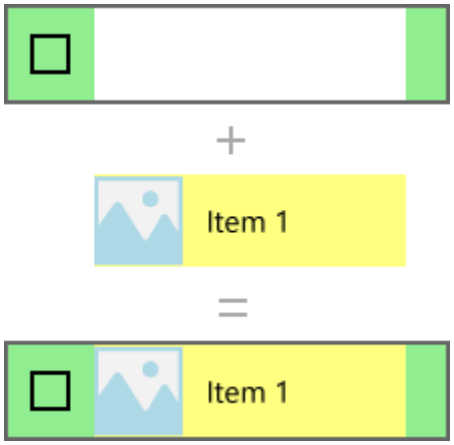
- **ListViewItems** - Items are primarily text-focused, and elongated in shape. Icons or images may appear to the left of the text.
- **GridViewItems** - Items are usually square in shape, or at least less of an elongated rectangle shape. Items are image-focused, and may have text appearing around or overlaid on the image.

Introduction to customization

Container controls (such as ListViewItem and GridViewItem) consist of two important parts that combine to create the final visuals shown for an item: the *data template* and the *control template*.

- **Data template** - You assign a DataTemplate to the ItemTemplate property of the list view to specify how individual data items are shown.
- **Control template** - The control template provides the part of the item visualization that the framework is responsible for, like visual states. You can use the ItemContainerStyle property to modify the control template. Typically, you do this to modify the list view colors to match your branding, or change how selected items are shown.

This image shows how the control template and the data template combine to create the final visual for an item.



Here's the XAML that creates this item. We explain the templates later.

XAML

```
<ListView Width="220" SelectionMode="Multiple">
    <ListView.ItemTemplate>
        <DataTemplate x:DataType="x:String">
            <Grid Background="Yellow">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="54"/>
                    <ColumnDefinition/>
                </Grid.ColumnDefinitions>
                <Image Source="Assets/placeholder.png" Width="44"
Height="44"
                       HorizontalAlignment="Left"/>
                <TextBlock Text="{x:Bind}" Foreground="Black"
                           FontSize="14" Grid.Column="1"
                           VerticalAlignment="Center"
                           Padding="0,0,54,0"/>
            </Grid>
        </DataTemplate>
    </ListView.ItemTemplate>
    <ListView.ItemContainerStyle>
```

```
        <Style TargetType="ListViewItem">
            <Setter Property="Background" Value="LightGreen"/>
        </Style>
    </ListView.ItemContainerStyle>
    <x:String>Item 1</x:String>
    <x:String>Item 2</x:String>
    <x:String>Item 3</x:String>
    <x:String>Item 4</x:String>
    <x:String>Item 5</x:String>
</ListView>
```

> **Important**
>
> Data templates and control templates are used to customize the style for many controls other than ListView and GridView. These include controls with their own built-in styling, such as FlipView, and custom-created controls, such as ItemsRepeater. While the below example is specific to ListView/GridView, the concepts can be applied to many other controls.

Prerequisites

- We assume that you know how to use a list view control. For more info, see the ListView and GridView article.
- We also assume that you understand control styles and templates, including how to use a style inline or as a resource. For more info, see Styling controls and Control templates.

The data

Before we look deeper into how to show data items in a list view, we need to understand the data to be shown. In this example, we create a data type called `NamedColor`. It combines a color name, color value, and a **SolidColorBrush** for the color, which are exposed as 3 properties: `Name`, `Color`, and `Brush`.

We then populate a **List** with a `NamedColor` object for each named color in the Colors class. The list is set as the ItemsSource for the list view.

Here's the code to define the class and populate the `NamedColors` list.

C#

```
C#
```

```
using System.Collections.Generic;
using System.Linq;
using System.Reflection;
using Windows.UI;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;

namespace ColorsListApp
{
    public sealed partial class MainPage : Page
    {
        // The list of colors won't change after it's populated, so we use
List<T>.
        // If the data can change, we should use an ObservableCollection<T>
intead.
        List<NamedColor> NamedColors = new List<NamedColor>();

        public MainPage()
        {
            this.InitializeComponent();

            // Use reflection to get all the properties of the Colors class.
            IEnumerable<PropertyInfo> propertyInfos =
typeof(Colors).GetRuntimeProperties();

            // For each property, create a NamedColor with the property name
(color name),
            // and property value (color value). Add it the NamedColors
list.
            for (int i = 0; i < propertyInfos.Count(); i++)
            {
                NamedColors.Add(new
NamedColor(propertyInfos.ElementAt(i).Name,

(Color)propertyInfos.ElementAt(i).GetValue(null)));
            }

            colorsListView.ItemsSource = NamedColors;
        }
    }

    class NamedColor
    {
        public NamedColor(string colorName, Color colorValue)
        {
            Name = colorName;
            Color = colorValue;
        }

        public string Name { get; set; }

        public Color Color { get; set; }

        public SolidColorBrush Brush
```

```
        {
            get { return new SolidColorBrush(Color); }
        }
    }
}
```

Data template

You specify a data template to tell the list view how your data item should be shown.

By default, a data item is displayed in the list view as the string representation of the data object it's bound to. If you show the 'NamedColors' data in a list view without telling the list view how it should look, it just shows whatever the **ToString** method returns, like this.

XAML

XAML

```
<ListView x:Name="colorsListView"/>
```

ColorsListApp.NamedColor

ColorsListApp.NamedColor

ColorsListApp.NamedColor

ColorsListApp.NamedColor

ColorsListApp.NamedColor

You can show the string representation of a particular property of the data item by setting the DisplayMemberPath to that property. Here, you set DisplayMemberPath to the `Name` property of the `NamedColor` item.

XAML

XAML

```
<ListView x:Name="colorsListView" DisplayMemberPath="Name" />
```

The list view now displays items by name, as shown here. It's more useful, but it's not very interesting and leaves a lot of information hidden.

Blue

BlueViolet

Brown

BurlyWood

CadetBlue

You typically want to show a more rich presentation of your data. To specify exactly how items in the list view are displayed, you create a DataTemplate. The XAML in the DataTemplate defines the layout and appearance of controls used to display an individual item. The controls in the layout can be bound to properties of a data object, or have static content defined inline. You assign the DataTemplate to the ItemTemplate property of the list control.

> **ⓘ Important**
>
> You can't use a **ItemTemplate** and **DisplayMemberPath** at the same time. If both properties are set, an exception occurs.

Here, you define a DataTemplate that shows a Rectangle in the color of the item, along with the color name and RGB values.

> **ⓘ Note**
>
> When you use the **x:Bind markup extension** in a DataTemplate, you have to specify the DataType (`x:DataType`) on the DataTemplate.

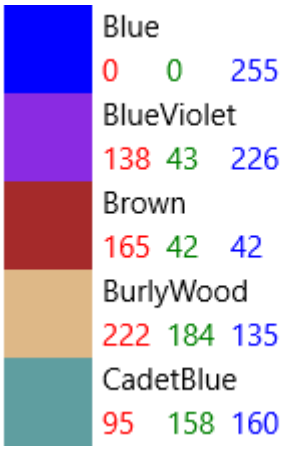
XAML

XAML

```
<ListView x:Name="colorsListView">
    <ListView.ItemTemplate>
        <DataTemplate x:DataType="local:NamedColor">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition MinWidth="54"/>
                    <ColumnDefinition Width="32"/>
                    <ColumnDefinition Width="32"/>
                    <ColumnDefinition Width="32"/>
                    <ColumnDefinition/>
```

```
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition/>
                    <RowDefinition/>
                </Grid.RowDefinitions>
                <Rectangle Width="44" Height="44" Fill="{x:Bind Brush}"
Grid.RowSpan="2"/>
                <TextBlock Text="{x:Bind Name}" Grid.Column="1"
Grid.ColumnSpan="4"/>
                <TextBlock Text="{x:Bind Color.R}" Grid.Column="1"
Grid.Row="1" Foreground="Red"/>
                <TextBlock Text="{x:Bind Color.G}" Grid.Column="2"
Grid.Row="1" Foreground="Green"/>
                <TextBlock Text="{x:Bind Color.B}" Grid.Column="3"
Grid.Row="1" Foreground="Blue"/>
            </Grid>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Here's what the data items look like when they're displayed with this data template.



> **ⓘ Important**
>
> ListViewItems by default have their content aligned left, i.e. their HorizontalContentAlignmentProperty is set to Left. If you have multiple elements within a ListViewItem that are horizontally adjacent, such as horizontally stacked elements or elements placed in the same Grid row, they will all be left-aligned and only separated by their defined margin.
>
> In order to have elements spread to fill the entire body of a ListItem, you will need to set the HorizontalContentAlignmentProperty to Stretch by using a Setter inside of your ListView:

XAML

```
<ListView.ItemContainerStyle>
    <Style TargetType="ListViewItem">
        <Setter Property="HorizontalContentAlignment" Value="Stretch"/>
    </Style>
</ListView.ItemContainerStyle>
```

You might want to show the data in a GridView. Here's another data template that displays the data in a way that's more appropriate for a grid layout. This time, the data template is defined as a resource rather than inline with the XAML for the GridView.

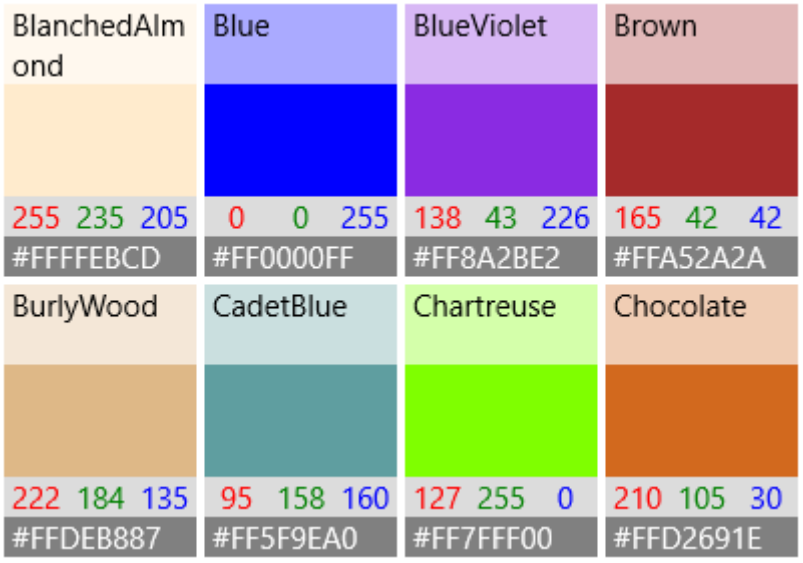
XAML

XAML

```
<Page.Resources>
    <DataTemplate x:Key="namedColorItemGridTemplate"
x:DataType="local:NamedColor">
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="32"/>
                <ColumnDefinition Width="32"/>
                <ColumnDefinition Width="32"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="96"/>
                <RowDefinition/>
                <RowDefinition/>
            </Grid.RowDefinitions>

            <Rectangle Width="96" Height="96" Fill="{x:Bind Brush}"
Grid.ColumnSpan="3" />
            <!-- Name -->
            <Border Background="#AAFFFFFF" Grid.ColumnSpan="3" Height="40"
VerticalAlignment="Top">
                <TextBlock Text="{x:Bind Name}" TextWrapping="Wrap"
Margin="4,0,0,0"/>
            </Border>
            <!-- RGB -->
            <Border Background="Gainsboro" Grid.Row="1"
Grid.ColumnSpan="3"/>
            <TextBlock Text="{x:Bind Color.R}" Foreground="Red"
                   Grid.Column="0" Grid.Row="1"
HorizontalAlignment="Center"/>
            <TextBlock Text="{x:Bind Color.G}" Foreground="Green"
                   Grid.Column="1" Grid.Row="1"
HorizontalAlignment="Center"/>
            <TextBlock Text="{x:Bind Color.B}" Foreground="Blue"
                   Grid.Column="2" Grid.Row="1"
HorizontalAlignment="Center"/>
            <!-- HEX -->
            <Border Background="Gray" Grid.Row="2" Grid.ColumnSpan="3">
```

```
                <TextBlock Text="{x:Bind Color}" Foreground="White"
Margin="4,0,0,0"/>
            </Border>
        </Grid>
    </DataTemplate>
</Page.Resources>

...

<GridView x:Name="colorsGridView"
          ItemTemplate="{StaticResource namedColorItemGridTemplate}"/>
```

When the data is shown in a grid using this data template, it looks like this.



Performance considerations

Data templates are the primary way you define the look of your list view. They can also have a significant impact on performance if your list displays a large number of items.

An instance of every XAML element in a data template is created for each item in the list view. For example, the grid template in the previous example has 10 XAML elements (1 Grid, 1 Rectangle, 3 Borders, 5 TextBlocks). A GridView that shows 20 items on screen using this data template creates at least 200 elements (20*10=200). Reducing the number of elements in a data template can greatly reduce the total number of elements created for your list view. For more info, see ListView and GridView UI optimization: Element count reduction per item.

Consider this section of the grid data template. Let's look at a few things that reduce the element count.

XAML

```
```

XAML

```
<!-- RGB -->
<Border Background="Gainsboro" Grid.Row="1" Grid.ColumnSpan="3"/>
<TextBlock Text="{x:Bind Color.R}" Foreground="Red"
           Grid.Column="0" Grid.Row="1" HorizontalAlignment="Center"/>
<TextBlock Text="{x:Bind Color.G}" Foreground="Green"
           Grid.Column="1" Grid.Row="1" HorizontalAlignment="Center"/>
<TextBlock Text="{x:Bind Color.B}" Foreground="Blue"
           Grid.Column="2" Grid.Row="1" HorizontalAlignment="Center"/>
```

- First, the layout uses a single Grid. You could have a single-column Grid and place these 3 TextBlocks in a StackPanel, but in a data template that gets created many times, you should look for ways to avoid embedding layout panels within other layout panels.
- Second, you can use a Border control to render a background without actually placing items within the Border element. A Border element can have only one child element, so you would need to add an additional layout panel to host the 3 TextBlock elements within the Border element in XAML. By not making the TextBlocks children of the Border, you eliminate the need for a panel to hold the TextBlocks.
- Finally, you could place the TextBlocks inside a StackPanel, and set the border properties on the StackPanel rather than using an explicit Border element. However, the Border element is a more lightweight control than a StackPanel, so it has less of an impact on performance when rendered many times over.

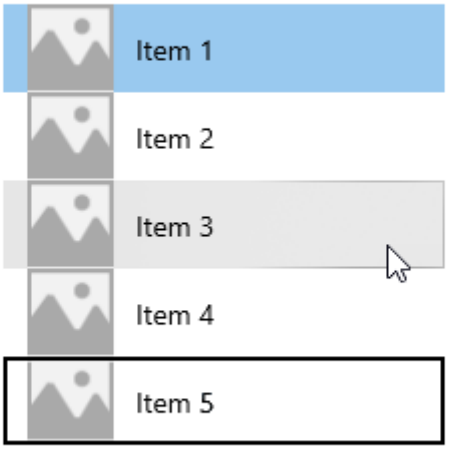
Using different layouts for different items

Control template

An item's control template contains the visuals that display state, like selection, pointer over, and focus. These visuals are rendered either on top of or below the data template. Some of the common default visuals drawn by the ListView control template are shown here.

- Hover – A light gray rectangle drawn below the data template.
- Selection – A light blue rectangle drawn below the data template.
- Keyboard focus– A High Visibility Focus Visual drawn on top of the item template.

The list view combines the elements from the data template and control template to create the final visuals rendered on the screen. Here, the state visuals are shown in the context of a list view.



ListViewItemPresenter

As we noted previously about data templates, the number of XAML elements created for each item can have a significant impact on the performance of a list view. Because the data template and control template are combined to display each item, the actual number of elements needed to display an item includes the elements in both templates.

The ListView and GridView controls are optimized to reduce the number of XAML elements created per item. The **ListViewItem** visuals are created by the ListViewItemPresenter, which is a special XAML element that displays complex visuals for focus, selection, and other visual states, without the overhead of numerous UIElements.

> **Note**

In UWP apps for Windows, both **ListViewItem** and **GridViewItem** use **ListViewItemPresenter**; the GridViewItemPresenter is deprecated and you should not use it. ListViewItem and GridViewItem set different property values on ListViewItemPresenter to achieve different default looks.)

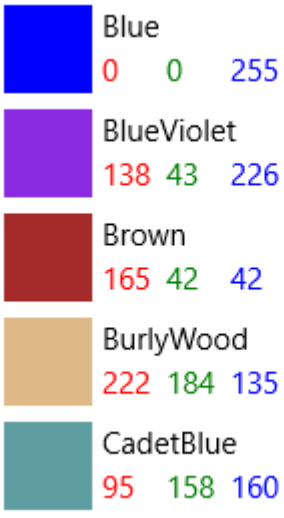
To modify the look of the item container, use the ItemContainerStyle property and provide a Style with its TargetType set to **ListViewItem** or **GridViewItem**.

In this example, you add padding to the ListViewItem to create some space between the items in the list.

XAML

```
<ListView x:Name="colorsListView">
    <ListView.ItemTemplate>
        <!-- DataTemplate XAML shown in previous ListView example -->
    </ListView.ItemTemplate>

    <ListView.ItemContainerStyle>
        <Style TargetType="ListViewItem">
            <Setter Property="Padding" Value="0,4"/>
        </Style>
    </ListView.ItemContainerStyle>
</ListView>
```

Now the list view looks like this with space between the items.



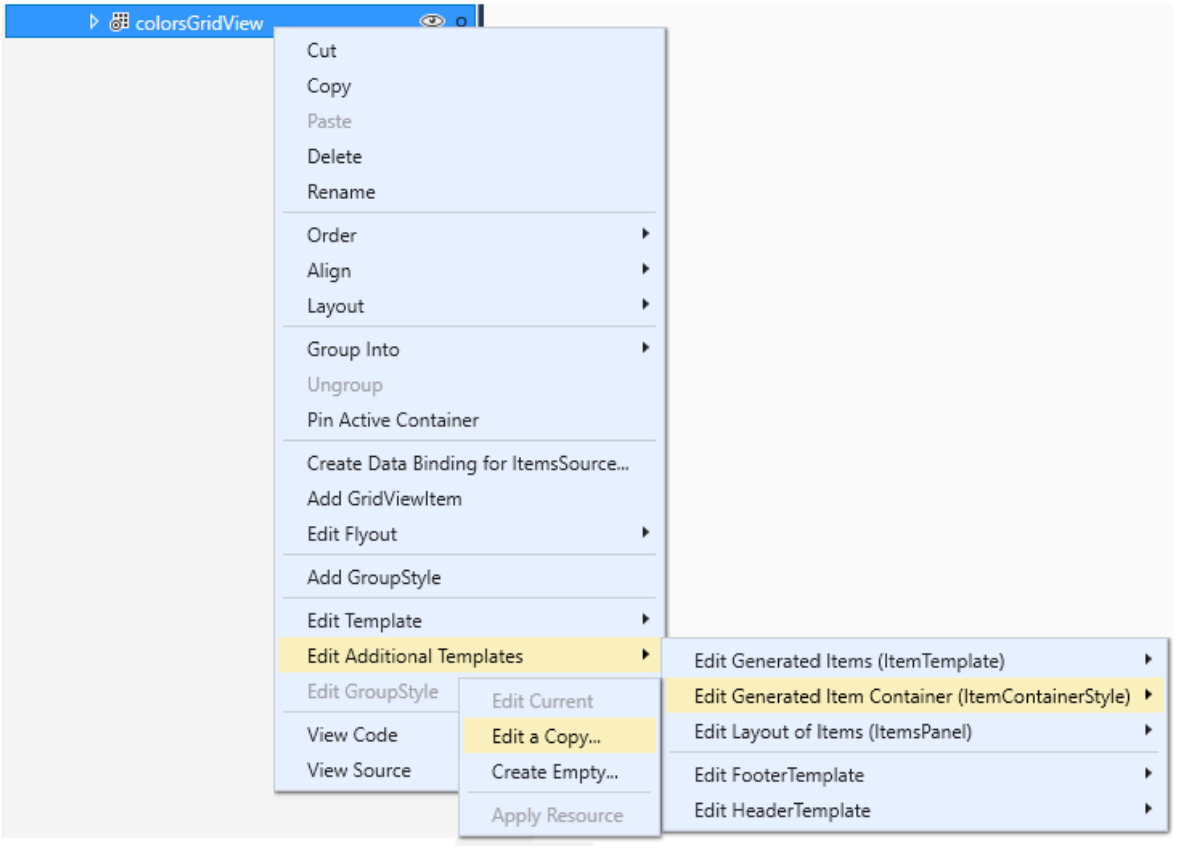
In the ListViewItem default style, the ListViewItemPresenter **ContentMargin** property has a TemplateBinding to the ListViewItem **Padding** property (`<ListViewItemPresenter ContentMargin="{TemplateBinding Padding}"/>`). When we set the Padding property, that value is really being passed to the ListViewItemPresenter ContentMargin property.

To modify other ListViewItemPresenter properties that aren't template bound to ListViewItems properties, you need to retemplate the ListViewItem with a new ListViewItemPresenter that you can modify properties on.

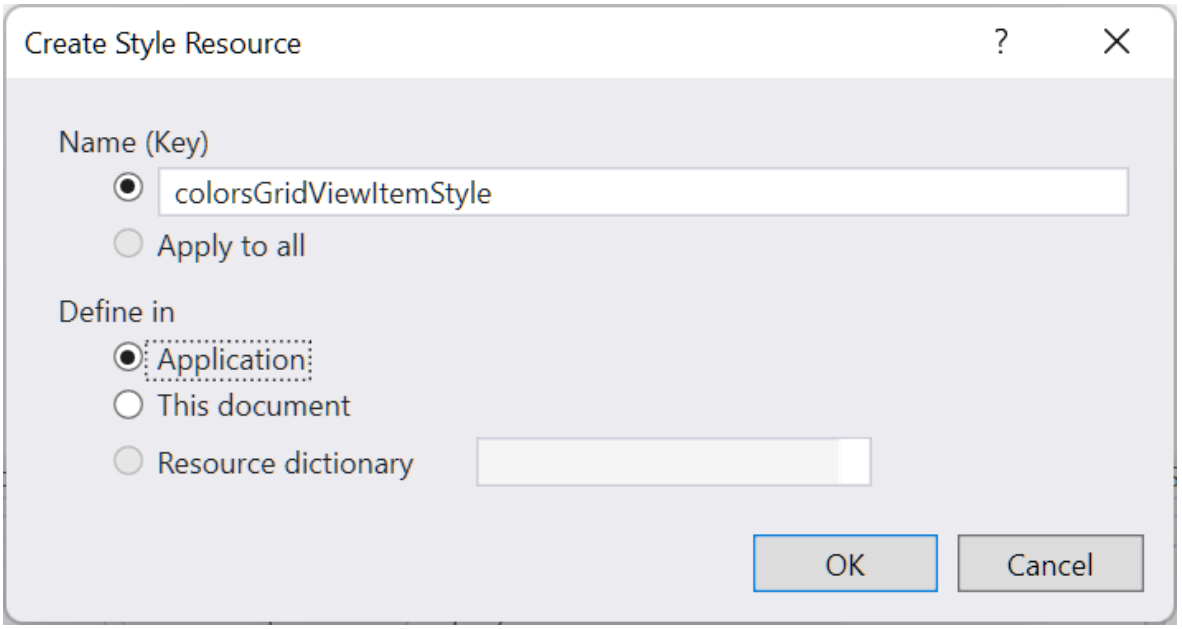
> **Note**
>
> ListViewItem and GridViewItem default styles set a lot of properties on ListViewItemPresenter. You should always start with a copy of the default style and modify only the properties you need too. Otherwise, the visuals will probably not show up the way you expect because some properties won't be set correctly.

To make a copy of the default template in Visual Studio

1. Open the Document Outline pane (**View** > **Other Windows** > **Document Outline**).
2. Select the list or grid element to modify. In this example, you modify the `colorsGridView` element.
3. Right-click and select **Edit Additional Templates** > **Edit Generated Item Container (ItemContainerStyle)** > **Edit a Copy**.



4. In the Create Style Resource dialog, enter a name for the style. In this example, you use `colorsGridViewItemStyle`.



A copy of the default style is added to your app as a resource, and the **GridView.ItemContainerStyle** property is set to that resource, as shown in this XAML.

XAML

```
<Style x:Key="colorsGridViewItemStyle" TargetType="GridViewItem">
    <Setter Property="FontFamily" Value="{ThemeResource
ContentControlThemeFontFamily}"/>
    <Setter Property="FontSize" Value="{ThemeResource
ControlContentThemeFontSize}" />
    <Setter Property="Background" Value="Transparent"/>
    <Setter Property="Foreground" Value="{ThemeResource
SystemControlForegroundBaseHighBrush}"/>
    <Setter Property="TabNavigation" Value="Local"/>
    <Setter Property="IsHoldingEnabled" Value="True"/>
    <Setter Property="HorizontalContentAlignment" Value="Center"/>
    <Setter Property="VerticalContentAlignment" Value="Center"/>
    <Setter Property="Margin" Value="0,0,4,4"/>
    <Setter Property="MinWidth" Value="{ThemeResource
GridViewItemMinWidth}"/>
    <Setter Property="MinHeight" Value="{ThemeResource
GridViewItemMinHeight}"/>
    <Setter Property="Template">
        <Setter.Value>
            <ControlTemplate TargetType="GridViewItem">
                <ListViewItemPresenter
                    CheckBrush="{ThemeResource
SystemControlForegroundBaseMediumHighBrush}"
                    ContentMargin="{TemplateBinding Padding}"
                    CheckMode="Overlay"
                    ContentTransitions="{TemplateBinding
ContentTransitions}"
                    CheckBoxBrush="{ThemeResource
SystemControlBackgroundChromeMediumBrush}"
                    DragForeground="{ThemeResource
ListViewItemDragForegroundThemeBrush}"
```

```
                    DragOpacity="{ThemeResource
ListViewItemDragThemeOpacity}"
                    DragBackground="{ThemeResource
ListViewItemDragBackgroundThemeBrush}"
                    DisabledOpacity="{ThemeResource
ListViewItemDisabledThemeOpacity}"
                    FocusBorderBrush="{ThemeResource
SystemControlForegroundAltHighBrush}"
                    FocusSecondaryBorderBrush="{ThemeResource
SystemControlForegroundBaseHighBrush}"
                    HorizontalContentAlignment="{TemplateBinding
HorizontalContentAlignment}"
                    PointerOverForeground="{ThemeResource
SystemControlForegroundBaseHighBrush}"
                    PressedBackground="{ThemeResource
SystemControlHighlightListMediumBrush}"
                    PlaceholderBackground="{ThemeResource
ListViewItemPlaceholderBackgroundThemeBrush}"
                    PointerOverBackground="{ThemeResource
SystemControlHighlightListLowBrush}"
                    ReorderHintOffset="{ThemeResource
GridViewItemReorderHintThemeOffset}"
                    SelectedPressedBackground="{ThemeResource
SystemControlHighlightListAccentHighBrush}"
                    SelectionCheckMarkVisualEnabled="True"
                    SelectedForeground="{ThemeResource
SystemControlForegroundBaseHighBrush}"
                    SelectedPointerOverBackground="{ThemeResource
SystemControlHighlightListAccentMediumBrush}"
                    SelectedBackground="{ThemeResource
SystemControlHighlightAccentBrush}"
                    VerticalContentAlignment="{TemplateBinding
VerticalContentAlignment}"/>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

...

<GridView x:Name="colorsGridView" ItemContainerStyle="{StaticResource
colorsGridViewItemStyle}"/>
```

You can now modify properties on the ListViewItemPresenter to control the selection check box, item positioning, and brush colors for visual states.

Inline and Overlay selection visuals

ListView and GridView indicate selected items in different ways depending on the control and the SelectionMode. For more info about list view selection, see ListView and GridView.

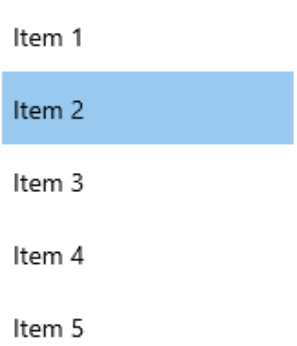
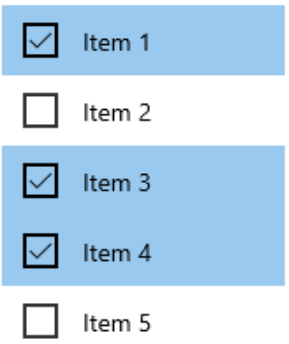
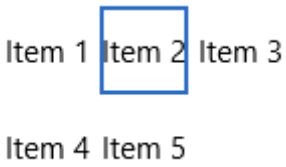
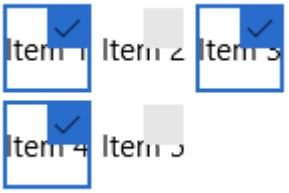
When **SelectionMode** is set to **Multiple**, a selection check box is shown as part of the item's control template. You can use the SelectionCheckMarkVisualEnabled property to turn off the selection check box in Multiple selection mode. However, this property is ignored in other selection modes, so you can't turn on the check box in Extended or Single selection mode.

You can set the CheckMode property to specify whether the check box is shown using the inline style or overlay style.

- **Inline**: This style shows the check box to the left of the content, and colors the background of the item container to indicate selection. This is the default style for ListView.
- **Overlay**: This style shows the check box on top of the content, and colors only the border of the item container to indicate selection. This is the default style for GridView.

This table shows the default visuals used to indicate selection.

Expand table

SelectionMode:	Single/Extended	Multiple
Inline		
Overlay		

Note

In this and the following examples, simple string data items are shown without data templates to emphasize the visuals provided by the control template.

There are also several brush properties to change the colors of the check box. We'll look at these next along with other brush properties.

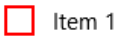
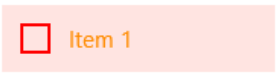
Brushes

Many of the properties specify the brushes used for different visual states. You might want to modify these to match the color of your brand.

This table shows the Common and Selection visual states for ListViewItem, and the brushes used to render the visuals for each state. The images show the effects of the brushes on both the inline and overlay selection visual styles.

> **Note**
>
> In this table, the modified color values for the brushes are hardcoded named colors and the colors are selected to make it more apparent where they are applied in the template. These are not the default colors for the visual states. If you modify the default colors in your app, you should use brush resources to modify the color values as done in the default template.

Expand table

State/Brush name	Inline style	Overlay style
Normal • **CheckBoxBrush="Red"**		
PointerOver • **PointerOverForeground="DarkOrange"** • **PointerOverBackground="MistyRose"** • CheckBoxBrush="Red"		
Pressed • **PressedBackground="LightCyan"** • PointerOverForeground="DarkOrange"		

State/Brush name	Inline style	Overlay style
• CheckBoxBrush="Red"		
Selected • **SelectedForeground="Navy"** • **SelectedBackground="Khaki"** • **CheckBrush="Green"** • CheckBoxBrush="Red" (inline only)		
PointerOverSelected • **SelectedPointerOverBackground="Lavender"** • SelectedForeground="Navy" • SelectedBackground="Khaki" (overlay only) • CheckBrush="Green" • CheckBoxBrush="Red" (inline only)		
PressedSelected • **SelectedPressedBackground="MediumTurquoise"** • SelectedForeground="Navy" • SelectedBackground="Khaki" (overlay only) • CheckBrush="Green" • CheckBoxBrush="Red" (inline only)		
Focused • **FocusBorderBrush="Crimson"** • **FocusSecondaryBorderBrush="Gold"** • CheckBoxBrush="Red"		

ListViewItemPresenter has other brush properties for data placeholders and drag states. If you use incremental loading or drag and drop in your list view, you should consider whether you need to also modify these additional brush properties. See the ListViewItemPresenter class for the complete list of properties you can modify.

Expanded XAML item templates

If you need to make more modifications than what is allowed by the **ListViewItemPresenter** properties - if you need to change the position of the check box, for example - you can use the *ListViewItemExpanded* or *GridViewItemExpanded*

templates. These templates are included with the default styles in generic.xaml. They follow the standard XAML pattern of building all the visuals from individual UIElements.

As mentioned previously, the number of UIElements in an item template has a significant impact on the performance of your list view. Replacing ListViewItemPresenter with the expanded XAML templates greatly increases the element count, and is not recommended when your list view will show a large number of items or when performance is a concern.

> **Note**
>
> **ListViewItemPresenter** is supported only when the list view's ItemsPanel is an ItemsWrapGrid or ItemsStackPanel. If you change the ItemsPanel to use VariableSizedWrapGrid, WrapGrid, or StackPanel, then the item template is automatically switched to the expanded XAML template. For more info, see ListView and GridView UI optimization.

To customize an expanded XAML template, you need to make a copy of it in your app, and set the **ItemContainerStyle** property to your copy.

To copy the expanded template

1. Set the ItemContainerStyle property as shown here for your ListView or GridView.

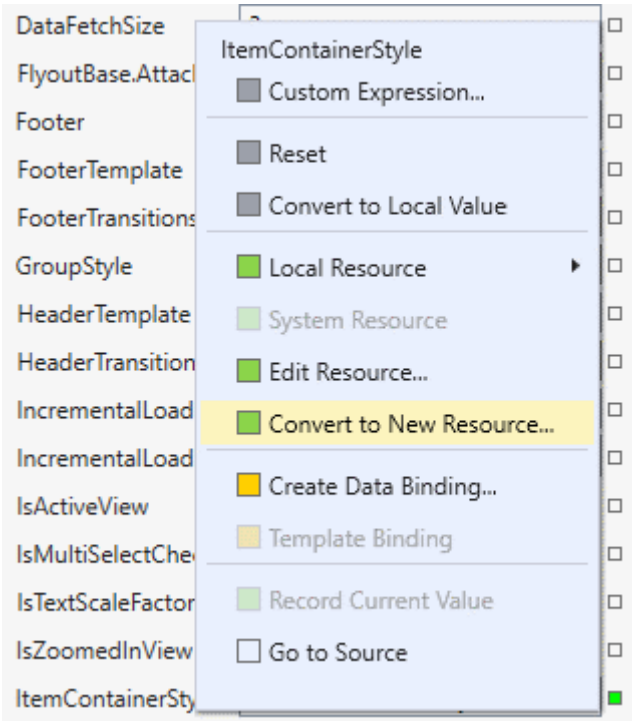
   XAML

   ```
   <ListView ItemContainerStyle="{StaticResource ListViewItemExpanded}"/>
   <GridView ItemContainerStyle="{StaticResource GridViewItemExpanded}"/>
   ```

2. In the Visual Studio Properties pane, expand the Miscellaneous section and find the ItemContainerStyle property. (Make sure the ListView or GridView is selected.)

3. Click the property marker for the ItemContainerStyle property. (It's the small box next to the TextBox. It's coloreed green to show that it's set to a StaticResource.) The property menu opens.

4. In the property menu, click **Convert to New Resource**.



5. In the Create Style Resource dialog, enter a name for the resource and click OK.

A copy of the expanded template from generic.xaml is created in your app, which you can modify as needed.

Related articles

- Lists
- ListView and GridView

Data template selection: Styling items based on their properties

Article • 06/09/2022

The customized design of collections controls are managed by a DataTemplate. Data templates define how each item should be laid out and styled, and that markup is applied to every item in the collection. This article explains how to use a DataTemplateSelector to apply different data templates on a collection and select which data template to use, based on certain item properties or values of your choosing.

> **Important APIs**: DataTemplateSelector, DataTemplate

DataTemplateSelector is a class that enables custom template selection logic. It lets you define rules that specify which data template to use for certain items in a collection. To implement this logic, you create a subclass of DataTemplateSelector in your code-behind and define the logic that determines which data template to use for which category of items (for example, items of a certain type or items with a certain property value, etc). You declare an instance of this class in the Resources section of your XAML file, along with the definitions of the data templates you'll be using. You identify these resources with an `x:Key` value, which lets you reference them in your XAML.

Prerequisites

- How to use and create a collection control, such as a ListView or GridView.
- How to customize the look of your items using a DataTemplate.

When not to use a DataTemplateSelector

Generally, you should not give every item in a ListView or GridView a completely different layout/style - this would be poor use of a DataTemplateSelector, and negatively impact performance.

Certain elements of the visual display of a list item can be controlled by using just one data template, through binding certain properties. For example, items can each have a different icon by binding to an icon source property in the data template, and giving each item a different value for that icon source property. This would achieve better performance than using a DataTemplateSelector.

When to use a DataTemplateSelector

You should create a DataTemplateSelector when you want to use multiple data templates in one collection control. A `DataTemplateSelector` gives you the flexibility to make certain items stand out, while still keeping items in a similar layout. There are many use cases in which a DataTemplateSelector is helpful, and a few scenarios in which it is better to re-think the control and strategy that you're using.

Collection controls are typically bound to a collection of items that are all of one type. However, even though items may be of the same type, they may have different values for certain properties or represent different meanings. Certain items also may be more important than others, or one item may be particularly important or different and has a need to visually stand out. In these situations, a DataTemplateSelector will be very helpful.

You can also bind to a collection that contains different types of items - the bound collection can have a mix of strings, ints, custom class objects, and more. This makes DataTemplateSelector especially useful as it can assign different data templates based on the object type of an item.

Here are some examples of when you might use a data template selector:

- Representing different levels of employees within a ListView - each type/level of employee may need a different color background to be easily distinguishable.
- Representing sale items in a product gallery that uses a GridView - a sale item may need a red background, or a different color font to make it stand out from regular-priced items.
- Representing winners/top photos in a photo gallery using FlipView.
- Needing to represent negative/positive numbers in a ListView differently, or short strings/long strings.

Create a DataTemplateSelector

When you create a data template selector, you define the template selection logic in your code, and you define the data templates in your XAML.

Code-behind component

To use a data template selector, you first create a subclass of DataTemplateSelector (a class that derives from it) in your code-behind. In your class, you declare each template

as a property of the class. Then, you override the SelectTemplateCore method to include your own template selection logic.

Here is an example of a simple `DataTemplateSelector` subclass called `MyDataTemplateSelector`.

C#

```
public class MyDataTemplateSelector : DataTemplateSelector
{
    public DataTemplate Normal { get; set; }
    public DataTemplate Accent { get; set; }

    protected override DataTemplate SelectTemplateCore(object item)
    {
        if ((int)item % 2 == 0)
        {
            return Normal;
        }
        else
        {
            return Accent;
        }
    }
}
```

The `MyDataTemplateSelector` class derives from the `DataTemplateSelector` class and first defines two DataTemplate objects: `Normal` and `Accent`. These are empty declarations for now, but will be "filled in" with markup in the XAML file.

The `SelectTemplateCore` method takes in an item object (i.e. each collection item), and is overridden with rules on which `DataTemplate` to return under which circumstances. In this case, if the item is an odd number, it receives the `Accent` data template, while if it is an even number it receives the `Normal` data template.

XAML component

Second, you must create an instance of this new `MyDataTemplateSelector` class in the Resources section of your XAML file. All resources require an `x:Key`, which you use to bind it to the `ItemTemplateSelector` property of your collection control (in a later step). You also create two instances of `DataTemplate` objects and define their layout in the resources section. You assign these data templates to the `Accent` and `Normal` properties you declared in the `MyDataTemplateSelector` class.

Here's an example of the XAML resources and markup needed:

XAML

```
<Page.Resources>

<DataTemplate x:Key="NormalItemTemplate" x:DataType="x:Int32">
    <Button HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
Background="{ThemeResource SystemChromeLowColor}">
        <TextBlock Text="{x:Bind}" />
    </Button>
</DataTemplate>

<DataTemplate x:Key="AccentItemTemplate" x:DataType="x:Int32">
    <Button HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
Background="{ThemeResource SystemAccentColor}">
        <TextBlock Text="{x:Bind}" />
    </Button>
</DataTemplate>

<l:MyDataTemplateSelector x:Key="MyDataTemplateSelector"
    Normal="{StaticResource NormalItemTemplate}"
    Accent="{StaticResource AccentItemTemplate}"/>

</Page.Resources>
```

As you can see above, the two data templates `Normal` and `Accent` are defined - they both display items as buttons, however the `Accent` data template uses an accent color brush for the background, while the `Normal` data template uses a gray color brush (`SystemChromeLowColor`). These two data templates are then assigned to the `Normal` and `Accent` DataTemplate objects that are attributes of the MyDataTemplateSelector class, created in the C# code-behind.

The last step is to bind your `DataTemplateSelector` to the `ItemTemplateSelector` property of your collections control (in this case, a ListView). This replaces the need for assigning a value to the `ItemTemplate` property.

XAML

```
<ListView x:Name = "TestListView"
          ItemsSource = "{x:Bind NumbersList}"
          ItemTemplateSelector = "{StaticResource MyDataTemplateSelector}">
</ListView>
```

Once your code compiles, each collection item will run through the overridden `SelectTemplateCore` method in `MyDataTemplateSelector`, and will be rendered with the appropriate DataTemplate.

> **ⓘ Important**
>
> When using `DataTemplateSelector` with an **ItemsRepeater**, you bind the `DataTemplateSelector` to the `ItemTemplate` property. `ItemsRepeater` doesn't have an `ItemTemplateSelector` property.

DataTemplateSelector performance considerations

When you use a ListView or GridView with a large data collection, scrolling and panning performance can be a concern. To keep large collections performing well, there are some steps you can take to improve the performance of your data templates. These are described in more detail in ListView and GridView UI optimization.

- *Element reduction per item* - Keep the number of UI elements in a data template to a reasonable minimum.
- Container-recycling with heterogeneous collections
  - Use *the ChoosingItemContainer event* - This event is a high-performance way to use different data templates for different items. To achieve best performance, you should optimize caching and selecting data templates for your specific data.
  - Use an *item template selector* - An item template selector (`DataTemplateSelector`) should be avoided in some instances due to its impact on performance.

Item templates for list view

Article • 06/09/2022

This section contains item templates that you can use with a **ListView** control. Use these templates to get the look of common app types.

To demonstrate data binding, these templates bind **ListViewItems** to the example Recording class from the data binding overview.

> ⓘ **Note**
>
> Currently, when a **DataTemplate** contains multiple controls (for example, more than a single **TextBlock**), the default accessible name for screenreaders comes from .ToString() on the item. As a convenience you can instead set the AutomationProperties.Name on the root element of the **DataTemplate**. For more on accessibility, see Accessibility overview.

Single line list item

Use this template to display a list of items with an image and a single line of text.

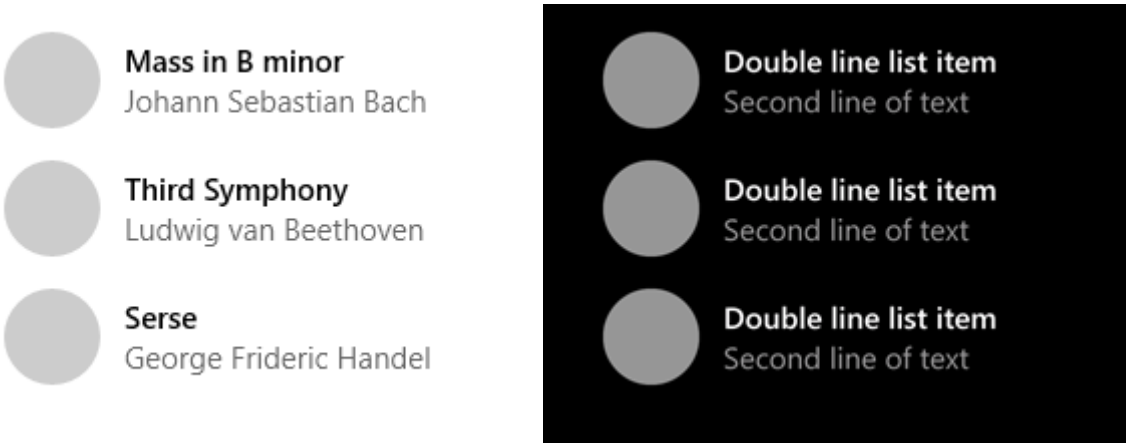
XAML

```
<ListView ItemsSource="{x:Bind ViewModel.Recordings}">
    <ListView.ItemTemplate>
        <DataTemplate x:Name="SingleLineDataTemplate"
x:DataType="local:Recording">
            <StackPanel Orientation="Horizontal" Height="44" Padding="12"
AutomationProperties.Name="{x:Bind CompositionName}">
                <Image Source="Placeholder.png" Height="16" Width="16"
VerticalAlignment="Center"/>
                <TextBlock Text="{x:Bind CompositionName}"
VerticalAlignment="Center" Style="{ThemeResource BaseTextBlockStyle}"
Foreground="{ThemeResource SystemControlPageTextBaseHighBrush}"
Margin="12,0,0,0"/>
            </StackPanel>
```

```
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Double line list item

Use this template to display a list of items with an image and two lines of text.

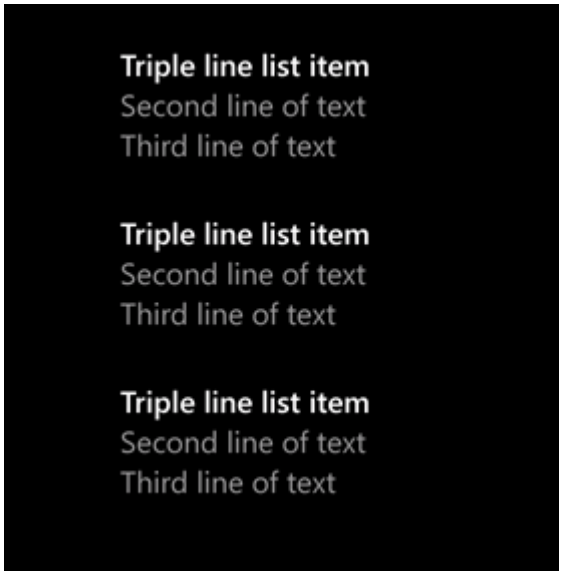


XAML

```
<ListView ItemsSource="{x:Bind ViewModel.Recordings}">
    <ListView.ItemTemplate>
        <DataTemplate x:Name="DoubleLineDataTemplate"
x:DataType="local:Recording">
            <StackPanel Orientation="Horizontal" Height="64"
AutomationProperties.Name="{x:Bind CompositionName}">
                <Ellipse Height="48" Width="48" VerticalAlignment="Center">
                    <Ellipse.Fill>
                        <ImageBrush ImageSource="Placeholder.png"/>
                    </Ellipse.Fill>
                </Ellipse>
                <StackPanel Orientation="Vertical"
VerticalAlignment="Center" Margin="12,0,0,0">
                    <TextBlock Text="{x:Bind CompositionName}"  Style="
{ThemeResource BaseTextBlockStyle}" Foreground="{ThemeResource
SystemControlPageTextBaseHighBrush}" />
                    <TextBlock Text="{x:Bind ArtistName}" Style="
{ThemeResource BodyTextBlockStyle}" Foreground="{ThemeResource
SystemControlPageTextBaseMediumBrush}"/>
                </StackPanel>
            </StackPanel>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Triple line list item

Use this template to display a list of items with three lines of text.

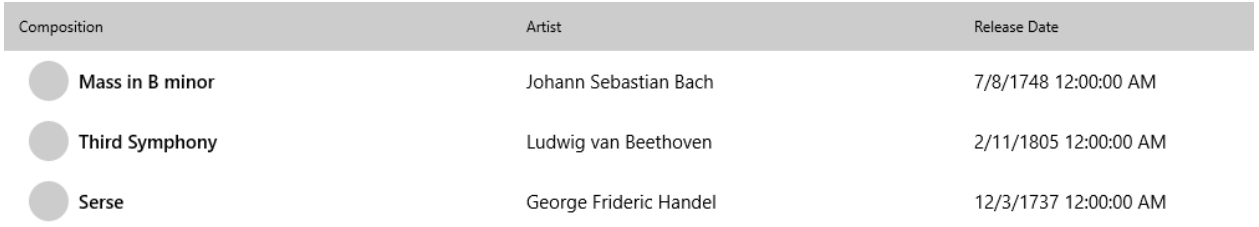


XAML

```
<ListView ItemsSource="{x:Bind ViewModel.Recordings}">
    <ListView.ItemTemplate>
        <DataTemplate x:Name="TripleLineDataTemplate"
x:DataType="local:Recording">
            <StackPanel Height="84" Padding="20" AutomationProperties.Name="
{x:Bind CompositionName}">
                <TextBlock Text="{x:Bind CompositionName}"  Style="
{ThemeResource BaseTextBlockStyle}" Margin="0,4,0,0"/>
                <TextBlock Text="{x:Bind ArtistName}" Style="{ThemeResource
CaptionTextBlockStyle}" Opacity=".8" Margin="0,4,0,0"/>
                <TextBlock Text="{x:Bind ReleaseDateTime}" Style="
{ThemeResource CaptionTextBlockStyle}" Opacity=".6" Margin="0,4,0,0"/>
            </StackPanel>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Table list item

Use this template to display a list of items with text in defined columns.



XAML

```
<ListView  ItemsSource="{x:Bind ViewModel.Recordings}">
    <ListView.HeaderTemplate>
        <DataTemplate>
            <Grid Padding="12" Background="{ThemeResource
SystemBaseLowColor}">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="408"/>
                    <ColumnDefinition Width="360"/>
                    <ColumnDefinition Width="360"/>
                </Grid.ColumnDefinitions>
                <TextBlock Text="Composition" Style="{ThemeResource
CaptionTextBlockStyle}"/>
                <TextBlock Grid.Column="1" Text="Artist" Style="
{ThemeResource CaptionTextBlockStyle}"/>
                <TextBlock Grid.Column="2" Text="Release Date" Style="
{ThemeResource CaptionTextBlockStyle}"/>
            </Grid>
        </DataTemplate>
    </ListView.HeaderTemplate>
    <ListView.ItemTemplate>
        <DataTemplate x:Name="TableDataTemplate"
x:DataType="local:Recording">
            <Grid Height="48" AutomationProperties.Name="{x:Bind
CompositionName}">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="48"/>
                    <ColumnDefinition Width="360"/>
                    <ColumnDefinition Width="360"/>
                    <ColumnDefinition Width="360"/>
                </Grid.ColumnDefinitions>
                <Ellipse Height="32" Width="32" VerticalAlignment="Center">
                    <Ellipse.Fill>
                        <ImageBrush ImageSource="Placeholder.png"/>
                    </Ellipse.Fill>
                </Ellipse>
                <TextBlock Grid.Column="1" VerticalAlignment="Center"
Style="{ThemeResource BaseTextBlockStyle}" Text="{x:Bind CompositionName}"
/>
                <TextBlock Grid.Column="2" VerticalAlignment="Center" Text="
{x:Bind ArtistName}"/>
                <TextBlock Grid.Column="3" VerticalAlignment="Center" Text="
{x:Bind ReleaseDateTime}"/>
            </Grid>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

Related articles

- ListView class
- Data binding overview

- Accessibility overview
- ListView and GridView sample
- Thumbnail images

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Item templates for grid view

Article • 06/09/2022

This section contains item templates that you can use with a **GridView** control. Use these templates to get the look of common app types.

To demonstrate data binding, these templates bind **GridViewItems** to the example Recording class from the data binding overview.

> **Note**
>
> Currently, when a **DataTemplate** contains multiple controls (for example, more than a single **TextBlock**), the default accessible name for screenreaders comes from .ToString() on the item. As a convenience you can instead set the AutomationProperties.Name on the root element of the **DataTemplate**. For more on accessibility, see Accessibility overview.

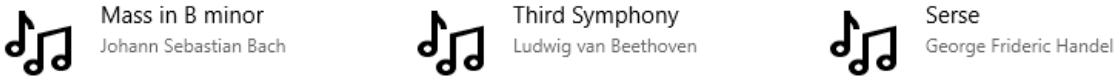
Icon and text

Use these templates to display a collection of apps in a grid with an icon and text.

XAML

```
<GridView ItemsSource="{x:Bind ViewModel.Recordings}">
    <GridView.ItemTemplate>
        <DataTemplate x:Name="IconTextTemplate"
x:DataType="local:Recording">
            <StackPanel Width="264" Height="48" Padding="12"
Orientation="Horizontal" AutomationProperties.Name="{x:Bind
CompositionName}">
                <SymbolIcon Symbol="Audio" VerticalAlignment="Top"/>
                <TextBlock Margin="12,0,0,0" Width="204" Text="{x:Bind
CompositionName}"/>
            </StackPanel>
        </DataTemplate>
    </GridView.ItemTemplate>
    <GridView.ItemsPanel>
        <ItemsPanelTemplate>
            <ItemsWrapGrid MaximumRowsOrColumns="4" Orientation="Horizontal"
HorizontalAlignment="Center"/>
        </ItemsPanelTemplate>
```

```
    </GridView.ItemsPanel>
</GridView>
```

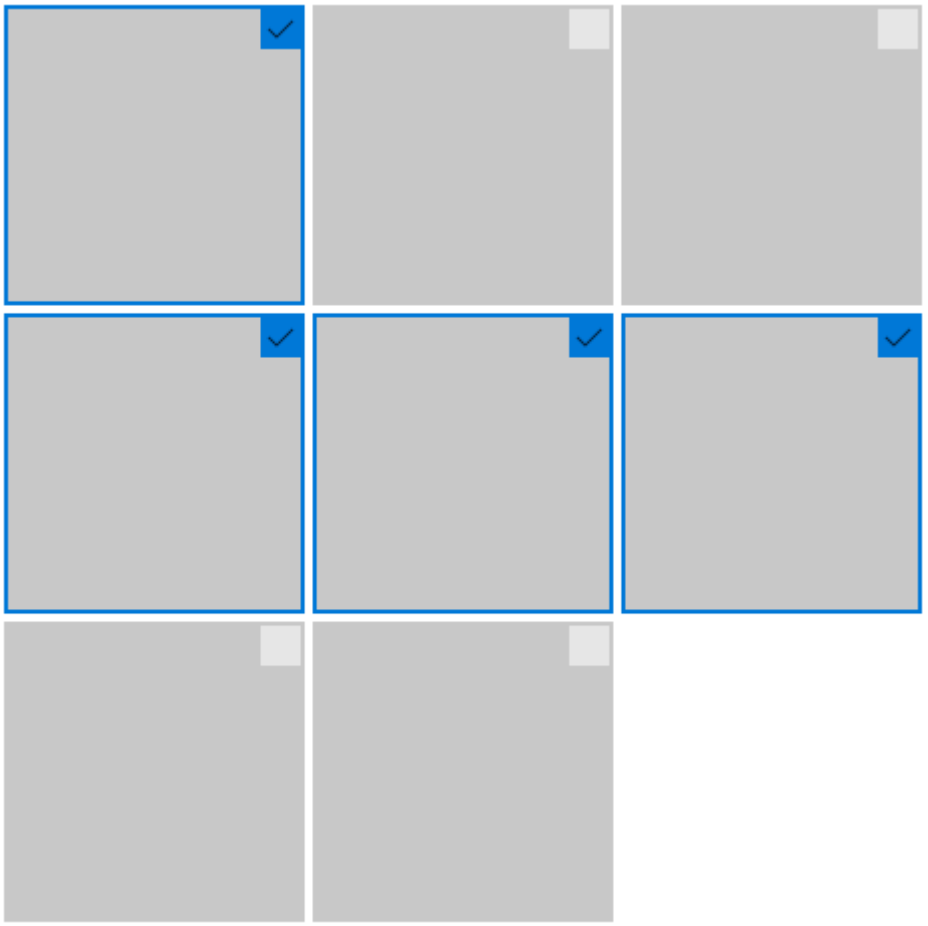


XAML

```
<GridView ItemsSource="{x:Bind ViewModel.Recordings}">
    <GridView.ItemTemplate>
        <DataTemplate x:Name="IconTextTemplate2"
x:DataType="local:Recording">
            <StackPanel Width="264" Height="120" Padding="12"
Orientation="Horizontal" AutomationProperties.Name="{x:Bind
CompositionName}">
                <FontIcon Margin="0,6,0,0" FontSize="48" FontFamily="Segoe
MDL2 Assets" FontWeight="Bold" Glyph="&#xE8D6;" VerticalAlignment="Top"/>
                <StackPanel Margin="16,1,0,0">
                    <TextBlock Width="176" Margin="0,0,0,2"
TextWrapping="WrapWholeWords" TextTrimming="Clip" Text="{x:Bind
CompositionName}"/>
                    <TextBlock Width="176" Height="48" Style="{ThemeResource
CaptionTextBlockStyle}" Foreground="{ThemeResource
SystemControlPageTextBaseMediumBrush}" TextWrapping="WrapWholeWords"
TextTrimming="Clip" Text="{x:Bind ArtistName}" />
                </StackPanel>
            </StackPanel>
        </DataTemplate>
    </GridView.ItemTemplate>
    <GridView.ItemsPanel>
        <ItemsPanelTemplate>
            <ItemsWrapGrid MaximumRowsOrColumns="4" Orientation="Horizontal"
HorizontalAlignment="Center" Margin="40,0"/>
        </ItemsPanelTemplate>
    </GridView.ItemsPanel>
</GridView>
```

Image gallery

Use this template to display a collection of images in a grid with multi-select mode.

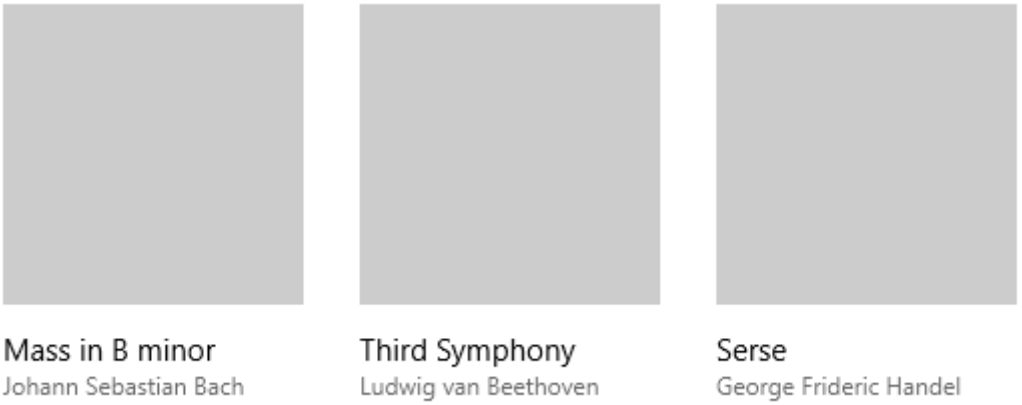


XAML

```
<GridView SelectionMode="Multiple">
    <GridView.ItemTemplate>
        <DataTemplate x:Name="ImageGalleryDataTemplate">
            <Image Source="Placeholder.png" Height="180" Width="180"
Stretch="UniformToFill"/>
        </DataTemplate>
    </GridView.ItemTemplate>
    <GridView.ItemsPanel>
        <ItemsPanelTemplate>
            <ItemsWrapGrid MaximumRowsOrColumns="3"
Orientation="Horizontal"/>
        </ItemsPanelTemplate>
    </GridView.ItemsPanel>
</GridView>
```

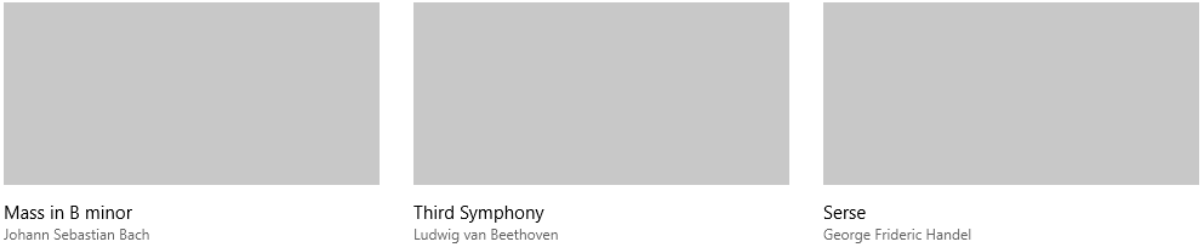
Image and text

Use these templates to display a media collection with text underneath.



XAML

```
<GridView ItemsSource="{x:Bind ViewModel.Recordings}">
    <GridView.ItemTemplate>
        <DataTemplate x:Name="ImageTextDataTemplate"
x:DataType="local:Recording">
            <StackPanel Height="280" Width="180" Margin="12"
AutomationProperties.Name="{x:Bind CompositionName}">
                <Image Source="Placeholder.png" Height="180" Width="180"
Stretch="UniformToFill"/>
                <StackPanel Margin="0,12">
                    <TextBlock Text="{x:Bind CompositionName}"/>
                    <TextBlock Text="{x:Bind ArtistName}" Style="
{ThemeResource CaptionTextBlockStyle}" Foreground="{ThemeResource
SystemControlPageTextBaseMediumBrush}"/>
                </StackPanel>
            </StackPanel>
        </DataTemplate>
    </GridView.ItemTemplate>
    <GridView.ItemsPanel>
        <ItemsPanelTemplate>
            <ItemsWrapGrid MaximumRowsOrColumns="10"
Orientation="Horizontal"/>
        </ItemsPanelTemplate>
    </GridView.ItemsPanel>
</GridView>
```

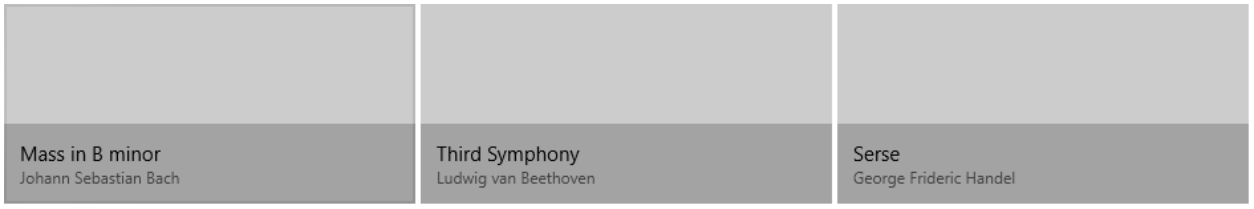


XAML

```
<GridView ItemsSource="{x:Bind ViewModel.Recordings}">
    <GridView.ItemTemplate>
        <DataTemplate x:Name="ImageTextDataTemplate2"
x:DataType="local:Recording">
            <StackPanel Height="280" Width="320" Margin="12"
AutomationProperties.Name="{x:Bind CompositionName}">
                <Image Source="Placeholder.png" Height="180" Width="320"
Stretch="UniformToFill"/>
                <StackPanel Margin="0,12">
                    <TextBlock Text="{x:Bind CompositionName}"/>
                    <TextBlock Text="{x:Bind ArtistName}" Style="
{ThemeResource CaptionTextBlockStyle}" Foreground="{ThemeResource
SystemControlPageTextBaseMediumBrush}"/>
                </StackPanel>
            </StackPanel>
        </DataTemplate>
    </GridView.ItemTemplate>
    <GridView.ItemsPanel>
        <ItemsPanelTemplate>
            <ItemsWrapGrid MaximumRowsOrColumns="4"
Orientation="Horizontal"/>
        </ItemsPanelTemplate>
    </GridView.ItemsPanel>
</GridView>
```

Image with text overlay

Use this template to display a media collection with text overlay.



XAML

```
<GridView ItemsSource="{x:Bind ViewModel.Recordings}">
    <GridView.ItemTemplate>
        <DataTemplate x:Name="ImageOverlayDataTemplate"
x:DataType="local:Recording">
            <Grid Height="180" Width="320" AutomationProperties.Name="
{x:Bind CompositionName}">
                <Image Source="Placeholder.png" Stretch="UniformToFill"/>
                <StackPanel Orientation="Vertical" Height="60"
VerticalAlignment="Bottom" Background="{ThemeResource SystemBaseLowColor}"
Padding="12">
                    <TextBlock Text="{x:Bind CompositionName}"/>
                    <TextBlock Text="{x:Bind ArtistName}" Style="
{ThemeResource CaptionTextBlockStyle}" Foreground="{ThemeResource
```

```
SystemControlPageTextBaseMediumBrush}"/>
                </StackPanel>
            </Grid>
        </DataTemplate>
    </GridView.ItemTemplate>
    <GridView.ItemsPanel>
        <ItemsPanelTemplate>
            <ItemsWrapGrid MaximumRowsOrColumns="4"
Orientation="Horizontal"/>
        </ItemsPanelTemplate>
    </GridView.ItemsPanel>
</GridView>
```

Related articles

- GridView class
- Data binding overview
- Accessibility overview
- ListView and GridView sample
- Thumbnail images

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

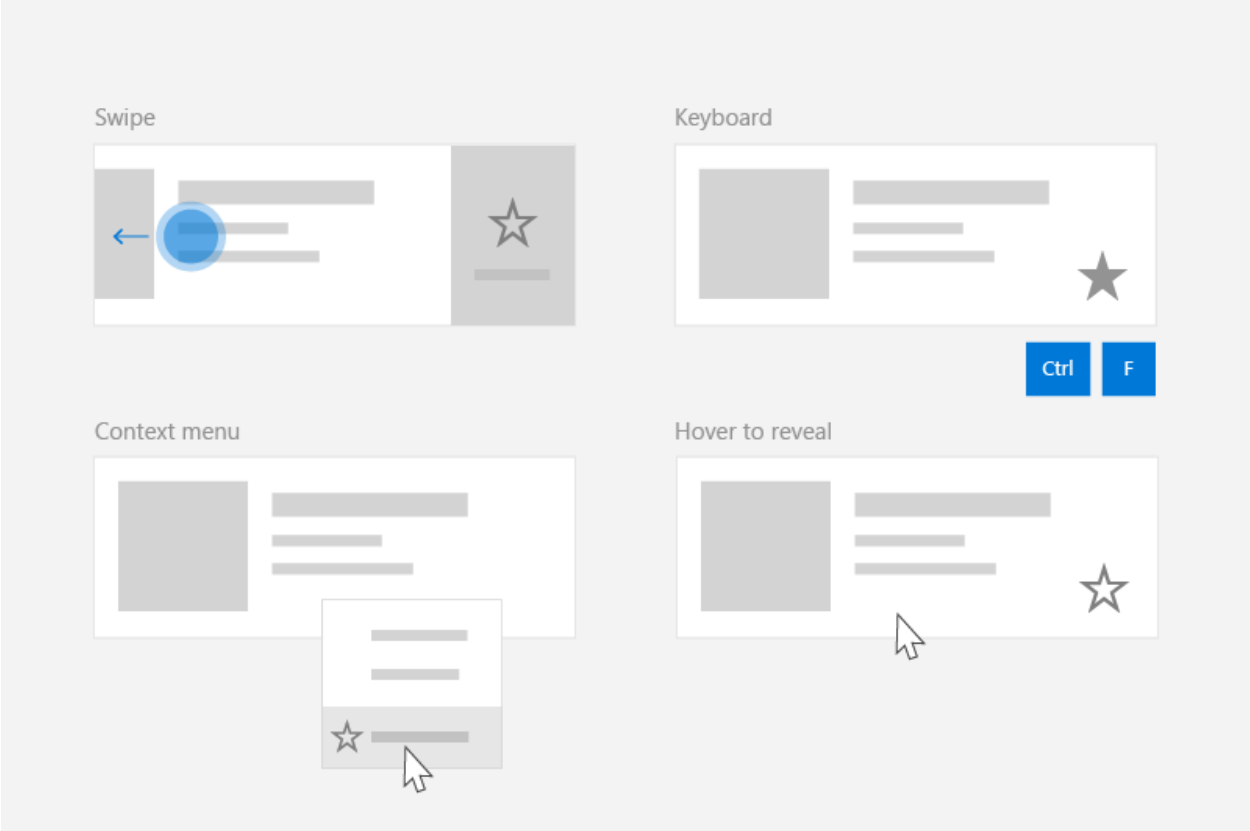
Provide product feedback

Contextual commanding for collections and lists

Article • 06/09/2022

Many apps contain collections of content in the form of lists, grids, and trees that users can manipulate. For example, users might be able to delete, rename, flag, or refresh items. This article shows you how to use contextual commands to implement these sorts of actions in a way that provides the best possible experience for all input types.

> **Important APIs**: ICommand interface, UIElement.ContextFlyout property, INotifyPropertyChanged interface



Creating commands for all input types

Because users can interact with a Windows app using a broad range of devices and inputs, your app should expose commands though both input-agnostic context menus and input-specific accelerators. Including both lets the user quickly invoke commands on content, regardless of input or device type.

This table shows some typical collection commands and ways to expose those commands.

Command	Input-agnostic	Mouse accelerator	Keyboard accelerator	Touch accelerator
Delete item	Context menu	Hover button	DEL key	Swipe to delete
Flag item	Context menu	Hover button	Ctrl+Shift+G	Swipe to flag
Refresh data	Context menu	N/A	F5 key	Pull to refresh
Favorite an item	Context menu	Hover button	F, Ctrl+S	Swipe to favorite

- **In general, you should make all commands for an item available in the item's context menu.** Context menus are accessible to users regardless of input type, and should contain all of the contextual commands that user can perform.
- **For frequently accessed commands, consider using input accelerators.** Input accelerators let the user perform actions quickly, based on their input device. Input accelerators include:
  - Swipe-to-action (touch accelerator)
  - Pull to refresh data (touch accelerator)
  - Keyboard shortcuts (keyboard accelerator)
  - Access keys (keyboard accelerator)
  - Mouse & Pen hover buttons (pointer accelerator)

> ⓘ Note
>
> Users should be able to access all commands from any type of device. For example, if your app's commands are only exposed through hover button pointer accelerators, touch users won't be able to access them. At a minimum, use a context menu to provide access to all commands.

Example: The PodcastObject data model

To demonstrate our commanding recommendations, this article creates a list of podcasts for a podcast app. The example code demonstrate how to enable the user to "favorite" a particular podcast from a list.

Here's the definition for the podcast object we'll be working with:

```
C#

public class PodcastObject : INotifyPropertyChanged
{
```

```
    // The title of the podcast
    public String Title { get; set; }

    // The podcast's description
    public String Description { get; set; }

    // Describes if the user has set this podcast as a favorite
    public bool IsFavorite
    {
        get
        {
            return _isFavorite;
        }
        set
        {
            _isFavorite = value;
            OnPropertyChanged("IsFavorite");
        }
    }
    private bool _isFavorite = false;

    public event PropertyChangedEventHandler PropertyChanged;

    private void OnPropertyChanged(String property)
    {
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(property));
    }
}
```

Notice that the PodcastObject implements INotifyPropertyChanged to respond to property changes when the user toggles the IsFavorite property.

Defining commands with the ICommand interface

The ICommand interface helps you to define a command that's available for multiple input types. For example, instead of writing the same code for a delete command in two different event handlers, one for when the user presses the Delete key and one for when the user right clicks "Delete" in a context menu, you can implement your delete logic once, as an ICommand, and then make it available to different input types.

We need to define the ICommand that represents the "Favorite" action. We will use the command's Execute method to favorite a podcast. The particular podcast will be provided to the execute method via the command's parameter, which can be bound using the CommandParameter property.

C#

```
public class FavoriteCommand: ICommand
{
    public event EventHandler CanExecuteChanged;

    public bool CanExecute(object parameter)
    {
        return true;
    }
    public void Execute(object parameter)
    {
        // Perform the logic to "favorite" an item.
        (parameter as PodcastObject).IsFavorite = true;
    }
}
```

To use the same command with multiple collections and elements, you can store the command as a resource on the page or on the app.

XAML

```
<Application.Resources>
    <local:FavoriteCommand x:Key="favoriteCommand" />
</Application.Resources>
```

To execute the command, you call its Execute method.

C#

```
// Favorite the item using the defined command
var favoriteCommand = Application.Current.Resources["favoriteCommand"] as
ICommand;
favoriteCommand.Execute(PodcastObject);
```

Creating a UserControl to respond to a variety of inputs

When you have a list of items and each of those items should respond to multiple inputs, you can simplify your code by defining a UserControl for the item and using it to define your items' context menu and event handlers.

To create a UserControl in Visual Studio:

1. In the Solution Explorer, right click the project. A context menu appears.

2. Select **Add** > **New Item...**
   The **Add New Item** dialog appears.
3. Select UserControl from the list of items. Give it the name you want and click **Add**.
   Visual Studio will generate a stub UserControl for you.

In our podcast example, each podcast will be displayed in a list, which will expose a variety of ways to "Favorite" a podcast. The user will be able to perform the following actions to "Favorite" the podcast:

- Invoke a context menu
- Perform keyboard shortcuts
- Show a hover button
- Perform a swipe gesture

In order to encapsulate these behaviors and use the FavoriteCommand, let's create a new UserControl named "PodcastUserControl" to represent a podcast in the list.

The PodcastUserControl displays the fields of the PodcastObject as TextBlocks, and responds to various user interactions. We will reference and expand upon the PodcastUserControl throughout this article.

PodcastUserControl.xaml

XAML

```
<UserControl
    x:Class="ContextCommanding.PodcastUserControl"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    IsTabStop="True" UseSystemFocusVisuals="True"
    >
    <Grid Margin="12,0,12,0">
        <StackPanel>
            <TextBlock Text="{x:Bind PodcastObject.Title, Mode=OneWay}"
Style="{StaticResource TitleTextBlockStyle}" />
            <TextBlock Text="{x:Bind PodcastObject.Description,
Mode=OneWay}" Style="{StaticResource SubtitleTextBlockStyle}" />
            <TextBlock Text="{x:Bind PodcastObject.IsFavorite, Mode=OneWay}"
Style="{StaticResource SubtitleTextBlockStyle}"/>
        </StackPanel>
    </Grid>
</UserControl>
```

PodcastUserControl.xaml.cs

C#

```
public sealed partial class PodcastUserControl : UserControl
{
    public static readonly DependencyProperty PodcastObjectProperty =
        DependencyProperty.Register(
            "PodcastObject",
            typeof(PodcastObject),
            typeof(PodcastUserControl),
            new PropertyMetadata(null));

    public PodcastObject PodcastObject
    {
        get { return (PodcastObject)GetValue(PodcastObjectProperty); }
        set { SetValue(PodcastObjectProperty, value); }
    }

    public PodcastUserControl()
    {
        this.InitializeComponent();

        // TODO: We will add event handlers here.
    }
}
```

Notice that the PodcastUserControl maintains a reference to the PodcastObject as a DependencyProperty. This enables us to bind PodcastObjects to the PodcastUserControl.

After you have generated some PodcastObjects, you can create a list of podcasts by binding the PodcastObjects to a ListView. The PodcastUserControl objects describe the visualization of the PodcastObjects, and are therefore set using the ListView's ItemTemplate.

MainPage.xaml

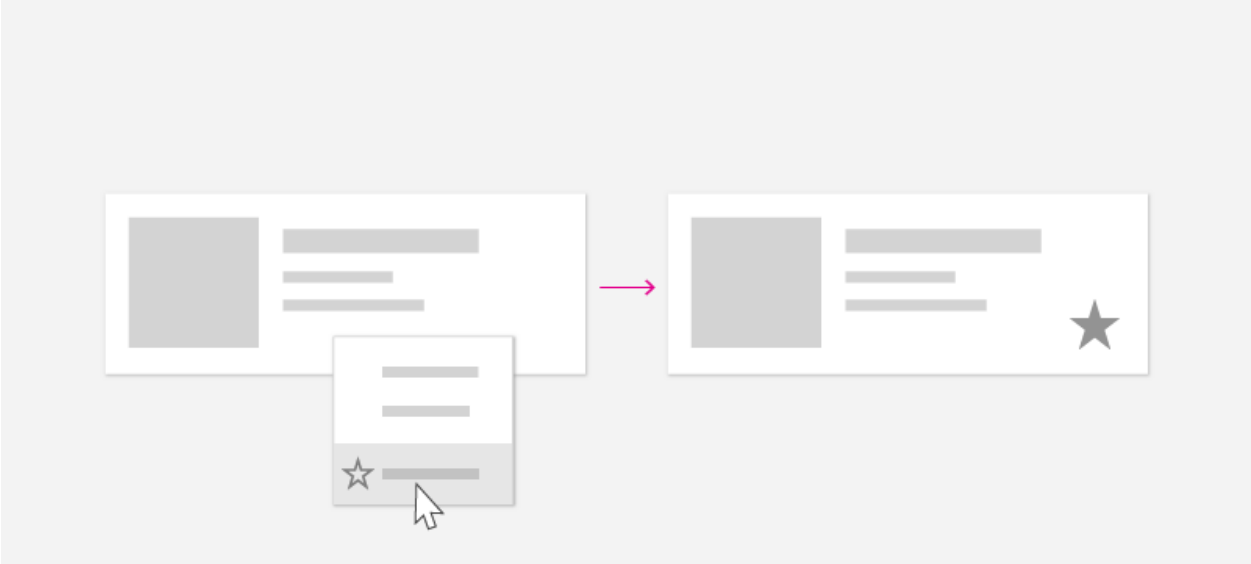
XAML

```
<ListView x:Name="ListOfPodcasts"
            ItemsSource="{x:Bind podcasts}">
    <ListView.ItemTemplate>
        <DataTemplate x:DataType="local:PodcastObject">
            <local:PodcastUserControl PodcastObject="{x:Bind Mode=OneWay}"
/>
        </DataTemplate>
    </ListView.ItemTemplate>
    <ListView.ItemContainerStyle>
        <!-- The PodcastUserControl will entirely fill the ListView item and
handle tabbing within itself. -->
        <Style TargetType="ListViewItem" BasedOn="{StaticResource
```

```
ListViewItemRevealStyle}">
            <Setter Property="HorizontalContentAlignment" Value="Stretch" />
            <Setter Property="Padding" Value="0"/>
            <Setter Property="IsTabStop" Value="False"/>
        </Style>
    </ListView.ItemContainerStyle>
</ListView>
```

Creating context menus

Context menus display a list of commands or options when the user requests them. Context menus provide contextual commands related to their attached element, and are generally reserved for secondary actions specific to that item.



The user can invoke context menus using these "context actions":

Input	Context action
Mouse	Right click
Keyboard	Shift+F10, Menu button
Touch	Long press on item
Pen	Barrel button press, long press on item
Gamepad	Menu button

Since the user can open a context menu regardless of input type, your context menu should contain all of the contextual commands available for the list item.

ContextFlyout

The ContextFlyout property, defined by the UIElement class, makes it easy to create a context menu that works with all input types. You provide a flyout representing your context menu using MenuFlyout or CommandBarFlyout, and when the user performs a "context action" as defined above, the MenuFlyout or CommandBarFlyout corresponding to the item will be displayed.

See menus and context menus for help identifying menu vs. context menu scenarios and guidance on when to use menu flyout vs. command bar flyout.

For this example, we will use MenuFlyout and will start by adding a ContextFlyout to the PodcastUserControl. The MenuFlyout specified as the ContextFlyout contains a single item to favorite a podcast. Notice that this MenuFlyoutItem uses the favoriteCommand defined above, with the CommandParameter bound to the PodcastObject.

PodcastUserControl.xaml

XAML

```
<UserControl>
    <UserControl.ContextFlyout>
        <MenuFlyout>
            <MenuFlyoutItem Text="Favorite" Command="{StaticResource
favoriteCommand}" CommandParameter="{x:Bind PodcastObject, Mode=OneWay}" />
        </MenuFlyout>
    </UserControl.ContextFlyout>
    <Grid Margin="12,0,12,0">
        <!-- ... -->
    </Grid>
</UserControl>
```

Note that you can also use the ContextRequested event to respond to context actions. The ContextRequested event will not fire if a ContextFlyout has been specified.

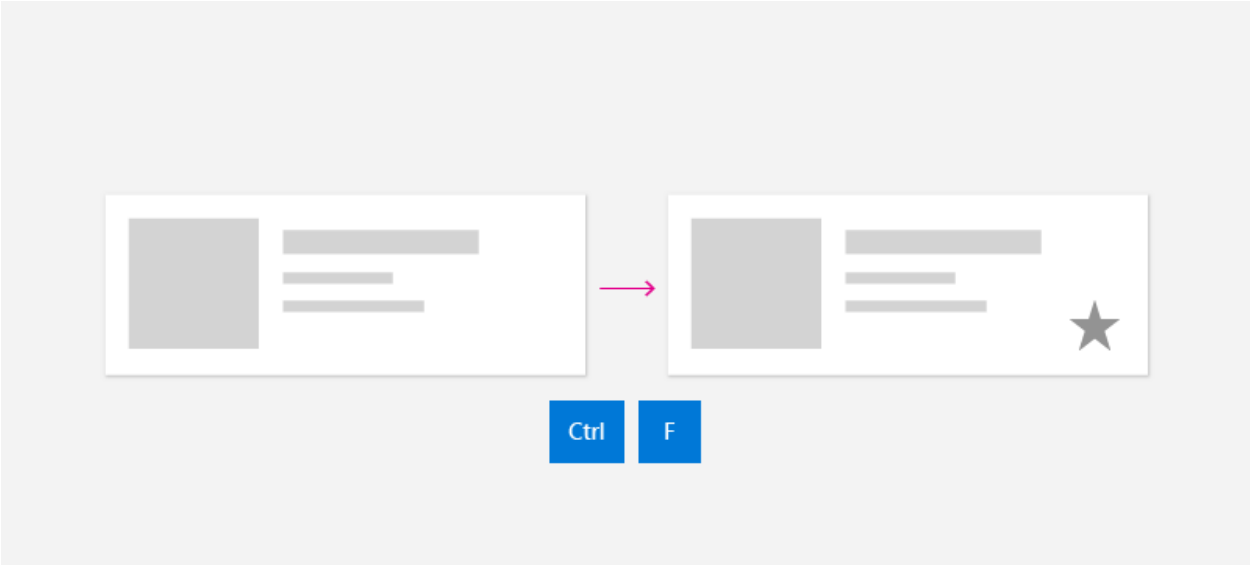
Creating input accelerators

Although each item in the collection should have a context menu containing all contextual commands, you might want to enable users to quickly perform a smaller set of frequently performed commands. For example, a mailing app may have secondary commands like Reply, Archive, Move to Folder, Set Flag, and Delete which appear in a context menu, but the most common commands are Delete and Flag. After you have identified which commands are most common, you can use input-based accelerators to make these commands easier for a user to perform.

In the podcast app, the frequently performed command is the "Favorite" command.

Keyboard accelerators

Shortcuts and direct key handling



Depending on the type of content, you may identify certain key combinations that should perform an action. In an email app, for example, the DEL key may be used to delete the email that is selected. In a podcast app, the Ctrl+S or F keys could favorite a podcast for later. Although some commands have common, well-known keyboard shortcuts like DEL to delete, other commands have app- or domain-specific shortcuts. Use well-known shortcuts if possible, or consider providing reminder text in a tooltip to teach the user about the shortcut command.

Your app can respond when the user presses a key using the KeyDown event. In general, users expect that the app will respond when they first press the key down, rather than waiting until they release the key.

This example walks through how to add the KeyDown handler to the PodcastUserControl to favorite a podcast when the user presses Ctrl+S or F. It uses the same command as before.

PodcastUserControl.xaml.cs

C#

```
// Respond to the F and Ctrl+S keys to favorite the focused item.
protected override void OnKeyDown(KeyRoutedEventArgs e)
{
    var ctrlState =
CoreWindow.GetForCurrentThread().GetKeyState(VirtualKey.Control);
    var isCtrlPressed = (ctrlState & CoreVirtualKeyStates.Down) ==
CoreVirtualKeyStates.Down || (ctrlState & CoreVirtualKeyStates.Locked) ==
CoreVirtualKeyStates.Locked;
```

```
    if (e.Key == Windows.System.VirtualKey.F || (e.Key ==
Windows.System.VirtualKey.S && isCtrlPressed))
    {
        // Favorite the item using the defined command
        var favoriteCommand =
Application.Current.Resources["favoriteCommand"] as ICommand;
        favoriteCommand.Execute(PodcastObject);
    }
}
```

Mouse accelerators

Users are familiar with right-click context menus, but you may wish to empower users to perform common commands using only a single click of the mouse. To enable this experience, you can include dedicated buttons on your collection item's canvas. To both empower users to act quickly using mouse, and to minimize visual clutter, you can choose to only reveal these buttons when the user has their pointer within a particular list item.

In this example, the Favorite command is represented by a button defined directly in the PodcastUserControl. Note that the button in this example uses the same command, FavoriteCommand, as before. To toggle visibility of this button, you can use the VisualStateManager to switch between visual states when the pointer enters and exits the control.

PodcastUserControl.xaml

XAML

```
<UserControl>
    <UserControl.ContextFlyout>
```

```
        <!-- ... -->
    </UserControl.ContextFlyout>
    <Grid Margin="12,0,12,0">
        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="HoveringStates">
                <VisualState x:Name="HoverButtonsShown">
                    <VisualState.Setters>
                        <Setter Target="hoverArea.Visibility"
Value="Visible" />
                    </VisualState.Setters>
                </VisualState>
                <VisualState x:Name="HoverButtonsHidden" />
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="Auto" />
        </Grid.ColumnDefinitions>
        <StackPanel>
            <TextBlock Text="{x:Bind PodcastObject.Title, Mode=OneWay}"
Style="{StaticResource TitleTextBlockStyle}" />
            <TextBlock Text="{x:Bind PodcastObject.Description,
Mode=OneWay}" Style="{StaticResource SubtitleTextBlockStyle}" />
            <TextBlock Text="{x:Bind PodcastObject.IsFavorite, Mode=OneWay}"
Style="{StaticResource SubtitleTextBlockStyle}"/>
        </StackPanel>
        <Grid Grid.Column="1" x:Name="hoverArea" Visibility="Collapsed"
VerticalAlignment="Stretch">
            <AppBarButton Icon="OutlineStar" Label="Favorite" Command="
{StaticResource favoriteCommand}" CommandParameter="{x:Bind PodcastObject,
Mode=OneWay}" IsTabStop="False" VerticalAlignment="Stretch"  />
        </Grid>
    </Grid>
</UserControl>
```

The hover buttons should appear and disappear when the mouse enters and exits the item. To respond to mouse events, you can use the PointerEntered and PointerExited events on the PodcastUserControl.

PodcastUserControl.xaml.cs

C#

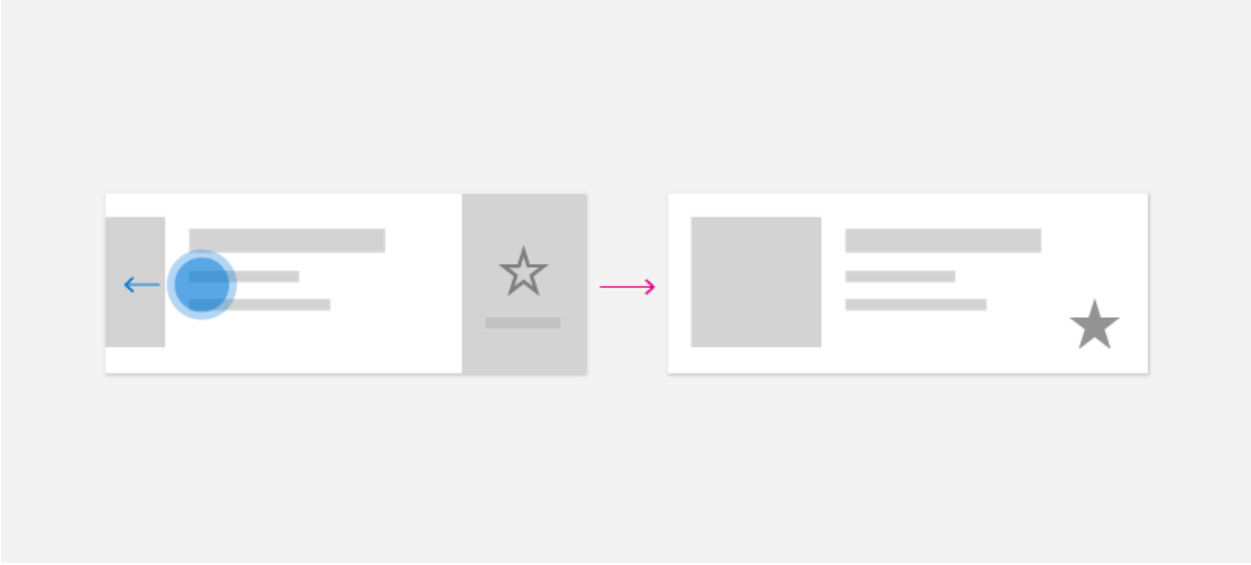
```
protected override void OnPointerEntered(PointerRoutedEventArgs e)
{
    base.OnPointerEntered(e);

    // Only show hover buttons when the user is using mouse or pen.
    if (e.Pointer.PointerDeviceType ==
Windows.Devices.Input.PointerDeviceType.Mouse || e.Pointer.PointerDeviceType
== Windows.Devices.Input.PointerDeviceType.Pen)
    {
```

```
            VisualStateManager.GoToState(this, "HoverButtonsShown", true);
        }
    }

    protected override void OnPointerExited(PointerRoutedEventArgs e)
    {
        base.OnPointerExited(e);

        VisualStateManager.GoToState(this, "HoverButtonsHidden", true);
    }
```

The buttons displayed in the hover state will only be accessible via the pointer input type. Because these buttons are limited to pointer input, you may choose to minimize or remove the padding around the icon of the button to optimize for pointer input. If you choose to do so, ensure that the button footprint is at least 20x20px to remain usable with pen and mouse.

Touch accelerators

Swipe



Swipe commanding is a touch accelerator that enables users on touch devices to perform common secondary actions using touch. Swipe empowers touch users to quickly and naturally interact with content, using common actions like Swipe-to-Delete or Swipe-to-Invoke. See the swipe commanding article to learn more.

In order to integrate swipe into your collection, you need two components: SwipeItems, which hosts the commands; and a SwipeControl, which wraps the item and allows for swipe interaction.

The SwipeItems can be defined as a Resource in the PodcastUserControl. In this example, SwipeItems contains a command to Favorite an item.

XAML

```
<UserControl.Resources>
    <SymbolIconSource x:Key="FavoriteIcon" Symbol="Favorite"/>
    <SwipeItems x:Key="RevealOtherCommands" Mode="Reveal">
        <SwipeItem IconSource="{StaticResource FavoriteIcon}"
Text="Favorite" Background="Yellow" Invoked="SwipeItem_Invoked"/>
    </SwipeItems>
</UserControl.Resources>
```

The SwipeControl wraps the item and allows the user to interact with it using the swipe gesture. Notice that the SwipeControl contains a reference to the SwipeItems as its RightItems. The Favorite item will show when the user swipes from right to left.

XAML

```
<SwipeControl x:Name="swipeContainer" RightItems="{StaticResource
RevealOtherCommands}">
   <!-- The visual state groups moved from the Grid to the SwipeControl,
since the SwipeControl wraps the Grid. -->
   <VisualStateManager.VisualStateGroups>
       <VisualStateGroup x:Name="HoveringStates">
           <VisualState x:Name="HoverButtonsShown">
               <VisualState.Setters>
                   <Setter Target="hoverArea.Visibility" Value="Visible" />
               </VisualState.Setters>
           </VisualState>
           <VisualState x:Name="HoverButtonsHidden" />
       </VisualStateGroup>
   </VisualStateManager.VisualStateGroups>
   <Grid Margin="12,0,12,0">
       <Grid.ColumnDefinitions>
           <ColumnDefinition Width="*" />
           <ColumnDefinition Width="Auto" />
       </Grid.ColumnDefinitions>
       <StackPanel>
           <TextBlock Text="{x:Bind PodcastObject.Title, Mode=OneWay}"
Style="{StaticResource TitleTextBlockStyle}" />
           <TextBlock Text="{x:Bind PodcastObject.Description, Mode=OneWay}"
Style="{StaticResource SubtitleTextBlockStyle}" />
           <TextBlock Text="{x:Bind PodcastObject.IsFavorite, Mode=OneWay}"
Style="{StaticResource SubtitleTextBlockStyle}"/>
       </StackPanel>
       <Grid Grid.Column="1" x:Name="hoverArea" Visibility="Collapsed"
VerticalAlignment="Stretch">
           <AppBarButton Icon="OutlineStar" Command="{StaticResource
favoriteCommand}" CommandParameter="{x:Bind PodcastObject, Mode=OneWay}"
IsTabStop="False" LabelPosition="Collapsed" VerticalAlignment="Stretch"  />
```

```
        </Grid>
    </Grid>
</SwipeControl>
```

When the user swipes to invoke the Favorite command, the Invoked method is called.

C#

```
private void SwipeItem_Invoked(SwipeItem sender, SwipeItemInvokedEventArgs
args)
{
    // Favorite the item using the defined command
    var favoriteCommand = Application.Current.Resources["favoriteCommand"]
as ICommand;
    favoriteCommand.Execute(PodcastObject);
}
```

Pull to refresh

Pull to refresh lets a user pull down on a collection of data using touch in order to retrieve more data. See the pull to refresh article to learn more.

Pen accelerators

The pen input type provides the precision of pointer input. Users can perform common actions such as opening context menus using pen-based accelerators. To open a context menu, users can tap the screen with the barrel button pressed, or long press on the content. Users can also use the pen to hover over content to get a deeper understanding of the UI like displaying tooltips, or to reveal secondary hover actions, similar to mouse.

To optimize your app for pen input, see the pen and stylus interaction article.

Recommendations

- Make sure that users can access all commands from all types of Windows devices.
- Include a context menu that provides access to all the commands available for a collection item.
- Provide input accelerators for frequently-used commands.
- Use the ICommand interface to implement commands.

Related topics

- Command design basics for Windows apps
- ICommand Interface
- Menus and context menus
- Swipe
- Pull to refresh
- Pen and stylus interaction
- Designing for Xbox and TV

Selection mode overview

Article • 06/24/2021

Selection mode lets users select and take action on a single item or on multiple items. It can be invoked through a context menu, by using CTRL+click or SHIFT+click on an item, or by rolling-over a target on an item in a gallery view. When selection mode is active, check boxes appear next to each list item, and actions can appear at the top or the bottom of the screen.

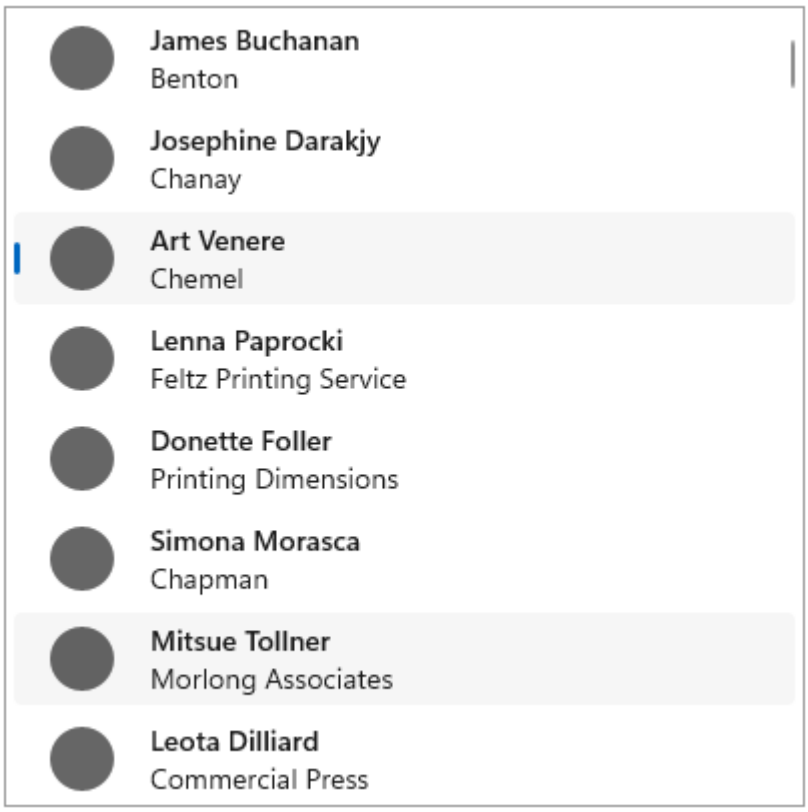
Different selection modes

There are three selection modes:

- Single: The user can select only one item at a time.
- Multiple: The user can select multiple items without using a modifier.
- Extended: The user can select multiple items with a modifier, such as holding down the SHIFT key.

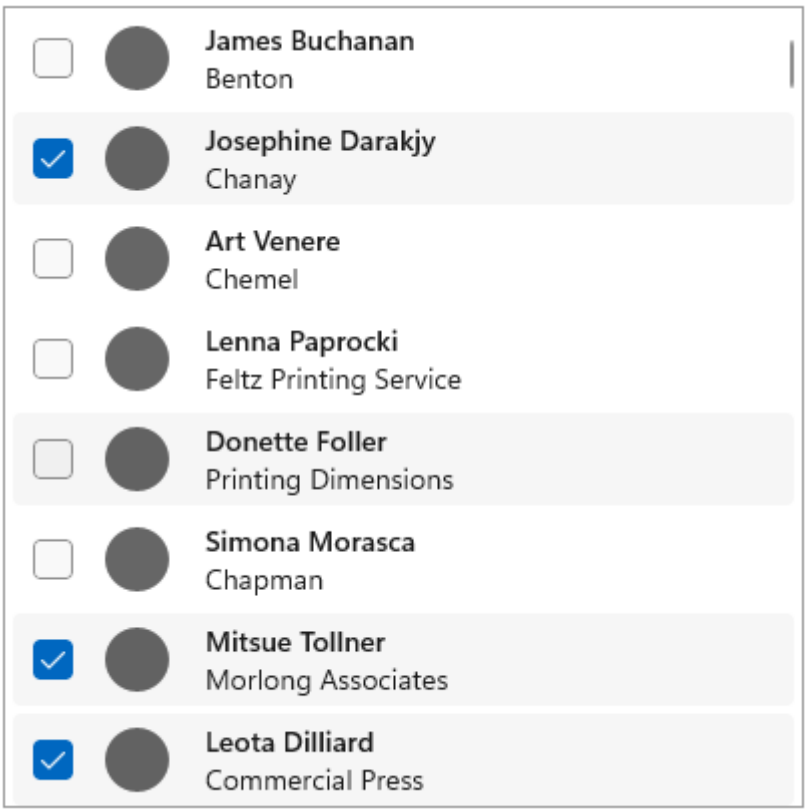
Selection mode examples

Here is a ListView with single selection mode enabled.



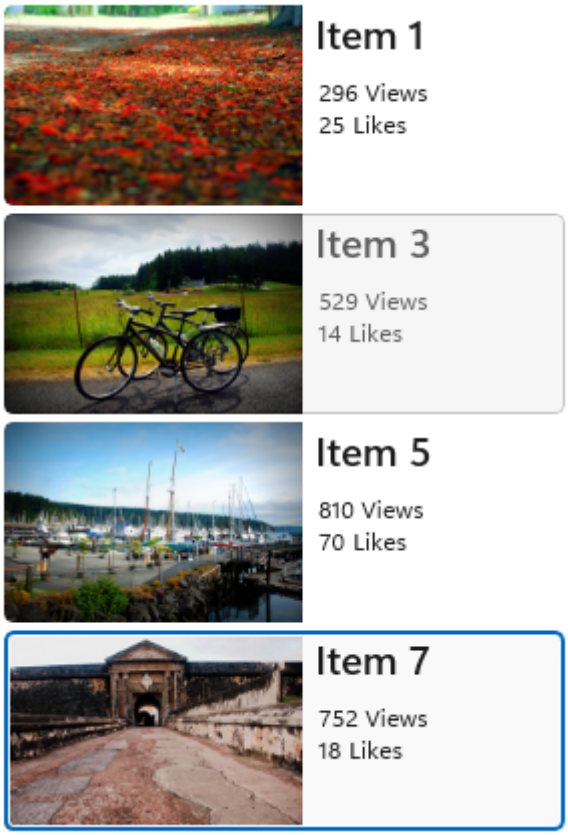
The item Art Venere is selected, and the item Mitsue Tollner is currently being hovered over.

Here is a ListView with multiple selection mode enabled:



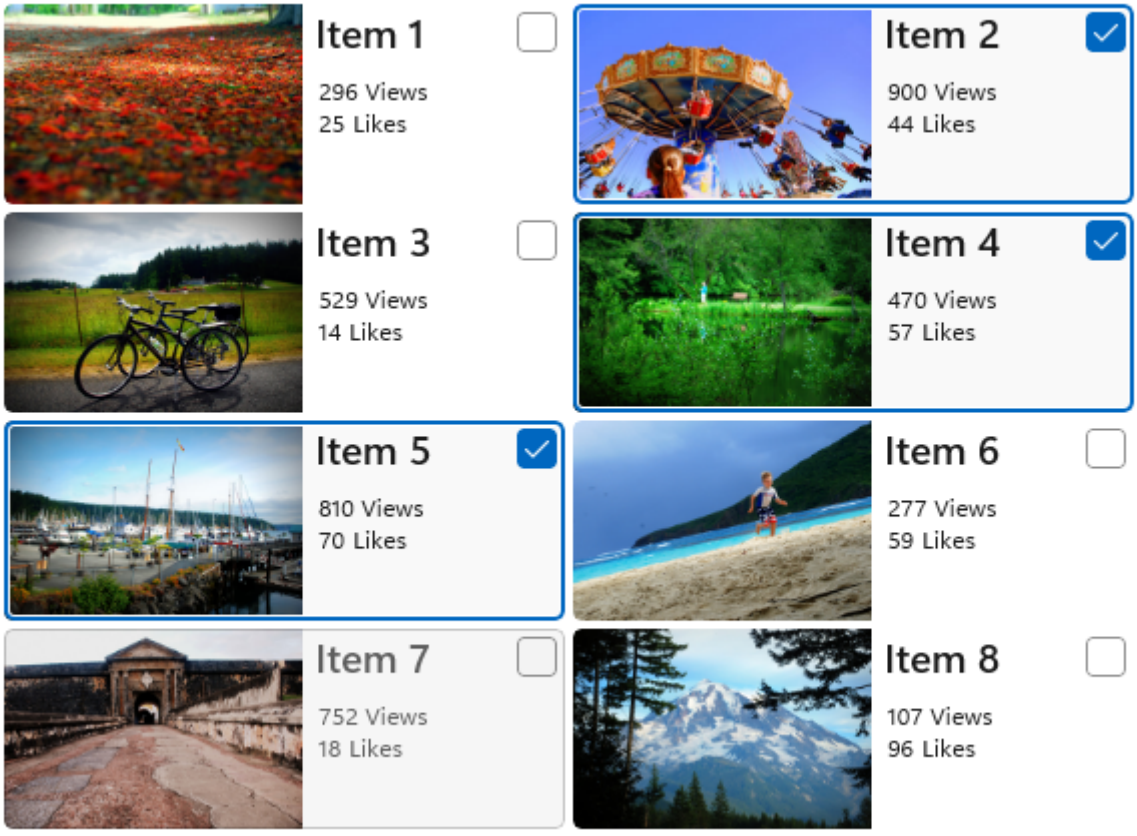
There are three items selected, and the item Donette Foller is being hovered over.

Here is a GridView with single selection mode enabled:



Item 7 is selected, and Item 3 is currently being hovered over.

Here is a GridView with multiple selection mode enabled:



Items 2, 4, and 5 are selected, and Item 7 is currently being hovered over.

Behavior and interaction

Tapping anywhere on an item selects it. Tapping on the command bar action affects all selected items. If no item is selected, command bar actions should be inactive, except for "Select All".

Selection mode doesn't have a light dismiss model; tapping outside of the frame in which selection mode is active won't cancel the mode. This is to prevent accidental deactivation of the mode. Clicking the back button dismisses the multi-select mode.

Show a visual confirmation when an action is selected. Consider displaying a confirmation dialog for certain actions, especially destructive actions such as delete.

Selection mode is confined to the page in which it is active, and can't affect any items outside of that page.

The entry point to selection mode should be juxtaposed against the content it affects.

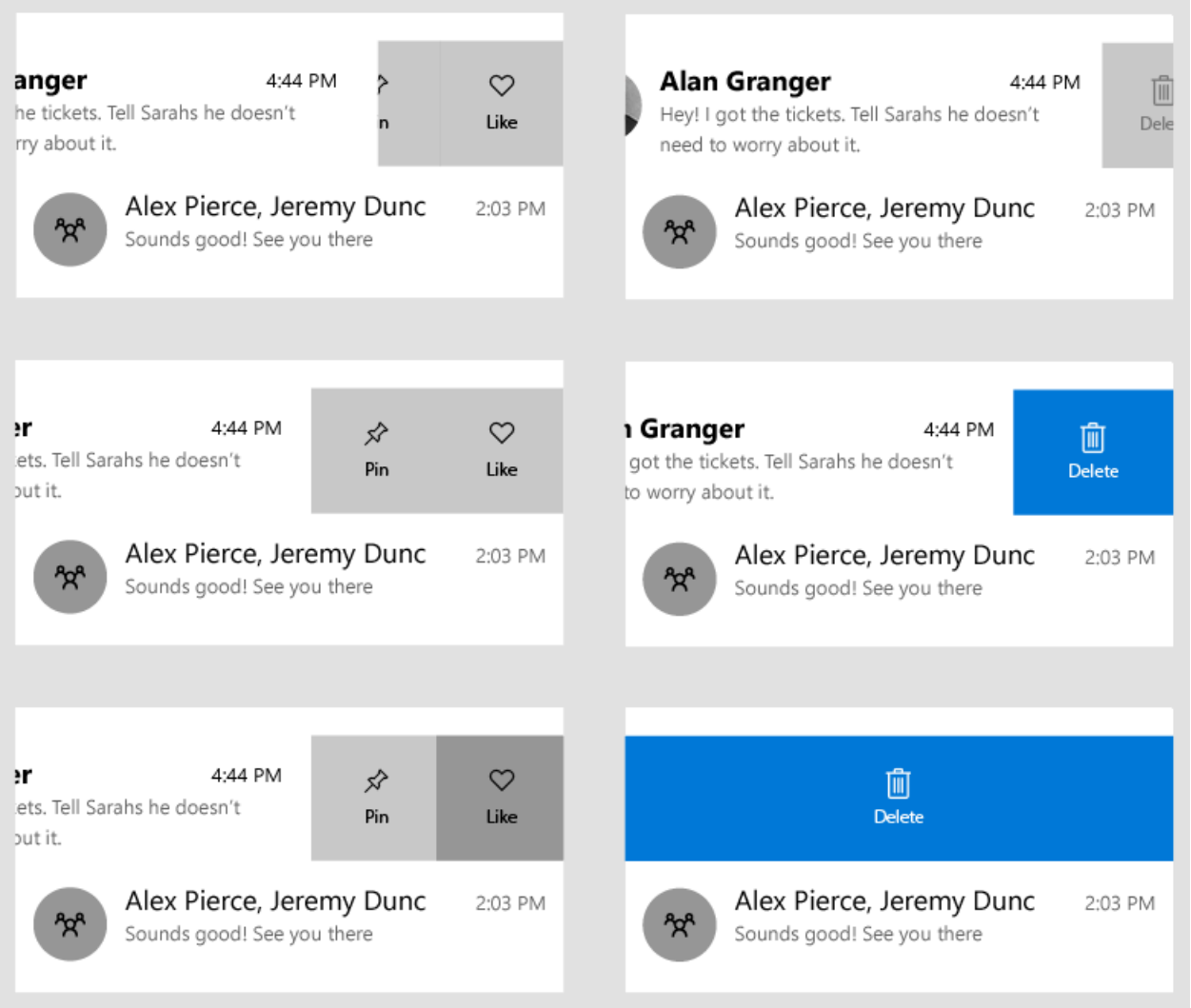
For command bar recommendations, see guidelines for command bars.

To learn more about selection modes and handling selection events for certain controls, visit the guidance page for that control.

Swipe

Article • 02/26/2025

Swipe commanding is an accelerator for context menus that lets users easily access common menu actions by touch, without needing to change states within the app.



Is this the right control?

Swipe commanding saves space. It's useful in situations where the user may perform the same operation on multiple items in quick succession. And it provides "quick actions" on items that don't need a full popup or state change within the page.

You should use swipe commanding when you have a potentially large group of items, and each item has 1-3 actions that a user may want to perform regularly. These actions may include, but are not limited to:

- Deleting
- Marking or archiving

- Saving or downloading
- Replying

How does Swipe work?

UWP swipe commanding has two modes: Reveal and Execute. It also supports four different swipe directions: up, down, left, and right.

Reveal mode

In Reveal mode, the user swipes an item to open a menu of one or more commands and must explicitly tap a command to execute it. When the user swipes and releases an item, the menu remains open until either a command is selected, or the menu is closed again through swiping back, tapping off, or scrolling the opened swipe item off the screen.

Reveal mode is a safer, more versatile swipe mode, and can be used for most types of menu actions, even potentially destructive actions, such as deletion.

When the user selects one of the menu options shown in the reveal's open and resting state, the command for that item is invoked and the swipe control is closed.

Execute mode

In Execute mode, the user swipes an item open to reveal and execute a single command with that one swipe. If the user releases the item being swiped before they swipe past a threshold, the menu closes and the command is not executed. If the user swipes past the threshold and then releases the item, the command is executed immediately.

If the user does not release their finger after the threshold is reached, and pulls the swipe item closed again, the command is not executed and no action is performed on the item.

Execute mode provides more visual feedback through color and label orientation while an item is being swiped.

Execute is best used when the action the user is performing is most common.

It may also be used for more destructive actions like deleting an item. However, keep in mind that Execute requires only one action of swiping in a direction, as opposed to Reveal, which requires the user to explicitly click on a button.

Swipe directions

Swipe works in all cardinal directions: up, down, left, and right. Each swipe direction can hold its own swipe items or content, but only one instance of a direction can be set at a time on a single swipe-able element.

For example, you cannot have two LeftItems definitions on the same SwipeControl.

Dos and don'ts

- Don't use swipe in FlipViews or Hubs. The combination may be confusing for the user because of conflicting swipe directions.
- Don't combine horizontal swipe with horizontal navigation, or vertical swipe with vertical navigation.
- Do make sure what the user is swiping is the same action, and is consistent across all related items that can be swiped.

- Do use swipe for the main actions a user will want to perform.
- Do use swipe on items where the same action is repeated many times.
- Do use horizontal swiping on wider items, and vertical swiping on taller items.
- Do use short, concise text labels.

Create a swipe command

✔ **Important APIs**: SwipeControl, SwipeItem, ListView class

Open the WinUI 3 Gallery app and see SwipeControl in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Swipe commands have two components that you need to define:

- The SwipeControl, which wraps around your content. In a collection, such as a ListView, this sits within your DataTemplate.
- The swipe menu items, which is one or more SwipeItem objects placed in the swipe control's directional containers: LeftItems, RightItems, TopItems, or BottomItems

Swipe content can be placed inline, or defined in the Resources section of your page or app.

Here's a simple example of a SwipeControl wrapped around some text. It shows the hierarchy of XAML elements required to create a swipe command.

XAML

```
<SwipeControl HorizontalAlignment="Center" VerticalAlignment="Center">
    <SwipeControl.LeftItems>
        <SwipeItems>
            <SwipeItem Text="Pin">
                <SwipeItem.IconSource>
                    <SymbolIconSource Symbol="Pin"/>
                </SwipeItem.IconSource>
            </SwipeItem>
        </SwipeItems>
    </SwipeControl.LeftItems>

     <!-- Swipeable content -->
    <Border Width="180" Height="44" BorderBrush="Black" BorderThickness="2">
        <TextBlock Text="Swipe to Pin" Margin="4,8,0,0"/>
```

```
    </Border>
</SwipeControl>
```

Now we'll take a look at a more complete example of how you would typically use swipe commands in a list. In this example, you'll set up a delete command that uses Execute mode, and a menu of other commands that uses Reveal mode. Both sets of commands are defined in the Resources section of the page. You'll apply the swipe commands to the items in a ListView.

First, create the swipe items, which represent the commands, as page level resources. SwipeItem uses an IconSource as its icon. Create the icons as resources, too.

XAML

```
<Page.Resources>
    <SymbolIconSource x:Key="ReplyIcon" Symbol="MailReply"/>
    <SymbolIconSource x:Key="DeleteIcon" Symbol="Delete"/>
    <SymbolIconSource x:Key="PinIcon" Symbol="Pin"/>

    <SwipeItems x:Key="RevealOptions" Mode="Reveal">
        <SwipeItem Text="Reply" IconSource="{StaticResource ReplyIcon}"/>
        <SwipeItem Text="Pin" IconSource="{StaticResource PinIcon}"/>
    </SwipeItems>

    <SwipeItems x:Key="ExecuteDelete" Mode="Execute">
        <SwipeItem Text="Delete" IconSource="{StaticResource DeleteIcon}"
                   Background="Red"/>
    </SwipeItems>
</Page.Resources>
```

Remember to keep the menu items in your swipe content to short, concise text labels. These actions should be the primary ones that a user may want to perform multiple times over a short period.

Setting up a swipe command to work in a collection or ListView is exactly the same as defining a single swipe command (shown previously), except you define your SwipeControl in a DataTemplate so it's applied to each item in the collection.

Here's a ListView with the SwipeControl applied in its item DataTemplate. The LeftItems and RightItems properties reference the swipe items that you created as resources.

XAML

```
<ListView x:Name="sampleList" Width="300">
    <ListView.ItemContainerStyle>
        <Style TargetType="ListViewItem">
            <Setter Property="HorizontalContentAlignment" Value="Stretch"/>
            <Setter Property="VerticalContentAlignment" Value="Stretch"/>
```

```
        </Style>
    </ListView.ItemContainerStyle>
    <ListView.ItemTemplate>
        <DataTemplate x:DataType="x:String">
            <SwipeControl x:Name="ListViewSwipeContainer"
                          LeftItems="{StaticResource RevealOptions}"
                          RightItems="{StaticResource ExecuteDelete}"
                          Height="60">
                <StackPanel Orientation="Vertical">
                    <TextBlock Text="{x:Bind}" FontSize="18"/>
                    <StackPanel Orientation="Horizontal">
                        <TextBlock Text="Lorem ipsum dolor sit amet,
consectetur adipiscing elit..." FontSize="12"/>
                    </StackPanel>
                </StackPanel>
            </SwipeControl>
        </DataTemplate>
    </ListView.ItemTemplate>
    <x:String>Item 1</x:String>
    <x:String>Item 2</x:String>
    <x:String>Item 3</x:String>
    <x:String>Item 4</x:String>
    <x:String>Item 5</x:String>
</ListView>
```

Handle an invoked swipe command

To act on a swipe command, you handle its Invoked event. (For more info about a how a user can invoke a command, review the *How does swipe work?* section earlier in this article.) Typically, a swipe command is in a ListView or list-like scenario. In that case, when a command is invoked, you want to perform an action on that swiped item.

Here's how to handle the Invoked event on the *delete* swipe item you created previously.

XAML

```
<SwipeItems x:Key="ExecuteDelete" Mode="Execute">
    <SwipeItem Text="Delete" IconSource="{StaticResource DeleteIcon}"
               Background="Red" Invoked="Delete_Invoked"/>
</SwipeItems>
```

The data item is the DataContext of the SwipeControl. In your code, you can access the item that was swiped by getting the SwipeControl.DataContext property from the event args, as shown here.

C#

```
 private void Delete_Invoked(SwipeItem sender, SwipeItemInvokedEventArgs
args)
 {
     sampleList.Items.Remove(args.SwipeControl.DataContext);
 }
```

> **Note**
>
> Here, the items were added directly to the ListView.Items collection for simplicity, so the item is also deleted the same way. If you instead set the ListView.ItemsSource to a collection, which is more typical, you need to delete the item from the source collection.

In this particular instance, you removed the item from the list, so the final visual state of the swiped item isn't important. However, in situations where you simply want to perform an action and then have the swipe collapse again, you can set the BehaviorOnInvoked property one of the SwipeBehaviorOnInvoked enum values.

- **Auto**
  - In Execute mode, the opened swipe item will remain open when invoked.
  - In Reveal mode, the opened swipe item will collapse when invoked.
- **Close**
  - When the item is invoked, the swipe control will always collapse and return to normal, regardless of the mode.
- **RemainOpen**
  - When the item is invoked, the swipe control will always stay open, regardless of the mode.

Here, a *reply* swipe item is set to close after its invoked.

XAML

```
<SwipeItem Text="Reply" IconSource="{StaticResource ReplyIcon}"
           Invoked="Reply_Invoked"
           BehaviorOnInvoked = "Close"/>
```

UWP and WinUI 2

> **Important**

The information and examples in this article are optimized for apps that use the **Windows App SDK** and **WinUI 3**, but are generally applicable to UWP apps that use **WinUI 2**. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

The SwipeControl for UWP apps is included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in both the Windows.UI.Xaml.Controls (UWP) and Microsoft.UI.Xaml.Controls (WinUI) namespaces.

- ✓ **UWP APIs:** SwipeControl, SwipeItem, ListView class
- ✓ **WinUI 2 Apis:** SwipeControl, SwipeItem
- ✓ Open the WinUI 2 Gallery app and see SwipeControl in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles, templates, and features for all controls.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:SwipeControl />
<muxc:SwipeItem />
```

Related articles

- List view and Grid view
- Item containers and templates
- Pull to refresh

Pull to refresh

Article • 02/26/2025

Pull-to-refresh lets a user pull down on a list of data using touch in order to retrieve more data. Pull-to-refresh is widely used on devices with a touch screen. You can use the APIs shown here to implement pull-to-refresh in your app.

Is this the right control?

Use pull-to-refresh when you have a list or grid of data that the user might want to refresh regularly, and your app is likely to be running on touch-first devices.

You can also use the RefreshVisualizer to create a consistent refresh experience that is invoked in other ways, such as by a refresh button.

Refresh controls

Pull-to-refresh is enabled by 2 controls.

- **RefreshContainer** - a ContentControl that provides a wrapper for the pull-to-refresh experience. It handles the touch interactions and manages the state of its internal refresh visualizer.

- **RefreshVisualizer** - encapsulates the refresh visualization explained in the next section.

The main control is the **RefreshContainer**, which you place as a wrapper around the content that the user pulls to trigger a refresh. RefreshContainer works only with touch, so we recommend that you also have a refresh button available for users who don't have a touch interface. You can position the refresh button at a suitable location in the app, either on a command bar or at a location close to the surface being refreshed.

Refresh visualization

The default refresh visualization is a circular progress spinner that is used to communicate when a refresh will happen and the progress of the refresh after it is initiated. The refresh visualizer has 5 states.

The distance the user needs to pull down on a list to initiate a refresh is called the *threshold*. The visualizer State is determined by the pull state as it relates to this threshold. The possible values are contained in the RefreshVisualizerState enumeration.

Idle

The visualizer's default state is **Idle**. The user is not interacting with the RefreshContainer via touch, and there is not a refresh in progress.

Visually, there is no evidence of the refresh visualizer.

Interacting

When the user pulls the list in the direction specified by the PullDirection property, and before the threshold is reached, the visualizer is in the **Interacting** state.

- If the user releases the control while in this state, the control returns to **Idle**.

Visually, the icon is displayed as disabled (60% opacity). In addition, the icon spins one full rotation with the scroll action.

- If the user pulls the list past the threshold, the visualizer transitions from **Interacting** to **Pending**.

Visually, the icon switches to 100% opacity and pulses in size up to 150% and then back to 100% size during the transition.

Pending

When the user has pulled the list past the threshold, the visualizer is in the **Pending** state.

- If the user moves the list back above the threshold without releasing it, it returns to the **Interacting** state.

- If the user releases the list, a refresh request is initiated and it transitions to the **Refreshing** state.

Visually, the icon is 100% in both size and opacity. In this state, the icon continues to move down with the scroll action but no longer spins.

Refreshing

When the user releases the visualiser past the threshold, it's in the **Refreshing** state.

When this state is entered, the **RefreshRequested** event is raised. This is the signal to start the app's content refresh. The event args (RefreshRequestedEventArgs) contain a Deferral object, which you should take a handle to in the event handler. Then, you should mark the deferral as completed when your code to perform the refresh has completed.

When the refresh is complete, the visualizer returns to the **Idle** state.

Visually, the icon settles back to the threshold location and spins for the duration of the refresh. This spinning is used to show progress of the refresh and is replaced by the animation of the incoming content.

Peeking

When the user pulls in the refresh direction from a start position where a refresh is not allowed, the visualizer enters the **Peeking** state. This typically happens when the ScrollViewer is not at position 0 when the user starts to pull.

- If the user releases the control while in this state, the control returns to **Idle**.

Pull direction

By default, the user pulls a list from top to bottom to initiate a refresh. If you have a list or grid with a different orientation, you should change the pull direction of the refresh container to match.

The PullDirection property takes one of these RefreshPullDirection values: **BottomToTop**, **TopToBottom**, **RightToLeft**, or **LeftToRight**.

When you change the pull direction, the starting position of the visualizer's progress spinner automatically rotates so the arrow starts in the appropriate position for the pull direction. If needed, you can change the RefreshVisualizer.Orientation property to override the automatic behavior. In most cases, we recommend leaving the default value of **Auto**.

Implement pull-to-refresh

✓ **Important APIs**: RefreshContainer, RefreshVisualizer

Open the WinUI 3 Gallery app and see PullToRefresh in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

To add pull-to-refresh functionality to a list requires just a few steps.

1. Wrap your list in a **RefreshContainer** control.
2. Handle the **RefreshRequested** event to refresh your content.
3. Optionally, initiate a refresh by calling **RequestRefresh** (for example, from a button click).

> **Note**
>
> You can instantiate a RefreshVisualizer on its own. However, we recommend that you wrap your content in a RefreshContainer and use the RefreshVisualizer provided by the RefreshContainer.Visualizer property, even for non-touch scenarios. In this article, we assume that the visualizer is always obtained from the refresh container.
>
> In addition, use the refresh container's RequestRefresh and RefreshRequested members for convenience. `refreshContainer.RequestRefresh()` is equivalent to

`refreshContainer.Visualizer.RequestRefresh()`, and either will raise both the RefreshContainer.RefreshRequested event and the RefreshVisualizer.RefreshRequested events.

Request a refresh

The refresh container handles touch interactions to let a user refresh content via touch. We recommend that you provide other affordances for non-touch interfaces, like a refresh button or voice control.

To initiate a refresh, call the RequestRefresh method.

C#

```
// See the Examples section for the full code.
private void RefreshButtonClick(object sender, RoutedEventArgs e)
{
    RefreshContainer.RequestRefresh();
}
```

When you call RequestRefresh, the visualizer state goes directly from **Idle** to **Refreshing**.

Handle a refresh request

To get fresh content when needed, handle the RefreshRequested event. In the event handler, you'll need code specific to your app to get the fresh content.

The event args (RefreshRequestedEventArgs) contain a Deferral object. Get a handle to the deferral in the event handler. Then, mark the deferral as completed when your code to perform the refresh has completed.

C#

```
// See the Examples section for the full code.
private async void RefreshContainer_RefreshRequested(RefreshContainer
sender, RefreshRequestedEventArgs args)
{
    // Respond to a request by performing a refresh and using the deferral
object.
    using (var RefreshCompletionDeferral = args.GetDeferral())
    {
        // Do some async operation to refresh the content

         await FetchAndInsertItemsAsync(3);

        // The 'using' statement ensures the deferral is marked as complete.
```

```
        // Otherwise, you'd call
        // RefreshCompletionDeferral.Complete();
        // RefreshCompletionDeferral.Dispose();
    }
}
```

Respond to state changes

You can respond to changes in the visualizer's state, if needed. For example, to prevent multiple refresh requests, you can disable a refresh button while the visualizer is refreshing.

C#

```
// See the Examples section for the full code.
private void Visualizer_RefreshStateChanged(RefreshVisualizer sender,
RefreshStateChangedEventArgs args)
{
    // Respond to visualizer state changes.
    // Disable the refresh button if the visualizer is refreshing.
    if (args.NewState == RefreshVisualizerState.Refreshing)
    {
        RefreshButton.IsEnabled = false;
    }
    else
    {
        RefreshButton.IsEnabled = true;
    }
}
```

Using a ScrollViewer in a RefreshContainer

> **Note**
>
> The Content of a RefreshContainer must be a scrollable control, such as ScrollViewer, GridView, ListView, etc. Setting the Content to a control like Grid will result in undefined behavior.

This example shows how to use pull-to-refresh with a scroll viewer.

XAML

```
<RefreshContainer>
    <ScrollViewer VerticalScrollMode="Enabled"
                  VerticalScrollBarVisibility="Auto"
```

```
                    HorizontalScrollBarVisibility="Auto">

        <!-- Scrollviewer content -->

    </ScrollViewer>
</RefreshContainer>
```

Adding pull-to-refresh to a ListView

This example shows how to use pull-to-refresh with a list view.

XAML

```
<StackPanel Margin="0,40" Width="280">
    <CommandBar OverflowButtonVisibility="Collapsed">
        <AppBarButton x:Name="RefreshButton" Click="RefreshButtonClick"
                      Icon="Refresh" Label="Refresh"/>
        <CommandBar.Content>
            <TextBlock Text="List of items"
                       Style="{StaticResource TitleTextBlockStyle}"
                       Margin="12,8"/>
        </CommandBar.Content>
    </CommandBar>

    <RefreshContainer x:Name="RefreshContainer">
        <ListView x:Name="ListView1" Height="400">
            <ListView.ItemTemplate>
                <DataTemplate x:DataType="local:ListItemData">
                    <Grid Height="80">
                        <Grid.RowDefinitions>
                            <RowDefinition Height="Auto" />
                            <RowDefinition Height="Auto" />
                            <RowDefinition Height="*" />
                        </Grid.RowDefinitions>
                        <TextBlock Text="{x:Bind Path=Header}"
                                   Style="{StaticResource
SubtitleTextBlockStyle}"
                                   Grid.Row="0"/>
                        <TextBlock Text="{x:Bind Path=Date}"
                                   Style="{StaticResource
CaptionTextBlockStyle}"
                                   Grid.Row="1"/>
                        <TextBlock Text="{x:Bind Path=Body}"
                                   Style="{StaticResource
BodyTextBlockStyle}"
                                   Grid.Row="2"
                                   Margin="0,4,0,0" />
                    </Grid>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
```

```
    </RefreshContainer>
</StackPanel>
```

C#

```
public sealed partial class MainPage : Page
{
    public ObservableCollection<ListItemData> Items { get; set; }
        = new ObservableCollection<ListItemData>();

    public MainPage()
    {
        this.InitializeComponent();

        Loaded += MainPage_Loaded;
        ListView1.ItemsSource = Items;
    }

    private async void MainPage_Loaded(object sender, RoutedEventArgs e)
    {
        Loaded -= MainPage_Loaded;
        RefreshContainer.RefreshRequested +=
RefreshContainer_RefreshRequested;
        RefreshContainer.Visualizer.RefreshStateChanged +=
Visualizer_RefreshStateChanged;

        // Add some initial content to the list.
        await FetchAndInsertItemsAsync(2);
    }

    private void RefreshButtonClick(object sender, RoutedEventArgs e)
    {
        RefreshContainer.RequestRefresh();
    }

    private async void RefreshContainer_RefreshRequested(RefreshContainer
sender, RefreshRequestedEventArgs args)
    {
        // Respond to a request by performing a refresh and using the
deferral object.
        using (var RefreshCompletionDeferral = args.GetDeferral())
        {
            // Do some async operation to refresh the content

            await FetchAndInsertItemsAsync(3);

            // The 'using' statement ensures the deferral is marked as
complete.
            // Otherwise, you'd call
            // RefreshCompletionDeferral.Complete();
            // RefreshCompletionDeferral.Dispose();
        }
    }
```

```
    private void Visualizer_RefreshStateChanged(RefreshVisualizer sender,
RefreshStateChangedEventArgs args)
    {
        // Respond to visualizer state changes.
        // Disable the refresh button if the visualizer is refreshing.
        if (args.NewState == RefreshVisualizerState.Refreshing)
        {
            RefreshButton.IsEnabled = false;
        }
        else
        {
            RefreshButton.IsEnabled = true;
        }
    }

    // App specific code to get fresh data.
    private async Task FetchAndInsertItemsAsync(int updateCount)
    {
        for (int i = 0; i < updateCount; ++i)
        {
            // Simulate delay while we go fetch new items.
            await Task.Delay(1000);
            Items.Insert(0, GetNextItem());
        }
    }

    private ListItemData GetNextItem()
    {
        return new ListItemData()
        {
            Header = "Header " + DateTime.Now.Second.ToString(),
            Date = DateTime.Now.ToLongDateString(),
            Body = DateTime.Now.ToLongTimeString()
        };
    }
}

public class ListItemData
{
    public string Header { get; set; }
    public string Date { get; set; }
    public string Body { get; set; }
}
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that

use **WinUI 2**. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

The refresh controls for UWP apps are included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in both the Windows.UI.Xaml.Controls (UWP) and Microsoft.UI.Xaml.Controls (WinUI) namespaces.

- ✔ **UWP APIs:** RefreshContainer, RefreshVisualizer
- ✔ **WinUI 2 Apis:** RefreshContainer, RefreshVisualizer
- ✔ Open the WinUI 2 Gallery app and see PullToRefresh in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles, templates, and features for all controls.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:RefreshContainer />
```

Related articles

- Touch interactions
- List view and grid view
- Item containers and templates
- Expression animations

Filtering collections and lists through user input

Article • 03/29/2024

If your collection displays many items or is heavily tied to user interaction, filtering is a useful feature to implement. Filtering using the method described in this article can be implemented with most collection controls, including ListView, GridView, and ItemsView. Many types of user input can be used to filter a collection - such as checkboxes, radio buttons, and sliders - but this article demonstrates taking text-based user input and using it to update a ListView in real time, according to the user's search.

Setting up the UI for filtering

To implement text filtering, your app will need a ListView and a TextBox or other control that allows user input. The text that the user types into the TextBox is used as the filter; that is, only results that contain the user's text input will appear in the ListView. As the user types into the TextBox, the ListView constantly updates with filtered results.

> **Note**
>
> This article demonstrates filtering with a ListView. However, the filtering that is demonstrated can also be applied to other collection controls such as GridView, ItemsView, or TreeView.

The following XAML shows a UI with a simple ListView along with an accompanying TextBox. In this example, the ListView displays a collection of `Contact` objects. `Contact` is a class defined in the code-behind, and each `Contact` object has the following properties: `FirstName`, `LastName`, and `Company`.

The user can type a filtering term into the TextBox to filter the list of `Contact` objects by last name. The TextBox has it's `x:Name` attribute set (`FilterByLastName`) so you can access the TextBox's Text property in the code-behind. You also handle it's TextChanged event (`OnFilterChanged`). The TextChanged event occurs whenever the user types in the TextBox, letting you perform a filtering operation upon receiving user input.

For filtering to work, the ListView must have a data source that can be manipulated in the code-behind, such as an ObservableCollection<T>. In this case, the ListView's

ItemsSource property is assigned to an `ObservableCollection<Contact>` in the code-behind.

> **Tip**
>
> This is a simplified version of the example in the ListView page of the **WinUI Gallery app**. Use the WinUI Gallery app to run and view the full code, including the ListView's **DataTemplate** and the `Contact` class.

XAML

```
<Grid>
    <StackPanel Width="300" Margin="24"
                HorizontalAlignment="Left">
        <TextBox x:Name="FilterByLastName"
                 Header="Filter by Last Name"
                 TextChanged="OnFilterChanged"/>
        <ListView x:Name="FilteredListView"
                  ItemTemplate="{StaticResource ContactListViewTemplate}"/>
    </StackPanel>
</Grid>
```

Filtering the data

Linq queries let you group, order, and select certain items in a collection. To filter a list, you construct a Linq query that selects only items that match the user-entered filtering term, entered in the `FilterByLastName` TextBox. The query result can be assigned to an IEnumerable<T> collection object. Once you have this collection, you can use it to compare with the original list, removing items that don't match and adding back items that do match (in case of a backspace).

> **Note**
>
> In order for the ListView to animate in the most intuitive way when adding and removing items, it's important to add and remove items in the ListView's **ItemsSource** collection itself, rather than create a new collection of filtered objects and assign that to the ListView's ItemsSource property.

To start, you'll need to initialize your original data source in a separate collection, such as a List<T>. In this example, you have a `List<Contact>` called `allContacts` that holds all of the `Contact` objects that can potentially be shown in the ListView.

You'll also need a collection to hold the filtered data, which will constantly change every time a filter is applied. For this, you'll use an ObservableCollection<T> so that the ListView is notified to update whenever the collection changes. In this example, it's an `ObservableCollection<Person>` called `contactsFiltered`, and is the ItemsSource for the ListView. At initialization, it will have the same contents as `allContacts`.

The filtering operation is performed through these steps, shown in the following code:

- Set the ListView's ItemsSource property to `contactsFiltered`.
- Handle the TextChanged event (`OnFilterChanged`) for the `FilterByLastName` TextBox. Inside this event handler function, filter the data.
- To filter the data, access the user-entered filtering term through the `FilterByLastName.Text` property. Use a Linq query to select the items in `allContacts` where the last name contains the term in `FilterByLastName.Text`, and add those matching items into a collection called `filtered`.
- Compare the current `contactsFiltered` collection with the newly filtered items in `filtered`, removing and adding items in `contactsFiltered` where necessary to make it match `filtered`.
- As items are removed and added in `contactsFiltered`, the ListView updates and animates accordingly.

C#

```
using System.Linq;

public sealed partial class MainPage : Page
{
   // Define Contact collection to hold all Contact objects.
   IList<Contact> allContacts = new List<Contact>();
   // Define an ObservableCollection<Contact> object to serve as the
ListView's
   // ItemsSource. This collection will get updated after the filters are
used:
   ObservableCollection<Contact> contactsFiltered;

   public MainPage()
   {
       this.InitializeComponent();

       // Populate allContacts collection.
       allContacts.Add(new Contact("Kendall", "Collins", "Adatum
Corporation"));
       allContacts.Add(new Contact("Victoria", "Burke", "Bellows College"));
       allContacts.Add(new Contact("Preston", "Morales", "Margie's
Travel"));
       allContacts.Add(new Contact("Miguel", "Reyes", "Tailspin Toys"));
```

```
        // Populate contactsFiltered with all Contact objects (in this case,
        // allContacts holds all of our Contact objects so we copy them into
        // contactsFiltered). Set this newly populated collection as the
        // ItemsSource for the ListView.
        contactsFiltered = new ObservableCollection<Contact>(allContacts);
        Filtereditemscontrol.itemssource = contactsFiltered;
    }

    // Whenever text changes in the filtering text box, this function is
called:
    private void OnFilterChanged(object sender, TextChangedEventArgs args)
    {
        // This is a Linq query that selects only items that return true
after
        // being passed through the Filter function, and adds all of those
        // selected items to filtered.
        var filtered = allContacts.Where(contact => Filter(contact));
        Remove_NonMatching(filtered);
        AddBack_Contacts(filtered);
    }

    // The following functions are called inside OnFilterChanged:

    // When the text in any filter is changed, perform a check on each item
in
    // the original contact list to see if the item should be displayed. If
the
    // item passes the check, the function returns true and the item is added
to
    // the filtered list. Make sure all text is case-insensitive when
comparing.
    private bool Filter(Contact contact)
    {
        return contact.LastName.Contains
            (FilterByLastName.Text,
StringComparison.InvariantCultureIgnoreCase);
    }

    // These functions go through the current list being displayed
    // (contactsFiltered), and remove any items not in the filtered
collection
    // (any items that don't belong), or add back any items from the original
    // allContacts list that are now supposed to be displayed.
(Adding/removing
    // the items ensures the list view uses the desired add/remove
animations.)

    private void Remove_NonMatching(IEnumerable<Contact> filteredData)
    {
        for (int i = contactsFiltered.Count - 1; i >= 0; i--)
        {
            var item = contactsFiltered[i];
            // If contact is not in the filtered argument list,
            // remove it from the ListView's source.
            if (!filteredData.Contains(item))
```

```
            {
                contactsFiltered.Remove(item);
            }
        }
    }

    private void AddBack_Contacts(IEnumerable<Contact> filteredData)
    {
        foreach (var item in filteredData)
        {
            // If the item in the filtered list is not currently in
            // the ListView's source collection, add it back in.
            if (!contactsFiltered.Contains(item))
            {
                contactsFiltered.Add(item);
            }
        }
    }
}
```

Now, as the user types in their filtering string in the `FilterByLastName` TextBox, the ListView immediately updates to show only the people whose last name contains the filtering string.

Get the sample code

Open the WinUI 3 Gallery app and see the ListView in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

> For UWP: The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

Related articles

- Lists
- Items view
- List view and grid view
- Collection commanding

Inverted lists

Article • 06/09/2022

You can use a list view to present a conversation in a chat experience with items that are visually distinct to represent the sender/receiver. Using different colors and horizontal alignment to separate messages from the sender/receiver helps the user quickly orient themselves in a conversation.

> **Important APIs**: ListView class, ItemsStackPanel class, ItemsUpdatingScrollMode property

You will typically need to present the list such that it appears to grow from the bottom up instead of from the top down. When a new message arrives and is added to the end, the previous messages slide up to make room drawing the user's attention to the latest arrival. However, if a user has scrolled up to view previous replies then the arrival of a new message must not cause a visual shift that would disrupt their focus.

Create an inverted list

To create an inverted list, use a list view with an ItemsStackPanel as its items panel. On the ItemsStackPanel, set the ItemsUpdatingScrollMode to KeepLastItemInView.

> **ⓘ Important**
>
> The **KeepLastItemInView** enum value is available starting with Windows 10, version 1607. You can't use this value when your app runs on earlier versions of Windows 10.

This example shows how to align the list view's items to the bottom and indicate that when there is a change to the items the last item should remain in view.

XAML

XAML

```
<ListView>
   <ListView.ItemsPanel>
       <ItemsPanelTemplate>
           <ItemsStackPanel VerticalAlignment="Bottom"
                            ItemsUpdatingScrollMode="KeepLastItemInView"/>
       </ItemsPanelTemplate>
   </ListView.ItemsPanel>
</ListView>
```

Recommendations

- Align messages from the sender/receiver on opposite sides to make the flow of conversation clear to users.
- Animate the existing messages out of the way to display the latest message if the user is already at the end of the conversation awaiting the next message.
- Don't disrupt the users focus by moving items if they're not reading the end of the conversation.

Get the sample code

- XAML Bottom-up list sample

Nested UI in list items

Article • 06/09/2022

Nested UI is a user interface (UI) that exposes nested actionable controls enclosed inside a container that also can take independent focus.

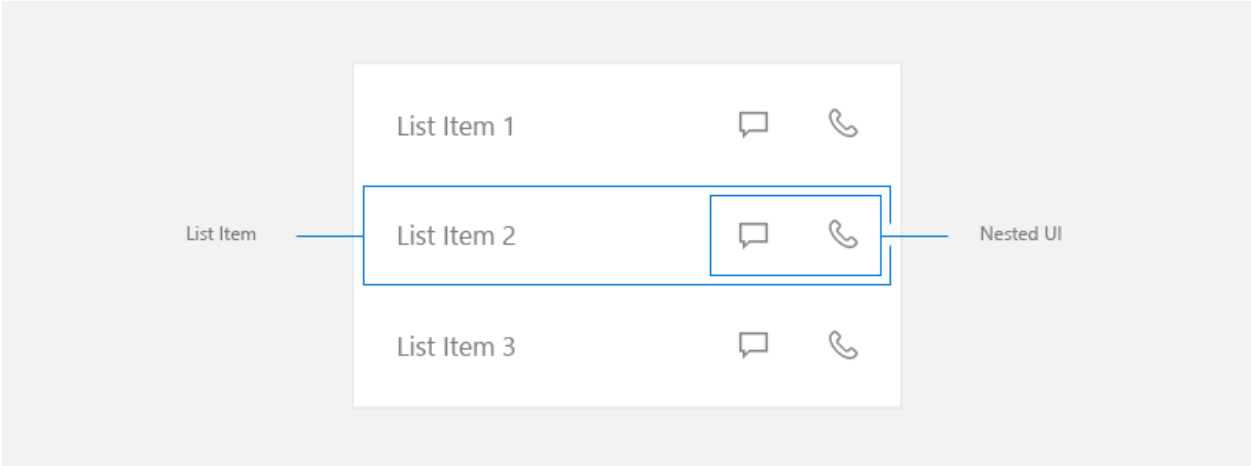
You can use nested UI to present a user with additional options that help accelerate taking important actions. However, the more actions you expose, the more complicated your UI becomes. You need to take extra care when you choose to use this UI pattern. This article provides guidelines to help you determine the best course of action for your particular UI.

> **Important APIs**: ListView class, GridView class

In this article, we discuss the creation of nested UI in ListView and GridView items. While this section does not talk about other nested UI cases, these concepts are transferrable. Before you start, you should be familiar with the general guidance for using ListView or GridView controls in your UI, which is found in the Lists and List view and grid view articles.

In this article, we use the terms *list*, *list item*, and *nested UI* as defined here:

- *List* refers to a collection of items contained in a list view or grid view.
- *List item* refers to an individual item that a user can take action on in a list.
- *Nested UI* refers to UI elements within a list item that a user can take action on separate from taking action on the list item itself.



> NOTE ListView and GridView both derive from the ListViewBase class, so they have the same functionality, but display data differently. In this article, when we talk about lists, the info applies to both the ListView and GridView controls.

Primary and secondary actions

When creating UI with a list, consider what actions the user might take from those list items.

- Can a user click on the item to perform an action?
  - Typically, clicking a list item initiates an action, but it doesn't have too.
- Is there more than one action the user can take?
  - For example, tapping an email in a list opens that email. However, there might be other actions, like deleting the email, that the user would want to take without opening it first. It would benefit the user to access this action directly in the list.
- How should the actions be exposed to the user?
  - Consider all input types. Some forms of nested UI work great with one method of input, but might not work with other methods.

The *primary action* is what the user expects to happen when they press the list item.

Secondary actions are typically accelerators associated with list items. These accelerators can be for list management or actions related to the list item.

Options for secondary actions

When creating list UI, you first need to make sure you account for all input methods that Windows supports. For more info about different kinds of input, see Input primer.

After you have made sure that your app supports all inputs that Windows supports, you should decide if your app's secondary actions are important enough to expose as accelerators in the main list. Remember that the more actions you expose, the more complicated your UI becomes. Do you really need to expose the secondary actions in the main list UI, or can you put them somewhere else?

You might consider exposing additional actions in the main list UI when those actions need to be accessible by any input at all times.

If you decide that putting secondary actions in the main list UI is not necessary, there are several other ways you can expose them to the user. Here are some options you can consider for where to place secondary actions.

Put secondary actions on the detail page

Put the secondary actions on the page that the list item navigates to when it's pressed. When you use the list/details pattern, the detail page is often a good place to put secondary actions.

For more info, see the list/detail pattern.

Put secondary actions in a context menu

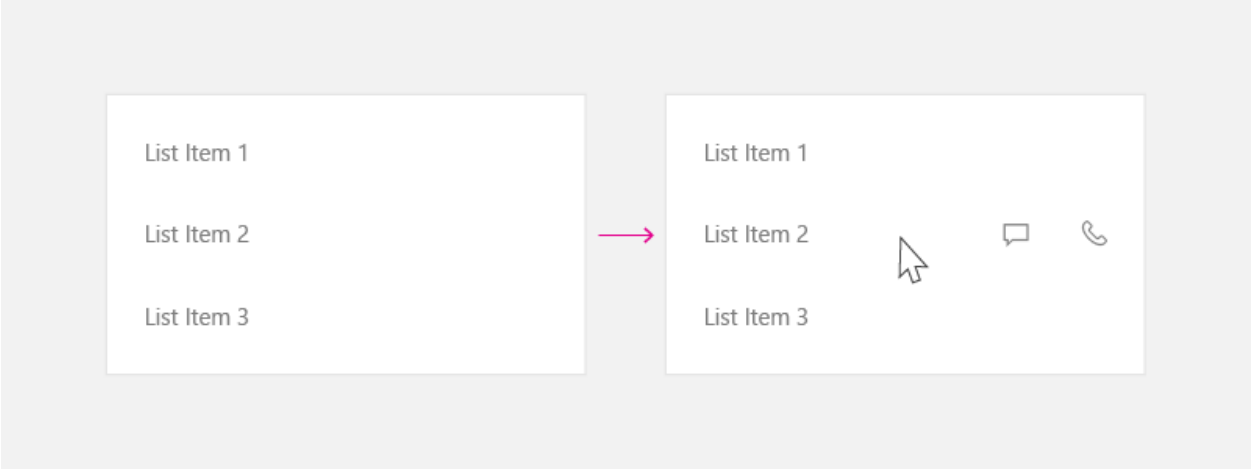
Put the secondary actions in a context menu that the user can access via right-click or press-and-hold. This provides the benefit of letting the user perform an action, such as deleting an email, without having to load the detail page. It's a good practice to also make these options available on the detail page, as context menus are intended to be accelerators rather than primary UI.

To expose secondary actions when input is from a gamepad or remote control, we recommend that you use a context menu.

For more info, see Context menus and flyouts.

Put secondary actions in hover UI to optimize for pointer input

If you expect your app to be used frequently with pointer input such as mouse and pen, and want to make secondary actions readily available only to those inputs, then you can show the secondary actions only on hover. This accelerator is visible only when a pointer input is used, so be sure to use the other options to support other input types as well.



For more info, see Mouse interactions.

UI placement for primary and secondary actions

If you decide that secondary actions should be exposed in the main list UI, we recommend the following guidelines.

When you create a list item with primary and secondary actions, place the primary action to the left and secondary actions to the right. In left-to-right reading cultures, users associate actions on the left side of list item as the primary action.

In these examples, we talk about list UI where the item flows more horizontally (it is wider than its height). However, you might have list items that are more square in shape, or taller than their width. Typically, these are items used in a grid. For these items, if the list doesn't scroll vertically, you can place the secondary actions at the bottom of the list item rather than to the right side.

Consider all inputs

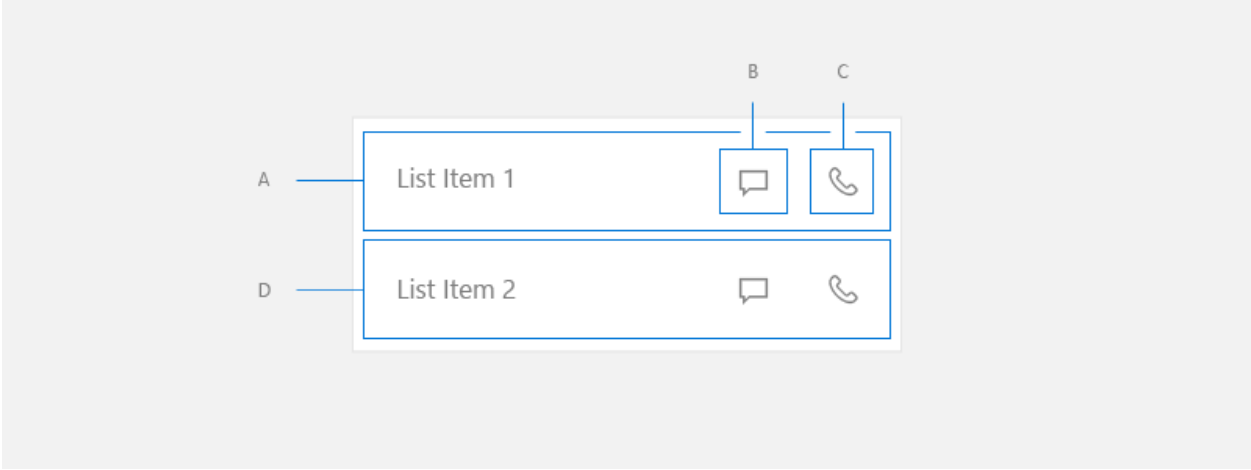
When deciding to use nested UI, also evaluate the user experience with all input types. As mentioned earlier, nested UI works great for some input types. However, it does not always work great for some other. In particular, keyboard, controller, and remote inputs can have difficulty accessing nested UI elements. Be sure to follow the guidance below to ensure your Windows works with all input types.

Nested UI handling

When you have more than one action nested in the list item, we recommend this guidance to handle navigation with a keyboard, gamepad, remote control, or other non-pointer input.

Nested UI where list items perform an action

If your list UI with nested elements supports actions such as invoking, selection (single or multiple), or drag-and-drop operations, we recommend these arrowing techniques to navigate through your nested UI elements.



Gamepad

When input is from a gamepad, provide this user experience:

- From **A**, right directional key puts focus on **B**.
- From **B**, right directional key puts focus on **C**.
- From **C**, right directional key is either no op, or if there is a focusable UI element to the right of List, put the focus there.
- From **C**, left directional key puts focus on **B**.
- From **B**, left directional key puts focus on **A**.
- From **A**, left directional key is either no op, or if there is a focusable UI element to the right of List, put the focus there.
- From **A**, **B**, or **C**, down directional key puts focus on **D**.
- From UI element to the left of List Item, right directional key puts focus on **A**.
- From UI element to the right of List Item, left directional key puts focus on **A**.

Keyboard

When input is from a keyboard, this is the experience user gets:

- From **A**, tab key puts focus on **B**.
- From **B**, tab key puts focus on **C**.
- From **C**, tab key puts focus on next focusable UI element in the tab order.
- From **C**, shift+tab key puts focus on **B**.
- From **B**, shift+tab or left arrow key puts focus on **A**.
- From **A**, shift+tab key puts focus on next focusable UI element in the reverse tab order.
- From **A**, **B**, or **C**, down arrow key puts focus on **D**.
- From UI element to the left of List Item, tab key puts focus on **A**.
- From UI element to the right of List Item, shift tab key puts focus on **C**.

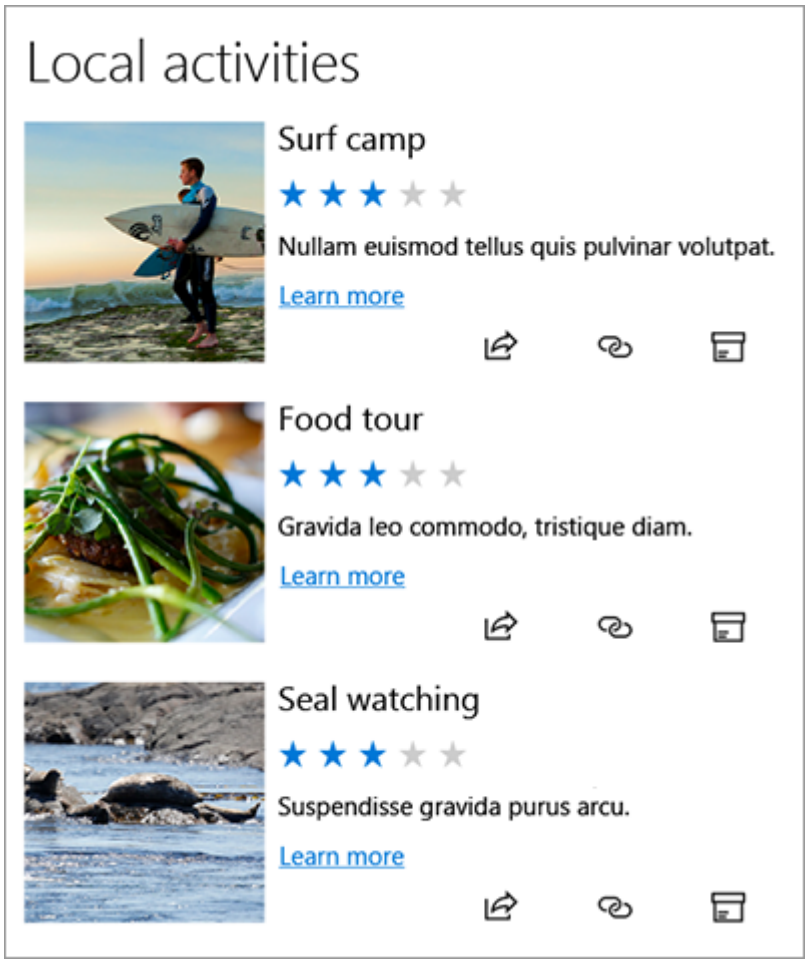
To achieve this UI, set IsItemClickEnabled to **true** on your list. SelectionMode can be any value.

For the code to implement this, see the Example section of this article.

Nested UI where list items do not perform an action

You might use a list view because it provides virtualization and optimized scrolling behavior, but not have an action associated with a list item. These UIs typically use the list item only to group elements and ensure they scroll as a set.

This kind of UI tends to be much more complicated than the previous examples, with a lot of nested elements that the user can take action on.



To achieve this UI, set the following properties on your list:

- SelectionMode to **None**.
- IsItemClickEnabled to **false**.
- IsFocusEngagementEnabled to **true**.

XAML

```
<ListView SelectionMode="None" IsItemClickEnabled="False" >
    <ListView.ItemContainerStyle>
        <Style TargetType="ListViewItem">
            <Setter Property="IsFocusEngagementEnabled" Value="True"/>
        </Style>
```

```
    </ListView.ItemContainerStyle>
</ListView>
```

When the list items do not perform an action, we recommend this guidance to handle navigation with a gamepad or keyboard.

Gamepad

When input is from a gamepad, provide this user experience:

- From List Item, down directional key puts focus on next List Item.
- From List Item, left/right key is either no op, or if there is a focusable UI element to the right of List, put the focus there.
- From List Item, 'A' button puts the focus on Nested UI in top/down left/right priority.
- While inside Nested UI, follow the XY Focus navigation model. Focus can only navigate around Nested UI contained inside the current List Item until user presses 'B' button, which puts the focus back onto the List Item.

Keyboard

When input is from a keyboard, this is the experience user gets:

- From List Item, down arrow key puts focus on the next List Item.
- From List Item, pressing left/right key is no op.
- From List Item, pressing tab key puts focus on the next tab stop amongst the Nested UI item.
- From one of the Nested UI items, pressing tab traverses the nested UI items in tab order. Once all the Nested UI items are traveled to, it puts the focus onto the next control in tab order after ListView.
- Shift+Tab behaves in reverse direction from tab behavior.

Example

This example shows how to implement nested UI where list items perform an action.

XAML

```
<ListView SelectionMode="None" IsItemClickEnabled="True"
          ChoosingItemContainer="listview1_ChoosingItemContainer"/>
```

C#

```
private void OnListViewItemKeyDown(object sender, KeyRoutedEventArgs e)
{
    // Code to handle going in/out of nested UI with gamepad and remote
only.
    if (e.Handled == true)
    {
        return;
    }

    var focusedElementAsListViewItem = FocusManager.GetFocusedElement() as
ListViewItem;
    if (focusedElementAsListViewItem != null)
    {
        // Focus is on the ListViewItem.
        // Go in with Right arrow.
        Control candidate = null;

        switch (e.OriginalKey)
        {
            case Windows.System.VirtualKey.GamepadDPadRight:
            case Windows.System.VirtualKey.GamepadLeftThumbstickRight:
                var rawPixelsPerViewPixel =
DisplayInformation.GetForCurrentView().RawPixelsPerViewPixel;
                GeneralTransform generalTransform =
focusedElementAsListViewItem.TransformToVisual(null);
                Point startPoint = generalTransform.TransformPoint(new
Point(0, 0));
                Rect hintRect = new Rect(startPoint.X *
rawPixelsPerViewPixel, startPoint.Y * rawPixelsPerViewPixel, 1,
focusedElementAsListViewItem.ActualHeight * rawPixelsPerViewPixel);
                candidate =
FocusManager.FindNextFocusableElement(FocusNavigationDirection.Right,
hintRect) as Control;
                break;
        }

        if (candidate != null)
        {
            candidate.Focus(FocusState.Keyboard);
            e.Handled = true;
        }
    }
    else
    {
        // Focus is inside the ListViewItem.
        FocusNavigationDirection direction = FocusNavigationDirection.None;
        switch (e.OriginalKey)
        {
            case Windows.System.VirtualKey.GamepadDPadUp:
            case Windows.System.VirtualKey.GamepadLeftThumbstickUp:
                direction = FocusNavigationDirection.Up;
                break;
            case Windows.System.VirtualKey.GamepadDPadDown:
            case Windows.System.VirtualKey.GamepadLeftThumbstickDown:
```

```
                direction = FocusNavigationDirection.Down;
                break;
            case Windows.System.VirtualKey.GamepadDPadLeft:
            case Windows.System.VirtualKey.GamepadLeftThumbstickLeft:
                direction = FocusNavigationDirection.Left;
                break;
            case Windows.System.VirtualKey.GamepadDPadRight:
            case Windows.System.VirtualKey.GamepadLeftThumbstickRight:
                direction = FocusNavigationDirection.Right;
                break;
            default:
                break;
        }

        if (direction != FocusNavigationDirection.None)
        {
            Control candidate =
FocusManager.FindNextFocusableElement(direction) as Control;
            if (candidate != null)
            {
                ListViewItem listViewItem = sender as ListViewItem;

                // If the next focusable candidate to the left is outside of
ListViewItem,
                // put the focus on ListViewItem.
                if (direction == FocusNavigationDirection.Left &&
                    !listViewItem.IsAncestorOf(candidate))
                {
                    listViewItem.Focus(FocusState.Keyboard);
                }
                else
                {
                    candidate.Focus(FocusState.Keyboard);
                }
            }

            e.Handled = true;
        }
    }
}

private void listview1_ChoosingItemContainer(ListViewBase sender,
ChoosingItemContainerEventArgs args)
{
    if (args.ItemContainer == null)
    {
        args.ItemContainer = new ListViewItem();
        args.ItemContainer.KeyDown += OnListViewItemKeyDown;
    }
}
```

C#

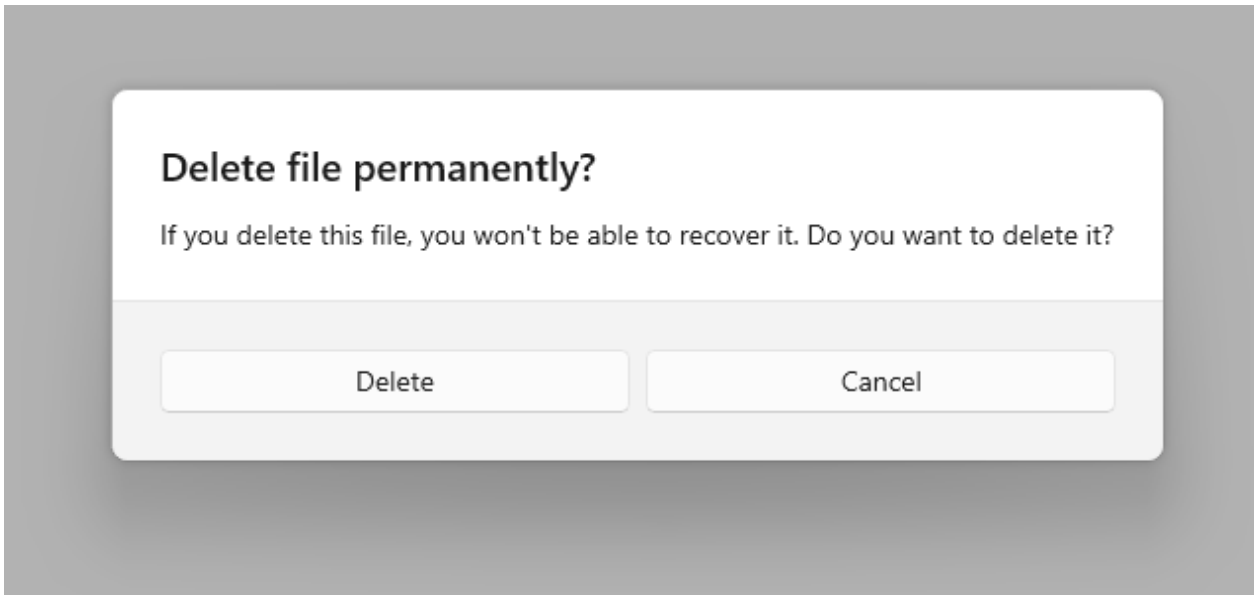
```
// DependencyObjectExtensions.cs definition.
public static class DependencyObjectExtensions
{
    public static bool IsAncestorOf(this DependencyObject parent,
DependencyObject child)
    {
        DependencyObject current = child;
        bool isAncestor = false;

        while (current != null && !isAncestor)
        {
            if (current == parent)
            {
                isAncestor = true;
            }

            current = VisualTreeHelper.GetParent(current);
        }

        return isAncestor;
    }
}
```

Dialogs and flyouts

Article • 10/13/2022

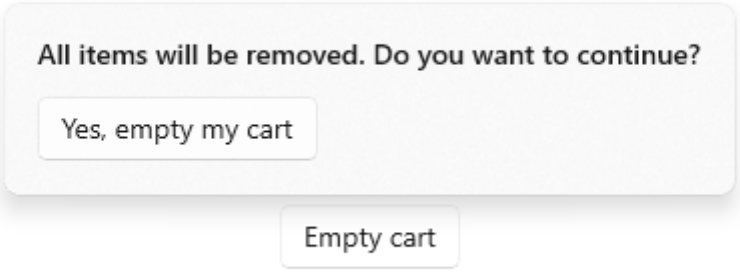
Dialogs and flyouts are transient UI elements that appear when something happens that requires notification, approval, or additional information from the user.

Dialogs



Dialogs are modal UI overlays that provide contextual app information. Dialogs block interactions with the app window until being explicitly dismissed. They often request some kind of action from the user.

Flyouts



A flyout is a lightweight contextual popup that displays UI related to what the user is doing. It includes placement and sizing logic, and can be used to reveal a secondary control or show more detail about an item.

Unlike a dialog, a flyout can be quickly dismissed by tapping or clicking somewhere outside the flyout, pressing the Escape key or Back button, resizing the app window, or changing the device's orientation.

Is this the right control?

Dialogs and flyouts make sure that users are aware of important information, but they also disrupt the user experience. Because dialogs are modal (blocking), they interrupt users, preventing them from doing anything else until they interact with the dialog. Flyouts provide a less jarring experience, but displaying too many flyouts can be distracting.

Once you've determined that you want to use a dialog or flyout, you need to choose which one to use.

Given that dialogs block interactions and flyouts do not, dialogs should be reserved for situations where you want the user to drop everything to focus on a specific bit of information or answer a question. Flyouts, on the other hand, can be used when you want to call attention to something, but it's ok if the user wants to ignore it.

Use a dialog for...

- Expressing important information that the user **must** read and acknowledge before proceeding. Examples include:
  - When the user's security might be compromised
  - When the user is about to permanently alter a valuable asset
  - When the user is about to delete a valuable asset
  - To confirm an in-app purchase
- Error messages that apply to the overall app context, such as a connectivity error.
- Questions, when the app needs to ask the user a blocking question, such as when the app can't choose on the user's behalf. A blocking question can't be ignored or postponed, and should offer the user well-defined choices.

Use a flyout for...

- Collecting additional information needed before an action can be completed.
- Displaying info that's only relevant some of the time. For example, in a photo gallery app, when the user clicks an image thumbnail, you might use a flyout to display a large version of the image.
- Displaying more information, such as details or longer descriptions of an item on the page.

Ways to avoid using dialogs and flyouts

Consider the importance of the information you want to share: is it important enough to interrupt the user? Also consider how frequently the information needs to be shown; if you're showing a dialog or notification every few minutes, you might want to allocate space for this info in the primary UI instead. For example, in a chat client, rather than showing a flyout every time a friend logs in, you might display a list of friends who are online at the moment and highlight friends as they log on.

Dialogs are frequently used to confirm an action (such as deleting a file) before executing it. If you expect the user to perform a particular action frequently, consider providing a way for the user to undo the action if it was a mistake, rather than forcing users to confirm the action every time.

Examples

✔ **Important APIs**: ContentDialog class, Flyout class

> - If you have the **WinUI 3 Gallery** app installed, click here to open the app and see the ContentDialog or Flyout in action. Get the app from the Microsoft Store or get the source code on GitHub.

How to create a dialog

See the Dialogs article.

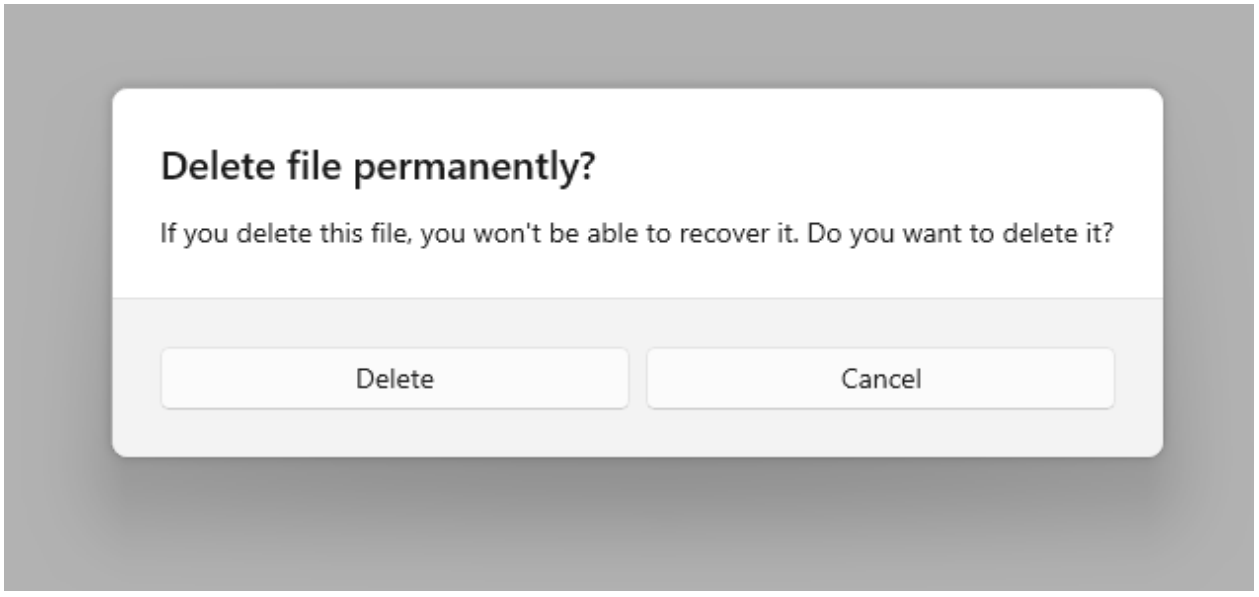
How to create a flyout

See the Flyout article.

Dialog controls

Article • 02/26/2025

Dialog controls are modal UI overlays that provide contextual app information. They block interactions with the app window until being explicitly dismissed. They often request some kind of action from the user.



Is this the right control?

Use dialogs to notify users of important information or to request confirmation or additional info before an action can be completed.

For recommendations on when to use a dialog vs. when to use a flyout (a similar control), see Dialogs and flyouts.

General guidelines

- Clearly identify the issue or the user's objective in the first line of the dialog's text.
- The dialog title is the main instruction and is optional.
  - Use a short title to explain what people need to do with the dialog.
  - If you're using the dialog to deliver a simple message, error or question, you can optionally omit the title. Rely on the content text to deliver that core information.
  - Make sure that the title relates directly to the button choices.
- The dialog content contains the descriptive text and is required.
  - Present the message, error, or blocking question as simply as possible.

    - If a dialog title is used, use the content area to provide more detail or define terminology. Don't repeat the title with slightly different wording.
- At least one dialog button must appear.
    - Ensure that your dialog has at least one button corresponding to a safe, nondestructive action like "Got it!", "Close", or "Cancel". Use the CloseButton API to add this button.
    - Use specific responses to the main instruction or content as button text. An example is, "Do you want to allow AppName to access your location?", followed by "Allow" and "Block" buttons. Specific responses can be understood more quickly, resulting in efficient decision making.
    - Ensure that the text of the action buttons is concise. Short strings enable the user to make a choice quickly and confidently.
    - In addition to the safe, nondestructive action, you may optionally present the user with one or two action buttons related to the main instruction. These "do it" action buttons confirm the main point of the dialog. Use the PrimaryButton and SecondaryButton APIs to add these "do it" actions.
    - The "do it" action button(s) should appear as the leftmost buttons. The safe, nondestructive action should appear as the rightmost button.
    - You may optionally choose to differentiate one of the three buttons as the dialog's default button. Use the DefaultButton API to differentiate one of the buttons.
- Don't use dialogs for errors that are contextual to a specific place on the page, such as validation errors (in password fields, for example), use the app's canvas itself to show inline errors.
- Use the ContentDialog class to build your dialog experience. Don't use the deprecated MessageDialog API.

Create a dialog

✔ **Important APIs**: ContentDialog class

Open the WinUI 3 Gallery app and see the ContentDialog in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

To create a dialog, you use the ContentDialog class. You can create a dialog in code or markup. Although its usually easier to define UI elements in XAML, in the case of a simple dialog, it's actually easier to just use code. This example creates a dialog to notify

the user that there's no WiFi connection, and then uses the ShowAsync method to display it.

C#

```
private async void DisplayNoWifiDialog()
{
    ContentDialog noWifiDialog = new ContentDialog
    {
        Title = "No wifi connection",
        Content = "Check your connection and try again.",
        CloseButtonText = "OK"
    };

    ContentDialogResult result = await noWifiDialog.ShowAsync();
}
```

When the user clicks a dialog button, the ShowAsync method returns a ContentDialogResult to let you know which button the user clicks.

The dialog in this example asks a question and uses the returned ContentDialogResult to determine the user's response.

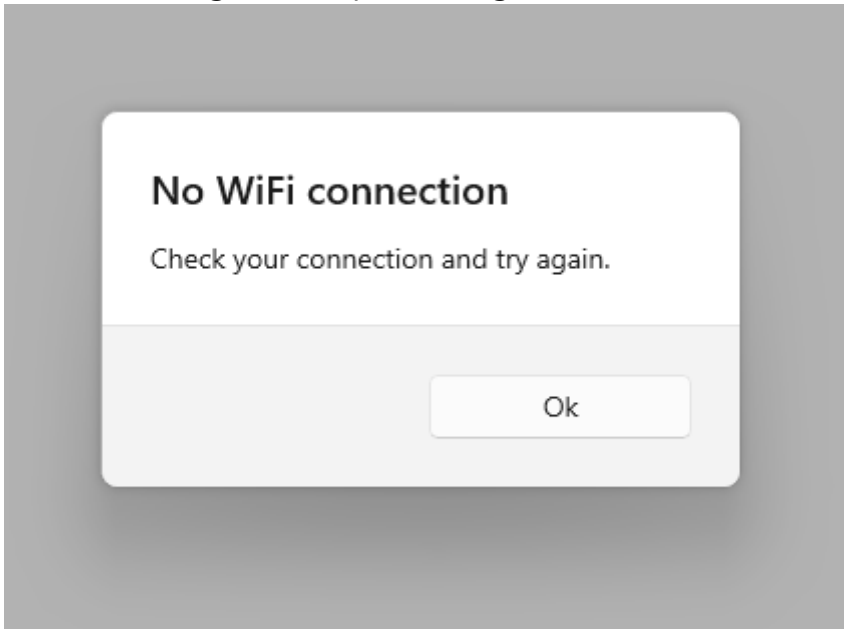
C#

```
private async void DisplayDeleteFileDialog()
{
    ContentDialog deleteFileDialog = new ContentDialog
    {
        Title = "Delete file permanently?",
        Content = "If you delete this file, you won't be able to recover it.
Do you want to delete it?",
        PrimaryButtonText = "Delete",
        CloseButtonText = "Cancel"
    };

    ContentDialogResult result = await deleteFileDialog.ShowAsync();

    // Delete the file if the user clicked the primary button.
    /// Otherwise, do nothing.
    if (result == ContentDialogResult.Primary)
    {
        // Delete the file.
    }
    else
    {
        // The user clicked the CloseButton, pressed ESC, Gamepad B, or the
system back button.
        // Do nothing.
```

```
    }
}
```

Provide a safe action

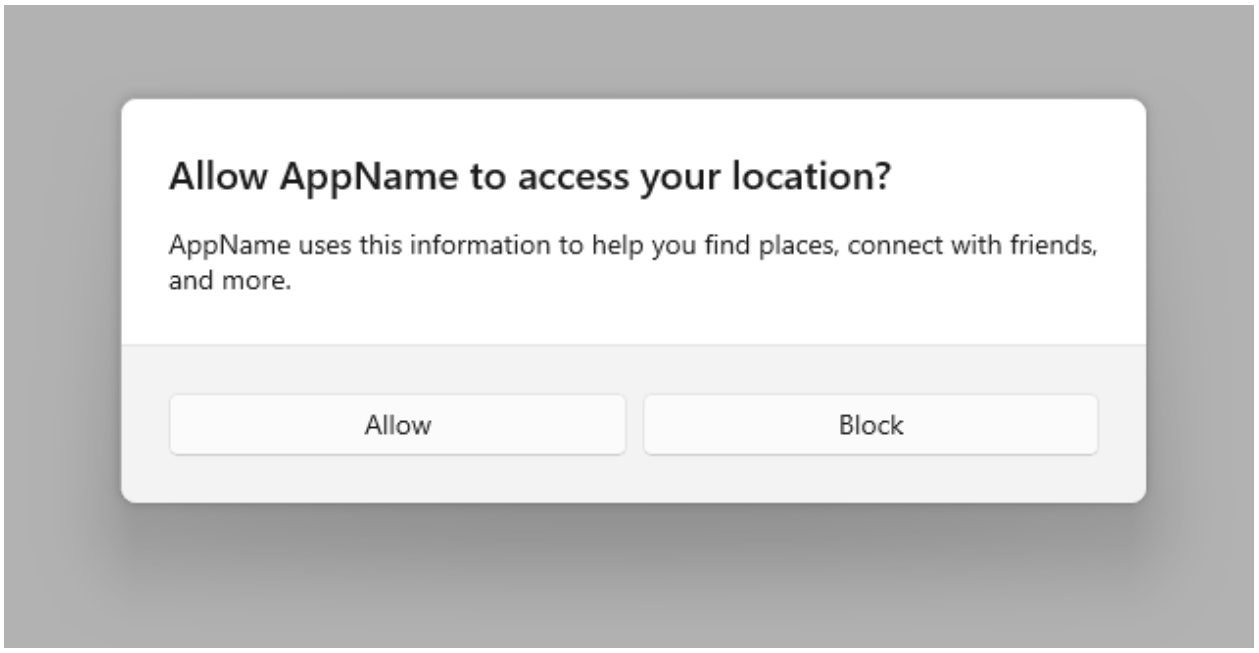
Because dialogs block user interaction, and because buttons are the primary mechanism for users to dismiss the dialog, ensure that your dialog contains at least one "safe" and nondestructive button such as "Close" or "Got it!". **All dialogs should contain at least one safe action button to close the dialog.** This ensures that the user can confidently close the dialog without performing an action.



C#

```
private async void DisplayNoWifiDialog()
{
    ContentDialog noWifiDialog = new ContentDialog
    {
        Title = "No wifi connection",
        Content = "Check your connection and try again.",
        CloseButtonText = "OK"
    };

    ContentDialogResult result = await noWifiDialog.ShowAsync();
}
```

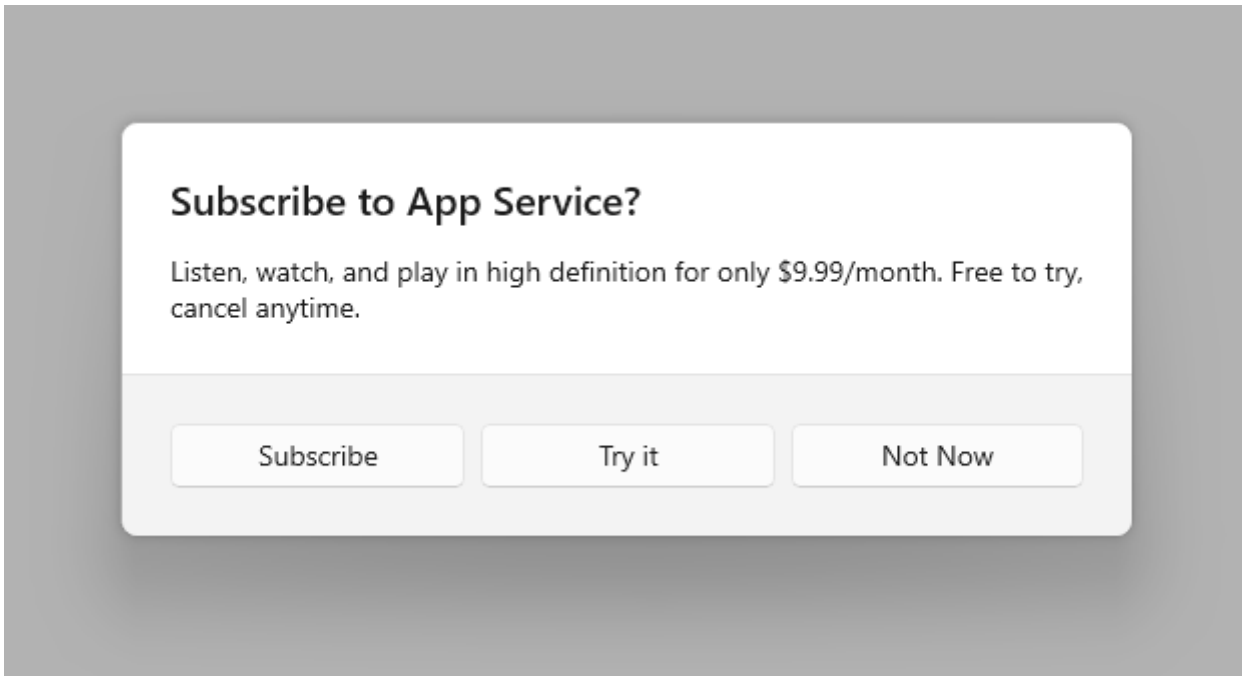
When dialogs are used to display a blocking question, your dialog should present the user with action buttons related to the question. The "safe" and nondestructive button may be accompanied by one or two "do it" action buttons. When presenting the user with multiple options, ensure that the buttons clearly explain the "do it" and safe/"don't do it" actions related to the question proposed.



C#

```
private async void DisplayLocationPromptDialog()
{
    ContentDialog locationPromptDialog = new ContentDialog
    {
        Title = "Allow AppName to access your location?",
        Content = "AppName uses this information to help you find places,
connect with friends, and more.",
        CloseButtonText = "Block",
        PrimaryButtonText = "Allow"
    };

    ContentDialogResult result = await locationPromptDialog.ShowAsync();
}
```

Three button dialogs are used when you present the user with two "do it" actions and a "don't do it" action. Three button dialogs should be used sparingly with clear distinctions between the secondary action and the safe/close action.



C#

```
private async void DisplaySubscribeDialog()
{
    ContentDialog subscribeDialog = new ContentDialog
    {
        Title = "Subscribe to App Service?",
        Content = "Listen, watch, and play in high definition for only
$9.99/month. Free to try, cancel anytime.",
        CloseButtonText = "Not Now",
        PrimaryButtonText = "Subscribe",
        SecondaryButtonText = "Try it"
    };

    ContentDialogResult result = await subscribeDialog.ShowAsync();
}
```

The three dialog buttons

ContentDialog has three different types of buttons that you can use to build a dialog experience.

- **CloseButton** - Required - Represents the safe, nondestructive action that enables the user to exit the dialog. Appears as the rightmost button.
- **PrimaryButton** - Optional - Represents the first "do it" action. Appears as the leftmost button.
- **SecondaryButton** - Optional - Represents the second "do it" action. Appears as the middle button.

Using the built-in buttons will position the buttons appropriately, ensure that they correctly respond to keyboard events, ensure that the command area remains visible even when the on-screen keyboard is up, and will make the dialog look consistent with other dialogs.

CloseButton

Every dialog should contain a safe, nondestructive action button that enables the user to confidently exit the dialog.

Use the ContentDialog.CloseButton API to create this button. This allows you to create the right user experience for all inputs including mouse, keyboard, touch, and gamepad. This experience will happen when:

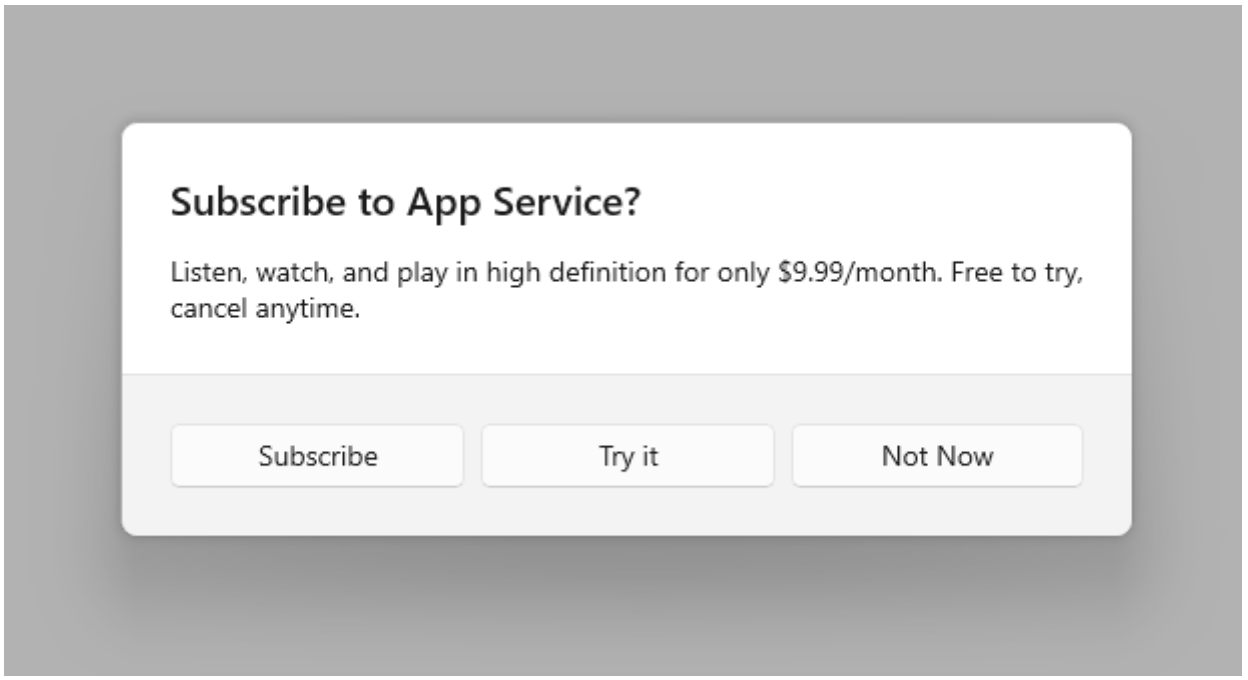
- The user clicks or taps on the CloseButton.
- The user presses the system back button.
- The user presses the ESC button on the keyboard.
- The user presses Gamepad B.

When the user clicks a dialog button, the ShowAsync method returns a ContentDialogResult to let you know which button the user clicks. Pressing on the CloseButton returns ContentDialogResult.None.

PrimaryButton and SecondaryButton

In addition to the CloseButton, you may optionally present the user with one or two action buttons related to the main instruction. Leverage PrimaryButton for the first "do it" action, and SecondaryButton for the second "do it" action. In three-button dialogs, the PrimaryButton generally represents the affirmative "do it" action, while the SecondaryButton generally represents a neutral or secondary "do it" action. For example, an app may prompt the user to subscribe to a service. The PrimaryButton as the affirmative "do it" action would host the Subscribe text, while the SecondaryButton as the neutral "do it" action would host the Try it text. The CloseButton would allow the user to cancel without performing either action.

When the user clicks on the PrimaryButton, the ShowAsync method returns ContentDialogResult.Primary. When the user clicks on the SecondaryButton, the ShowAsync method returns ContentDialogResult.Secondary.

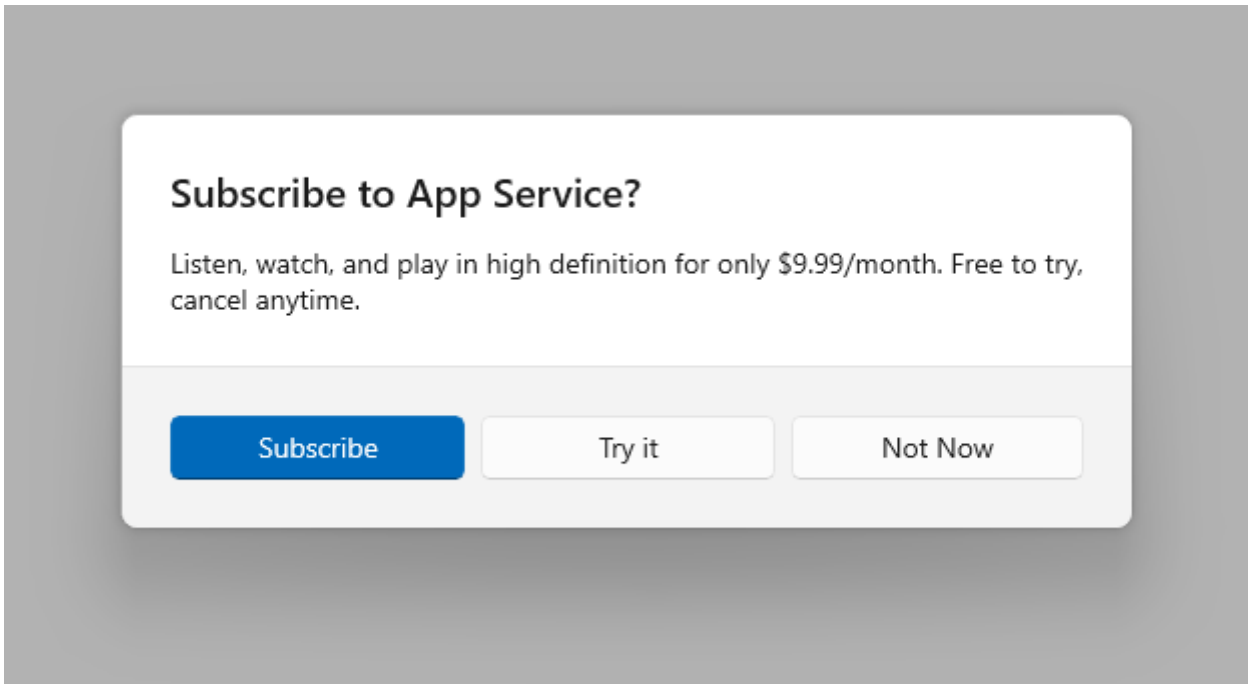


DefaultButton

You may optionally choose to differentiate one of the three buttons as the default button. Specifying the default button causes the following to happen:

- The button receives the Accent Button visual treatment
- The button will respond to the ENTER key automatically
  - When the user presses the ENTER key on the keyboard, the click handler associated with the Default Button will fire and the ContentDialogResult will return the value associated with the Default Button
  - If the user has placed Keyboard Focus on a control that handles ENTER, the Default Button will not respond to ENTER presses
- The button will receive focus automatically when the Dialog is opened unless the dialog's content contains focusable UI

Use the ContentDialog.DefaultButton property to indicate the default button. By default, no default button is set.



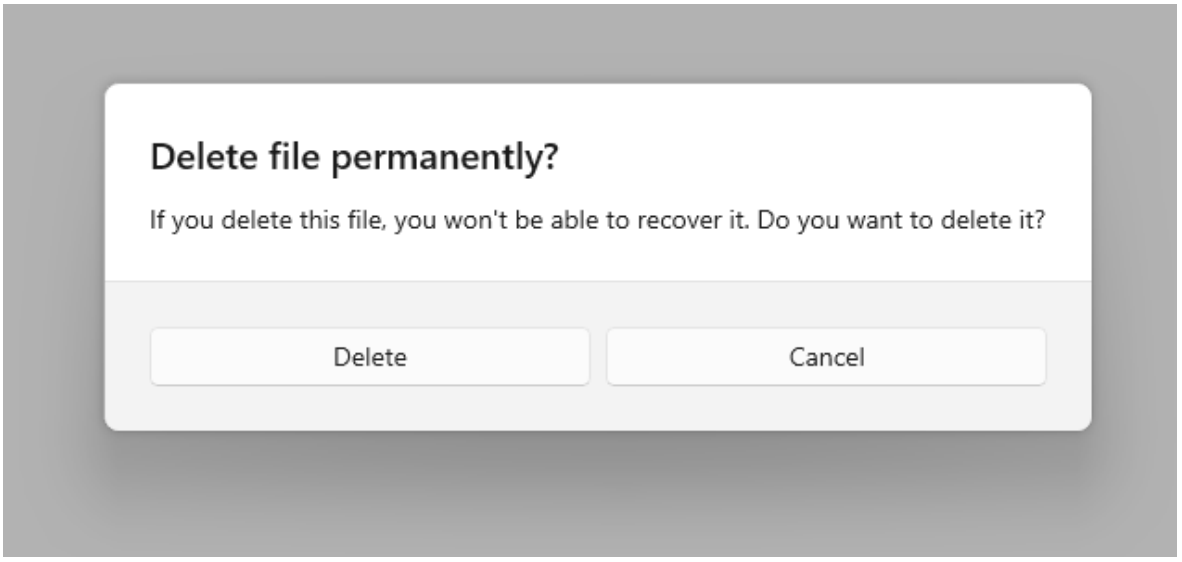
C#

```
private async void DisplaySubscribeDialog()
{
    ContentDialog subscribeDialog = new ContentDialog
    {
        Title = "Subscribe to App Service?",
        Content = "Listen, watch, and play in high definition for only
$9.99/month. Free to try, cancel anytime.",
        CloseButtonText = "Not Now",
        PrimaryButtonText = "Subscribe",
        SecondaryButtonText = "Try it",
        DefaultButton = ContentDialogButton.Primary
    };

    ContentDialogResult result = await subscribeDialog.ShowAsync();
}
```

Confirmation dialogs (OK/Cancel)

A confirmation dialog gives users the chance to confirm that they want to perform an action. They can affirm the action, or choose to cancel. A typical confirmation dialog has two buttons: an affirmation ("OK") button and a cancel button.

- In general, the affirmation button should be on the left (the primary button) and the cancel button (the secondary button) should be on the right.



- As noted in the general recommendations section, use buttons with text that identifies specific responses to the main instruction or content.

> Some platforms put the affirmation button on the right instead of the left. So why do we recommend putting it on the left? If you assume that the majority of users are right-handed and they hold their phone with that hand, it's actually more comfortable to press the affirmation button when it's on the left, because the button is more likely to be within the user's thumb-arc. Buttons on the right-side of the screen require the user to pull their thumb inward into a less-comfortable position.

ContentDialog in AppWindow or Xaml Islands

> NOTE: This section applies only to apps that target Windows 10, version 1903 or later. AppWindow and XAML Islands are not available in earlier versions. For more info about versioning, see Version adaptive apps.

By default, content dialogs display modally relative to the root ApplicationView. When you use ContentDialog inside of either an AppWindow or a XAML Island, you need to manually set the XamlRoot on the dialog to the root of the XAML host.

To do so, set the ContentDialog's XamlRoot property to the same XamlRoot as an element already in the AppWindow or XAML Island, as shown here.

C#

```
private async void DisplayNoWifiDialog()
{
    ContentDialog noWifiDialog = new ContentDialog
    {
        Title = "No wifi connection",
```

```
        Content = "Check your connection and try again.",
        CloseButtonText = "OK"
    };

    // Use this code to associate the dialog to the appropriate AppWindow by
setting
    // the dialog's XamlRoot to the same XamlRoot as an element that is
already present in the AppWindow.
    if
(ApiInformation.IsApiContractPresent("Windows.Foundation.UniversalApiContrac
t", 8))
    {
        noWifiDialog.XamlRoot = elementAlreadyInMyAppWindow.XamlRoot;
    }

    ContentDialogResult result = await noWifiDialog.ShowAsync();
}
```

> **⚠ Warning**
>
> There can only be one ContentDialog open per thread at a time. Attempting to open two ContentDialogs will throw an exception, even if they are attempting to open in separate AppWindows.

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** ContentDialog class
- ✓ Open the WinUI 2 Gallery app and see the ContentDialog in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related articles

- Tooltips
- Menus and context menu
- Flyout class
- ContentDialog class

Flyouts

Article • 02/26/2025

A flyout is a light dismiss container that can show arbitrary UI as its content. Flyouts can contain other flyouts or context menus to create a nested experience.

Is this the right control?

- Don't use a flyout instead of tooltip or context menu. Use a tooltip to show a short description that hides after a specified time. Use a context menu for contextual actions related to a UI element, such as copy and paste.

For recommendations on when to use a flyout vs. when to use a dialog (a similar control), see Dialogs and flyouts.

Create a flyout

✓ **Important APIs**: Flyout class, Button.Flyout property

Open the WinUI 3 Gallery app and see the Flyout in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Flyouts are attached to specific controls. You can use the Placement property to specify where a flyout appears: Top, Left, Bottom, Right, or Full. If you select the Full placement mode, the app stretches the flyout and centers it inside the app window. Some controls, such as Button, provide a Flyout property that you can use to associate a flyout or context menu.

This example creates a simple flyout that displays some text when the button is pressed.

XAML

```
<Button Content="Click me">
  <Button.Flyout>
     <Flyout>
        <TextBlock Text="This is a flyout!"/>
     </Flyout>
```

```
  </Button.Flyout>
</Button>
```

If the control doesn't have a flyout property, you can use the FlyoutBase.AttachedFlyout attached property instead. When you do this, you also need to call the FlyoutBase.ShowAttachedFlyout method to show the flyout.

This example adds a simple flyout to an image. When the user taps the image, the app shows the flyout.

XAML

```
<Image Source="Assets/cliff.jpg" Width="50" Height="50"
  Margin="10" Tapped="Image_Tapped">
  <FlyoutBase.AttachedFlyout>
    <Flyout>
      <TextBlock Text="This is some text in a flyout."  />
    </Flyout>
  </FlyoutBase.AttachedFlyout>
</Image>
```

C#

```
private void Image_Tapped(object sender, TappedRoutedEventArgs e)
{
    FlyoutBase.ShowAttachedFlyout((FrameworkElement)sender);
}
```

The previous examples defined their flyouts inline. You can also define a flyout as a static resource and then use it with multiple elements. This example creates a more complicated flyout that displays a larger version of an image when its thumbnail is tapped.

XAML

```
<!-- Declare the shared flyout as a resource. -->
<Page.Resources>
    <Flyout x:Key="ImagePreviewFlyout" Placement="Right">
        <!-- The flyout's DataContext must be the Image Source
             of the image the flyout is attached to. -->
        <Image Source="{Binding Path=Source}"
            MaxHeight="400" MaxWidth="400" Stretch="Uniform"/>
    </Flyout>
</Page.Resources>
```

XAML

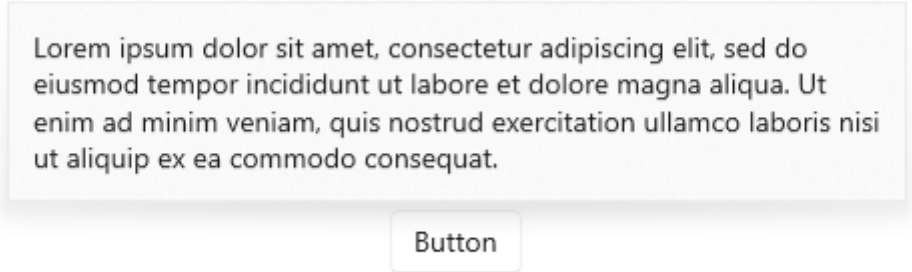
```
<!-- Assign the flyout to each element that shares it. -->
<StackPanel>
    <Image Source="Assets/cliff.jpg" Width="50" Height="50"
           Margin="10" Tapped="Image_Tapped"
           FlyoutBase.AttachedFlyout="{StaticResource ImagePreviewFlyout}"
           DataContext="{Binding RelativeSource={RelativeSource
Mode=Self}}"/>
    <Image Source="Assets/grapes.jpg" Width="50" Height="50"
           Margin="10" Tapped="Image_Tapped"
           FlyoutBase.AttachedFlyout="{StaticResource ImagePreviewFlyout}"
           DataContext="{Binding RelativeSource={RelativeSource
Mode=Self}}"/>
    <Image Source="Assets/rainier.jpg" Width="50" Height="50"
           Margin="10" Tapped="Image_Tapped"
           FlyoutBase.AttachedFlyout="{StaticResource ImagePreviewFlyout}"
           DataContext="{Binding RelativeSource={RelativeSource
Mode=Self}}"/>
</StackPanel>
```

C#

```
private void Image_Tapped(object sender, TappedRoutedEventArgs e)
{
    FlyoutBase.ShowAttachedFlyout((FrameworkElement)sender);
}
```

Style a flyout

To style a Flyout, modify its FlyoutPresenterStyle. This example shows a paragraph of wrapping text and makes the text block accessible to a screen reader.



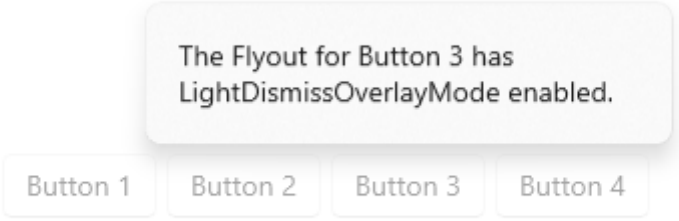
XAML

```
<Flyout>
  <Flyout.FlyoutPresenterStyle>
    <Style TargetType="FlyoutPresenter">
```

```
      <Setter Property="ScrollViewer.HorizontalScrollMode"
          Value="Disabled"/>
      <Setter Property="ScrollViewer.HorizontalScrollBarVisibility"
Value="Disabled"/>
      <Setter Property="IsTabStop" Value="True"/>
      <Setter Property="TabNavigation" Value="Cycle"/>
    </Style>
  </Flyout.FlyoutPresenterStyle>
  <TextBlock Style="{StaticResource BodyTextBlockStyle}" Text="Lorem ipsum
dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat."/>
</Flyout>
```

Styling flyouts for 10-foot experiences

Light dismiss controls like flyout trap keyboard and gamepad focus inside their transient UI until dismissed. To provide a visual cue for this behavior, light dismiss controls on Xbox draw an overlay that dims the contrast and visibility of out of scope UI. This behavior can be modified with the LightDismissOverlayMode property. By default, flyouts will draw the light dismiss overlay on Xbox but not other device families, but apps can choose to force the overlay to be always **On** or always **Off**.



XAML

```
<MenuFlyout LightDismissOverlayMode="On">
```

Light dismiss behavior

Flyouts can be closed with a quick light dismiss action, including

- Tap outside the flyout
- Press the Escape keyboard key

- Press the hardware or software system Back button
- Press the gamepad B button

When dismissing with a tap, this gesture is typically absorbed and not passed on to the UI underneath. For example, if there's a button visible behind an open flyout, the user's first tap dismisses the flyout but does not activate this button. Pressing the button requires a second tap.

You can change this behavior by designating the button as an input pass-through element for the flyout. The flyout will close as a result of the light dismiss actions described above and will also pass the tap event to its designated `OverlayInputPassThroughElement`. Consider adopting this behavior to speed up user interactions on functionally similar items. If your app has a favorites collection and each item in the collection includes an attached flyout, it's reasonable to expect that users may want to interact with multiple flyouts in rapid succession.

> **Note**
>
> Be careful not to designate an overlay input pass-through element which results in a destructive action. Users have become habituated to discreet light dismiss actions which do not activate primary UI. Close, Delete, or similarly destructive buttons should not activate on light dismiss to avoid unexpected and disruptive behavior.

In the following example, all three buttons inside FavoritesBar will be activated on the first tap.

XAML

```
<Page>
    <Page.Resources>
        <Flyout x:Name="TravelFlyout" x:Key="TravelFlyout"
                OverlayInputPassThroughElement="{x:Bind FavoritesBar}">
            <StackPanel>
                <HyperlinkButton Content="Washington Trails Association"/>
                <HyperlinkButton Content="Washington Cascades - Go
Northwest! A Travel Guide"/>
            </StackPanel>
        </Flyout>
    </Page.Resources>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <StackPanel x:Name="FavoritesBar" Orientation="Horizontal">
```

```
            <HyperlinkButton x:Name="PageLinkBtn">Bing</HyperlinkButton>
            <Button x:Name="Folder1" Content="Travel" Flyout="
{StaticResource TravelFlyout}"/>
            <Button x:Name="Folder2" Content="Entertainment"
Click="Folder2_Click"/>
        </StackPanel>
        <ScrollViewer Grid.Row="1">
            <WebView x:Name="WebContent"/>
        </ScrollViewer>
    </Grid>
</Page>
```

C#

```
private void Folder2_Click(object sender, RoutedEventArgs e)
{
     Flyout flyout = new Flyout();
     flyout.OverlayInputPassThroughElement = FavoritesBar;
     ...
     flyout.ShowAt(sender as FrameworkElement);
{
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** Flyout class, Button.Flyout property
- ✓ Open the WinUI 2 Gallery app and see the Flyout in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses

rounded corners. For more info, see Corner radius.

Related articles

- Tooltips
- Menus and context menu
- Flyout class
- ContentDialog class

Teaching tip

Article • 02/26/2025

A teaching tip is a semi-persistent and content-rich flyout that provides contextual information. It is often used for informing, reminding, and teaching users about important and new features that may enhance their experience.

A teaching tip may be light-dismiss or require explicit action to close. A teaching tip can target a specific UI element with its tail and also be used without a tail or target.

Is this the right control?

Use a **TeachingTip** control to focus a user's attention on new or important updates and features, remind a user of nonessential options that would improve their experience, or teach a user how a task should be completed.

Because teaching tip is transient, it would not be the recommended control for prompting users about errors or important status changes.

Recommendations

- Tips are impermanent and should not contain information or options that are critical to the experience of an application.
- Try to avoid showing teaching tips too often. Teaching tips are most likely to each receive individual attention when they are staggered throughout long sessions or across multiple sessions.
- Keep tips succinct and their topic clear. Research shows users, on average, only read 3-5 words and only comprehend 2-3 words before deciding whether to interact with a tip.
- Gamepad accessibility of a teaching tip is not guaranteed. For applications that predict gamepad input, please see gamepad and remote control interactions. It is encouraged to test gamepad accessibility of each teaching tip using all possible configurations of an app's UI.
- When enabling a teaching tip to escape the xaml root, it is encouraged to also enable the IsLightDismissEnabled property and set the PreferredPlacement mode nearest to the center of the xaml root.

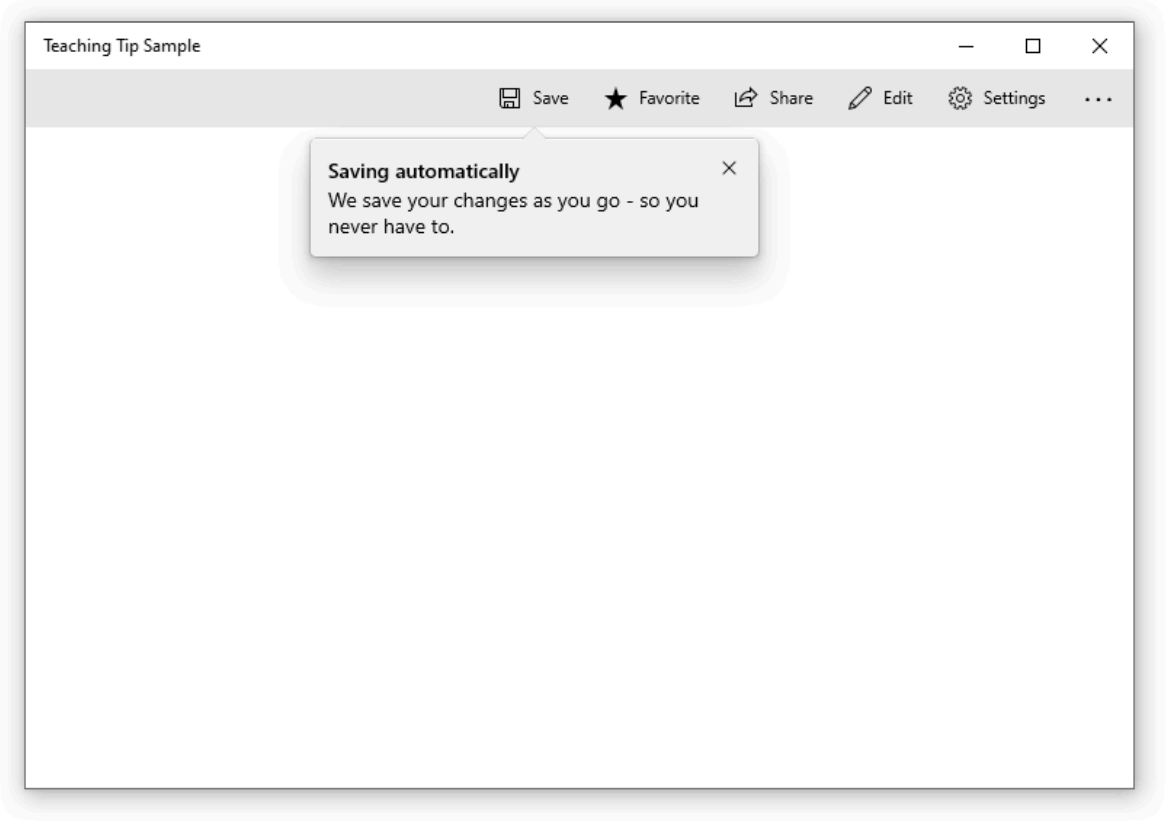
Reconfiguring an open teaching tip

Some content and properties can be reconfigured while the teaching tip is open and will take effect immediately. Other content and properties, such as the icon property, the Action and Close buttons, and reconfiguring between light-dismiss and explicit-dismiss will all require the teaching tip to be closed and reopened for changes to these properties to take affect. Note that changing dismissal behavior from manual-dismiss to light-dismiss while a teaching tip is open will cause the teaching tip to have its Close button removed before the light-dismiss behavior is enabled and the tip can remain stuck on-screen.

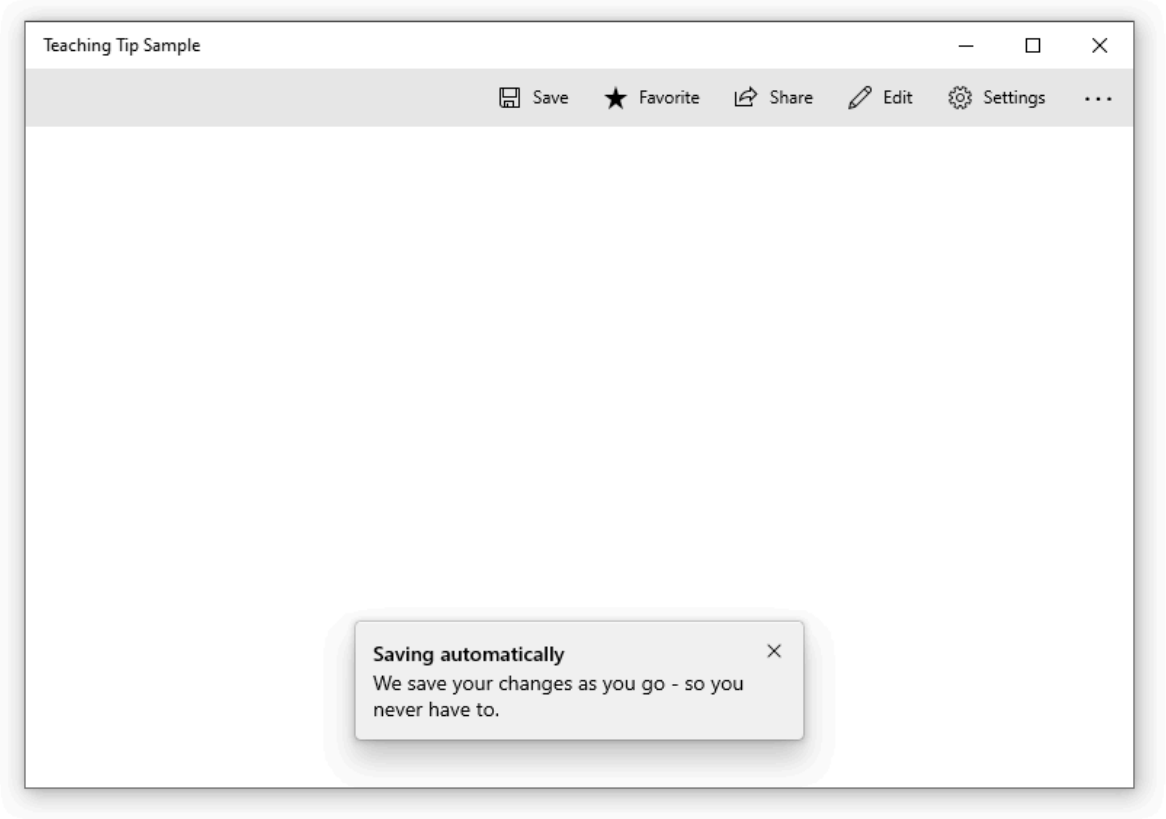
Examples

A teaching tip can have several configurations, including these notable ones:

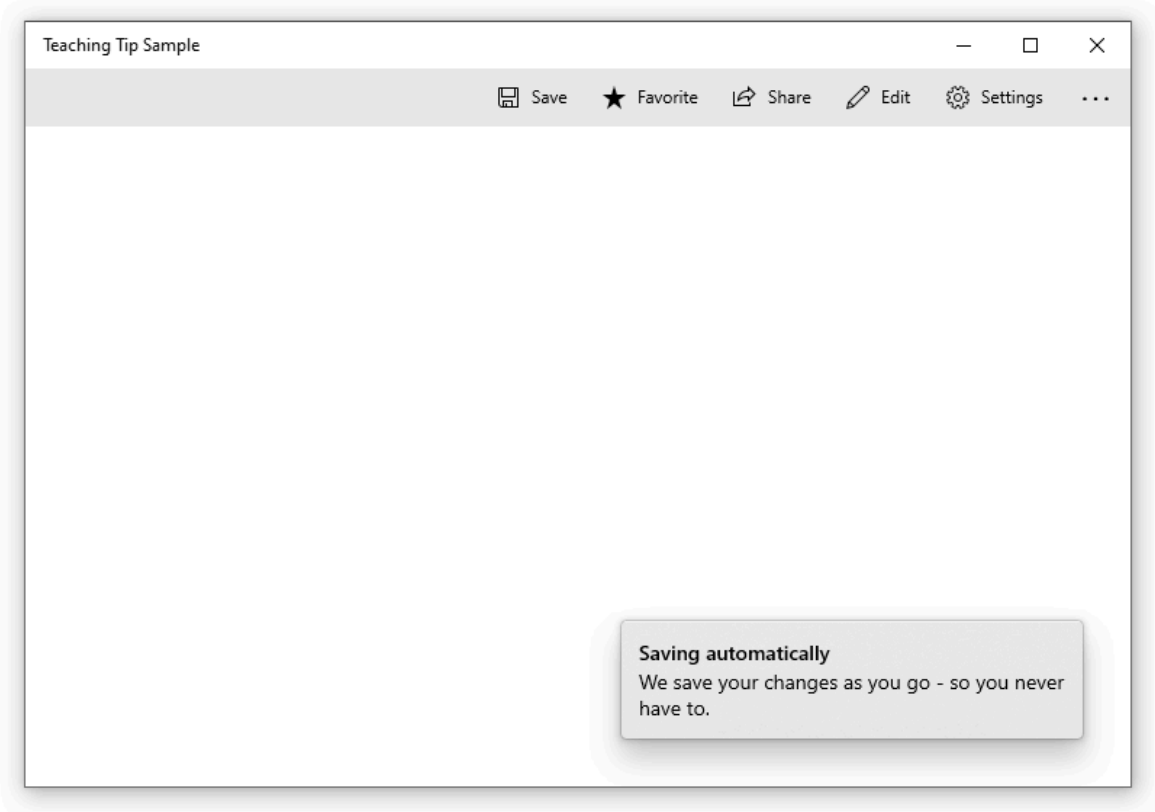
A teaching tip can target a specific UI element with its tail to enhance contextual clarity of the information it is presenting.



When the information presented does not pertain to a particular UI element, a nontargeted teaching tip can be created by removing the tail.



A teaching tip can require the user to dismiss it via an "X" button in a top corner or a "Close" button at the bottom. A teaching tip may also be light-dismiss enabled in which case there is no dismiss button and the teaching tip will instead dismiss when a user scrolls or interacts with other elements of the application. Because of this behavior, light-dismiss tips are the best solution when a tip needs to be placed in a scrollable area.



Create a teaching tip

✔ **Important APIs:** TeachingTip class

Open the WinUI 3 Gallery app and see the TeachingTip in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Here's the XAML for a targeted teaching tip control that demonstrates the default look of the TeachingTip with a title and subtitle. Note that the teaching tip can appear anywhere in the element tree or code behind. In this example below, it's located in a ResourceDictionary.

XAML

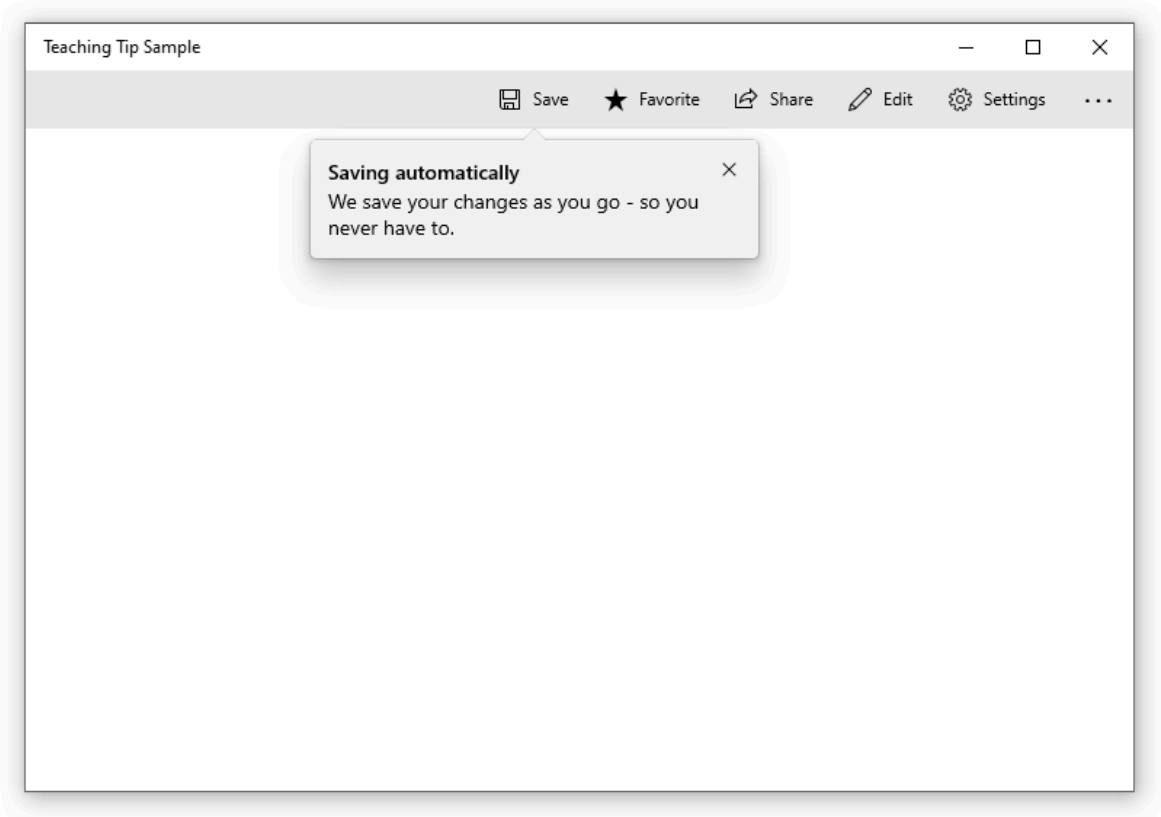
```
<Button x:Name="SaveButton" Content="Save">
    <Button.Resources>
        <TeachingTip x:Name="AutoSaveTip"
            Target="{x:Bind SaveButton}"
            Title="Save automatically"
            Subtitle="When you save your file to OneDrive, we save your
changes as you go - so you never have to.">
```

```
        </TeachingTip>
    </Button.Resources>
</Button>
```

C#

```
public MainPage()
{
    this.InitializeComponent();

    if(!HaveExplainedAutoSave())
    {
        AutoSaveTip.IsOpen = true;
        SetHaveExplainedAutoSave();
    }
}
```

Here's the result when the Page containing the button and teaching tip is shown:



In the example above, the Title and Subtitle properties are used to set the teaching tip's title and subtitle. The Target property is set to the "SaveButton" to establish the visual connection between itself and the button. To show the teaching tip, its IsOpen property is set to `true`.

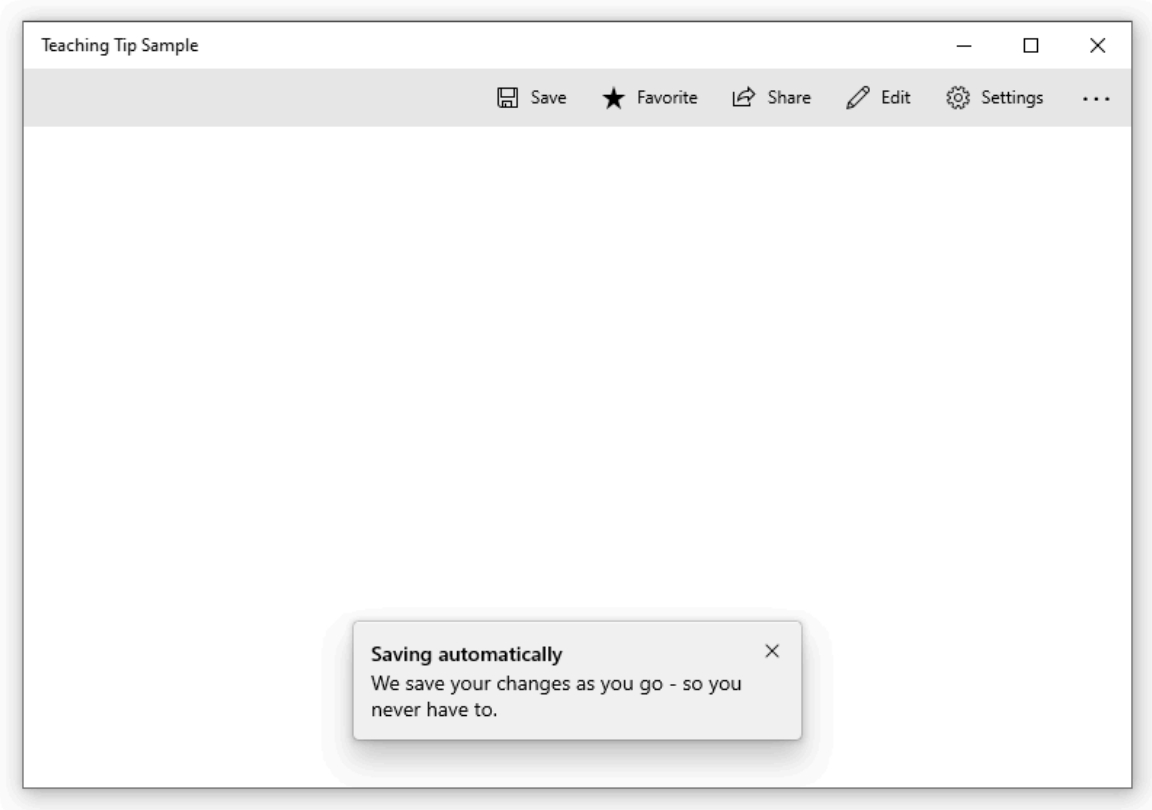
Non-targeted tips

Not all tips relate to an element onscreen. For these scenarios, do not set a target and the teaching tip will instead display relative to the edges of the xaml root. However, a teaching tip can have the tail removed while retaining placement relative to a UI element by setting the TailVisibility property to "Collapsed". The following example is of a non-targeted teaching tip.

XAML

```
<Button x:Name="SaveButton" Content="Save" />

<TeachingTip x:Name="AutoSaveTip"
    Title="Saving automatically"
    Subtitle="We save your changes as you go - so you never have to.">
</TeachingTip>
```

Note that in this example the TeachingTip is in the element tree rather than in a ResourceDictionary or in code behind. This has no effect on behavior; the TeachingTip only displays when opened, and takes up no layout space.



Preferred placement

Teaching tip replicates Flyout's FlyoutPlacementMode placement behavior with the PreferredPlacement property. The default placement mode will try to place a targeted

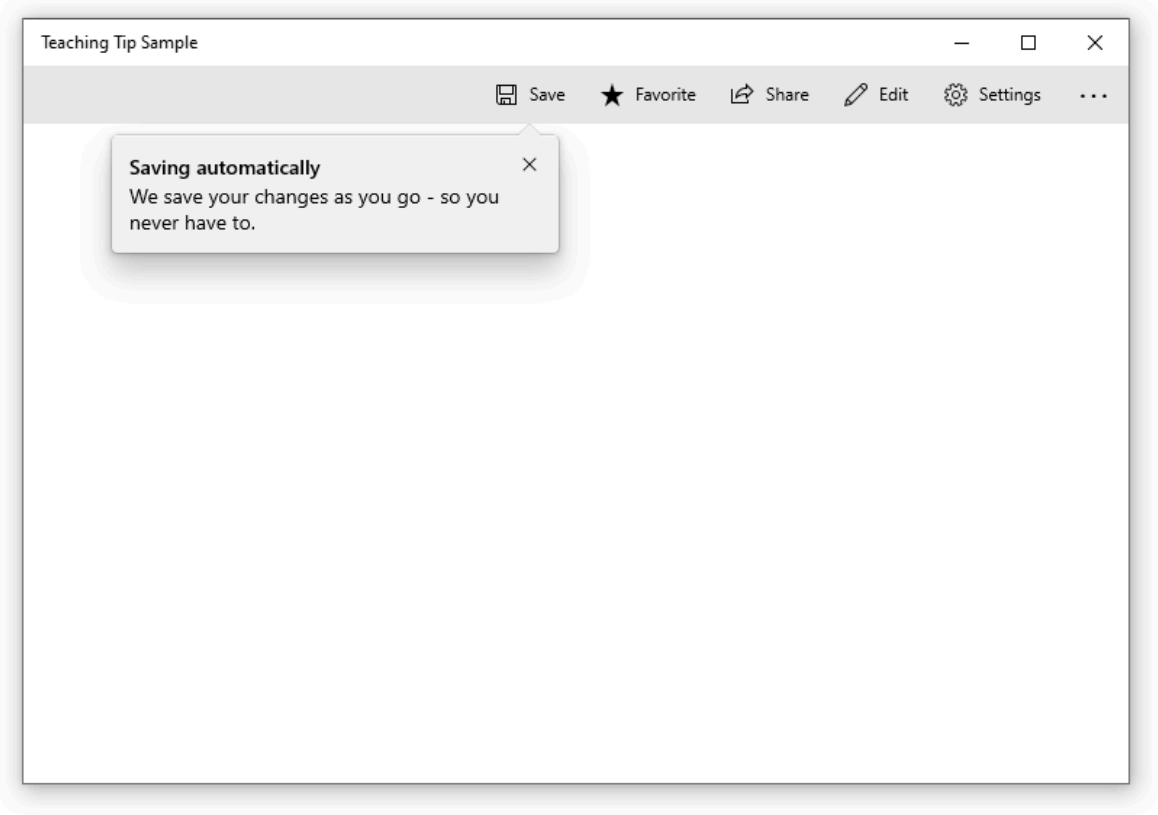
teaching tip above its target and a non-targeted teaching tip centered at the bottom of the xaml root. As with Flyout, if the preferred placement mode would not leave room for the teaching tip to show, another placement mode will be automatically chosen.

For applications that predict gamepad input, please see gamepad and remote control interactions. It is encouraged to test gamepad accessibility of each teaching tip using all possible configurations of an app's UI.

A targeted teaching tip with its PreferredPlacement set to "BottomLeft" will appear with the tail centered at the bottom of its target with the teaching tip's body shifted toward the left.

XAML

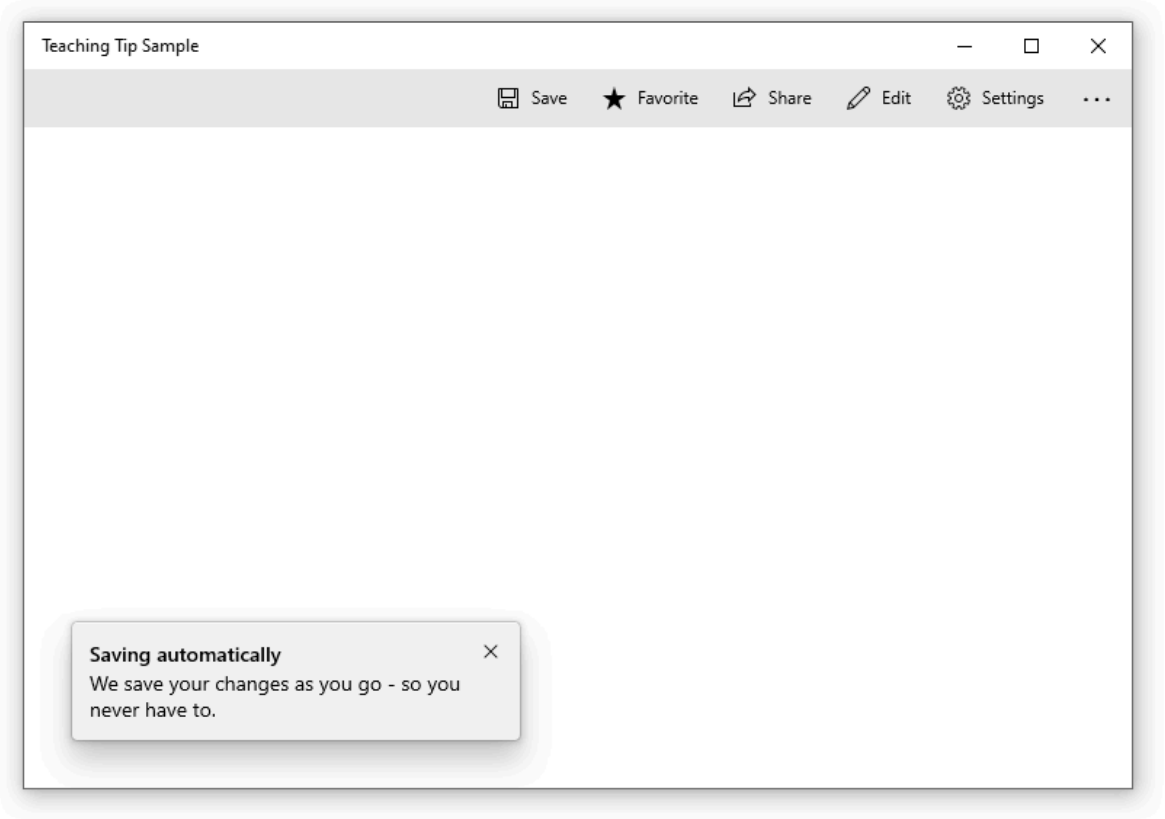
```
<Button x:Name="SaveButton" Content="Save">
    <Button.Resources>
        <TeachingTip x:Name="AutoSaveTip"
            Target="{x:Bind SaveButton}"
            Title="Saving automatically"
            Subtitle="We save your changes as you go - so you never have
to."
            PreferredPlacement="BottomLeft">
        </TeachingTip>
    </Button.Resources>
</Button>
```



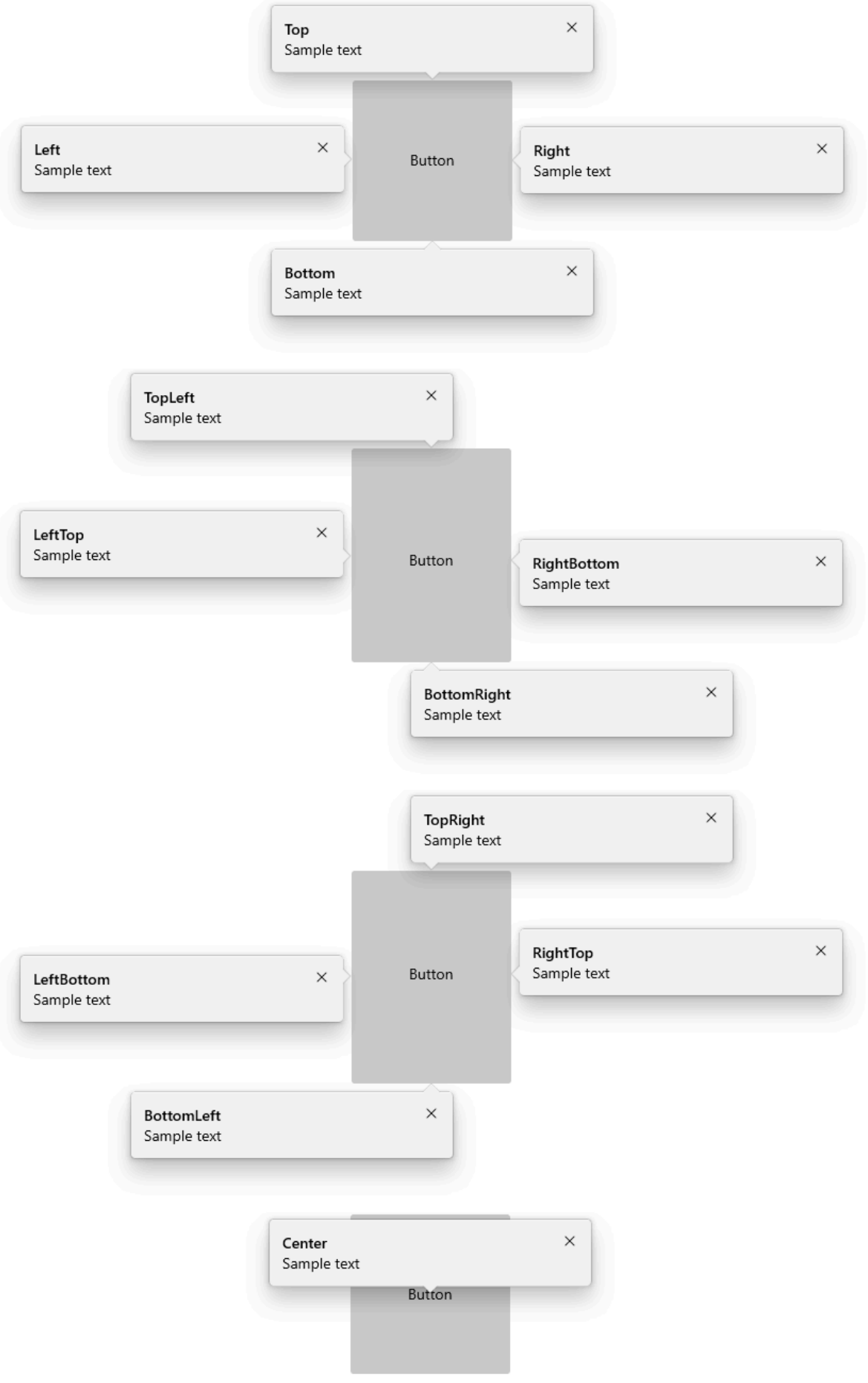
A non-targeted teaching tip with its PreferredPlacement set to "BottomLeft" will appear in the bottom left corner of the xaml root.

XAML

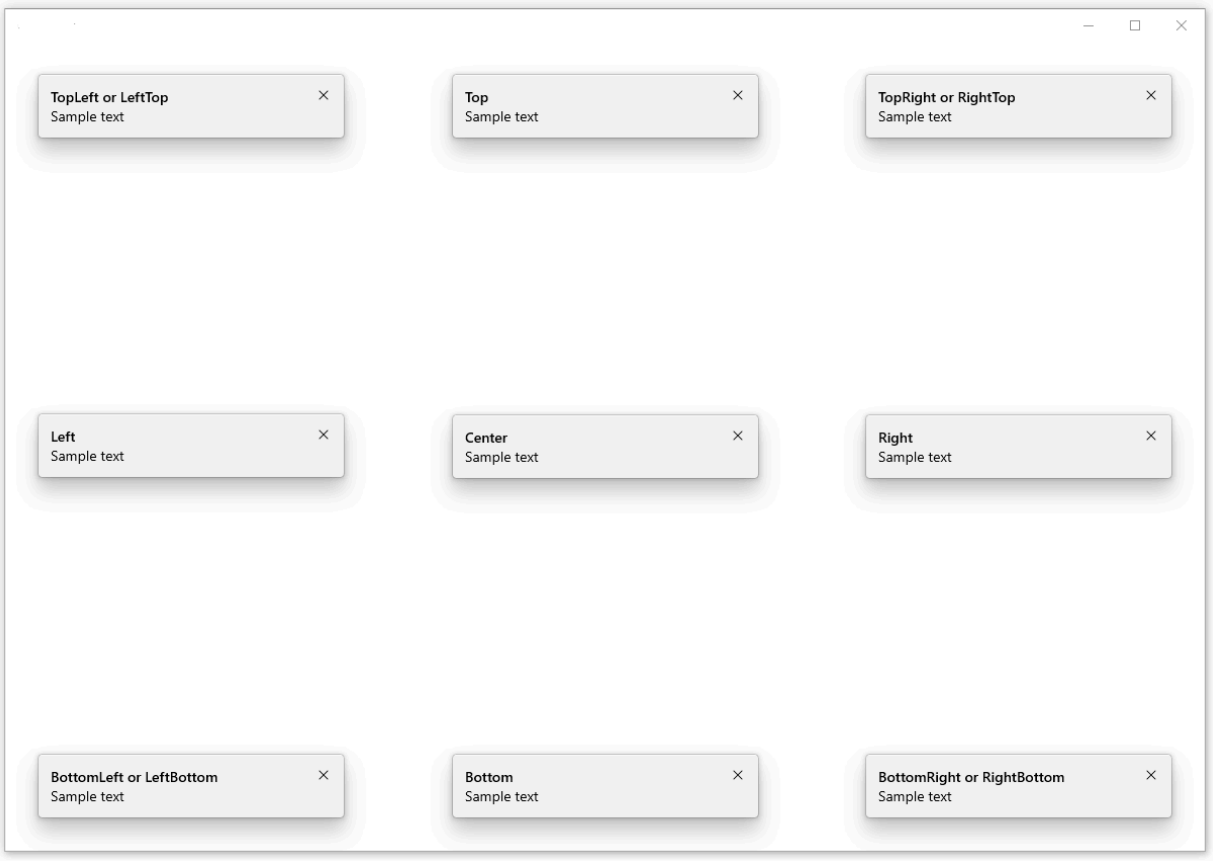
```
<Button x:Name="SaveButton" Content="Save" />

<TeachingTip x:Name="AutoSaveTip"
    Title="Saving automatically"
    Subtitle="We save your changes as you go - so you never have to."
    PreferredPlacement="BottomLeft">
</TeachingTip>
```



The diagram below depicts the result of all 13 PreferredPlacement modes that can be set for targeted teaching tips.


Top
Sample text
Left
Sample text
Button
Right
Sample text
Bottom
Sample text
TopLeft
Sample text
LeftTop
Sample text
Button
RightBottom
Sample text
BottomRight
Sample text
TopRight
Sample text
LeftBottom
Sample text
Button
RightTop
Sample text
BottomLeft
Sample text
Center
Sample text
Button

The diagram below depicts the result of all 13 PreferredPlacement modes that can be set for non-targeted teaching tips.



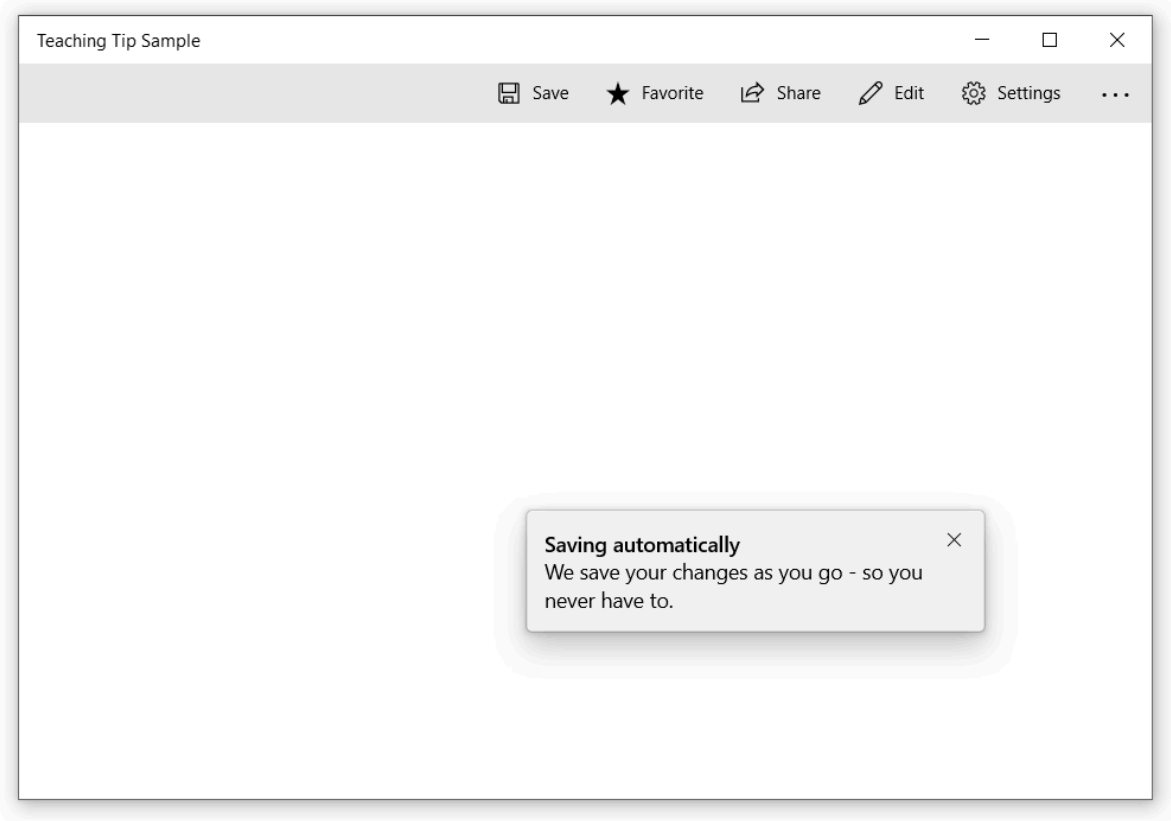
Add a placement margin

You can control how far a targeted teaching tip is set apart from its target and how far a non-targeted teaching tip is set apart from the edges of the xaml root by using the PlacementMargin property. Like Margin, PlacementMargin has four values – left, right, top, and bottom – so only the relevant values are used. For example, PlacementMargin.Left applies when the tip is left of the target or on the left edge of the xaml root.

The following example shows a non-targeted tip with the PlacementMargin's Left/Top/Right/Bottom all set to 80.

XAML

```
<Button x:Name="SaveButton" Content="Save" />

<TeachingTip x:Name="AutoSaveTip"
    Title="Saving automatically"
    Subtitle="We save your changes as you go - so you never have to."
    PreferredPlacement="BottomLeft"
    PlacementMargin="80">
</TeachingTip>
```

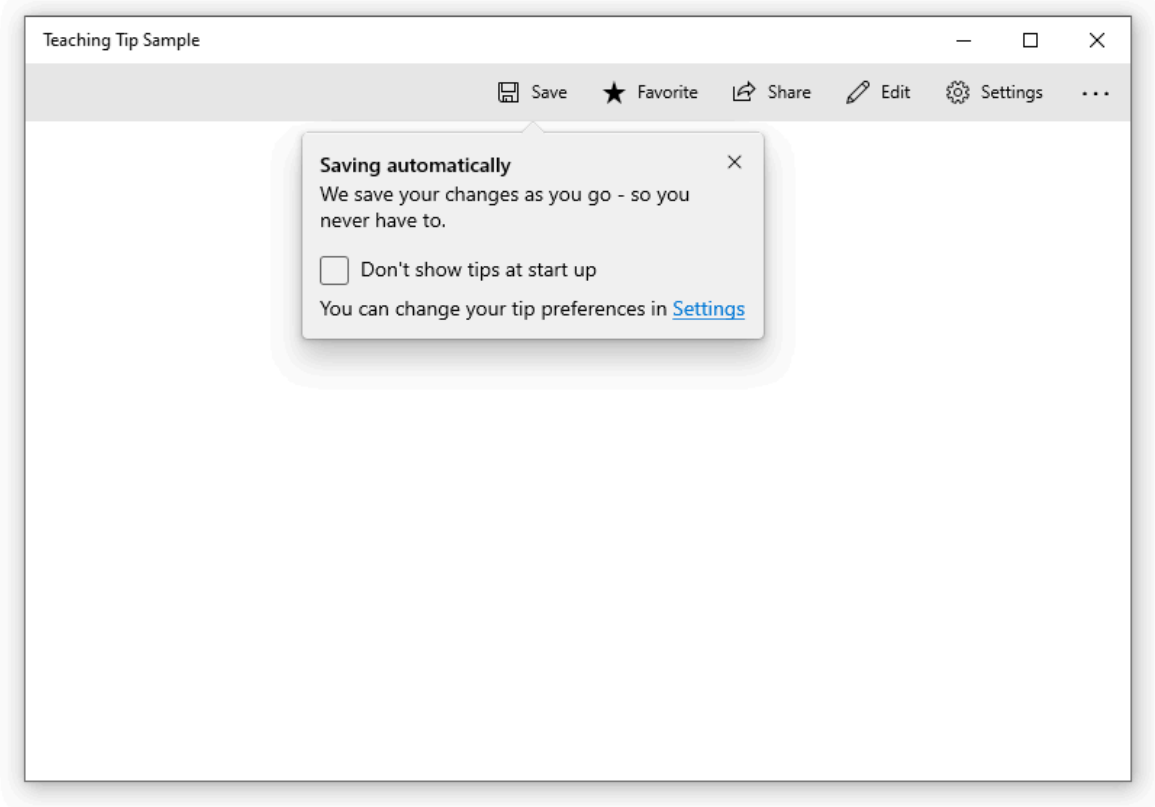


Add content

Content can be added to a teaching tip using the Content property. If there is more content to show than what the size of a teaching tip will allow, a scrollbar will be automatically enabled to allow a user to scroll the content area.

XAML

```
<Button x:Name="SaveButton" Content="Save">
    <Button.Resources>
        <TeachingTip x:Name="AutoSaveTip"
            Target="{x:Bind SaveButton}"
            Title="Saving automatically"
            Subtitle="We save your changes as you go - so you never have
to.">
                <StackPanel>
                    <CheckBox x:Name="HideTipsCheckBox" Content="Don't show
tips at start up" IsChecked="{x:Bind HidingTips, Mode=TwoWay}" />
                    <TextBlock>You can change your tip preferences in
<Hyperlink NavigateUri="app:/item/SettingsPage">Settings</Hyperlink> if you
change your mind.</TextBlock>
                </StackPanel>
        </TeachingTip>
    </Button.Resources>
</Button>
```



Add buttons

By default, a standard "X" close button is shown next to the title of a teaching tip. The Close button can be customized with the CloseButtonContent property, in which case the button is moved to the bottom of the teaching tip.

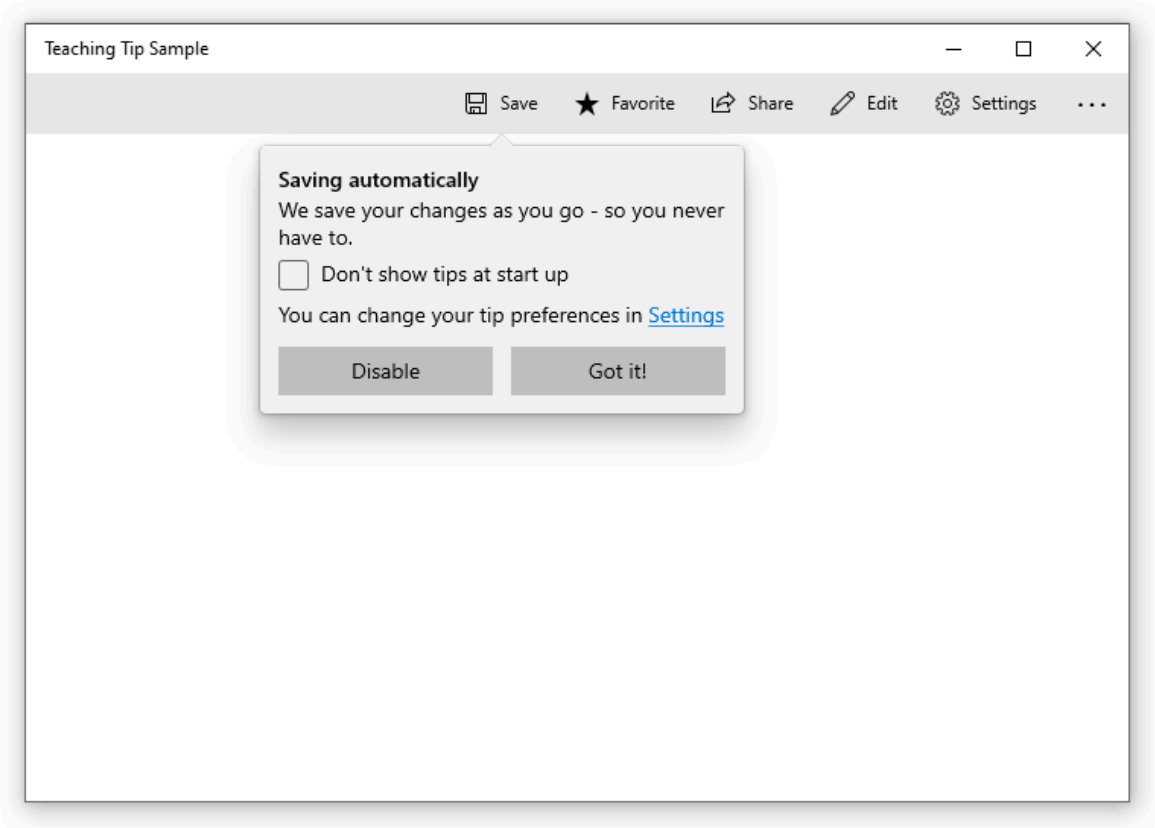
Note: No close button will appear on light-dismiss enabled tips

A custom action button can be added by setting ActionButtonContent property (and optionally the ActionButtonCommand and the ActionButtonCommandParameter properties).

XAML

```
<Button x:Name="SaveButton" Content="Save">
    <Button.Resources>
        <TeachingTip x:Name="AutoSaveTip"
            Target="{x:Bind SaveButton}"
            Title="Saving automatically"
            Subtitle="We save your changes as you go - so you never have
to."
            ActionButtonContent="Disable"
            ActionButtonCommand="{x:Bind DisableAutoSaveCommand}"
            CloseButtonContent="Got it!">
                <StackPanel>
                    <CheckBox x:Name="HideTipsCheckBox" Content="Don't show
```

```
tips at start up" IsChecked="{x:Bind HidingTips, Mode=TwoWay}" />
                    <TextBlock>You can change your tip preferences in
<Hyperlink NavigateUri="app:/item/SettingsPage">Settings</Hyperlink> if you
change your mind.</TextBlock>
                </StackPanel>
        </TeachingTip>
    </Button.Resources>
</Button>
```



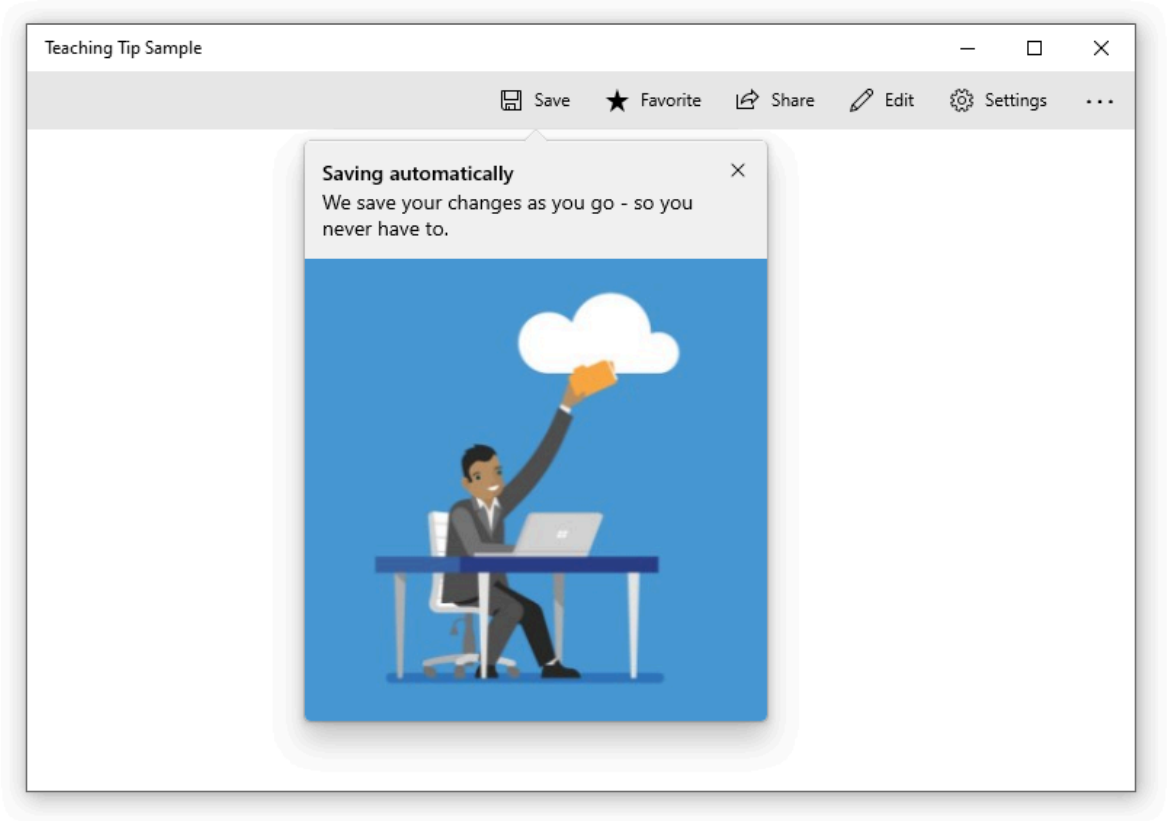
Hero content

Edge to edge content can be added to a teaching tip by setting the HeroContent property. The location of hero content can be set to the top or bottom of a teaching tip by setting the HeroContentPlacement property.

XAML

```
<Button x:Name="SaveButton" Content="Save">
    <Button.Resources>
        <TeachingTip x:Name="AutoSaveTip"
            Target="{x:Bind SaveButton}"
            Title="Saving automatically"
            Subtitle="We save your changes as you go - so you never have
to.">
            <TeachingTip.HeroContent>
                <Image Source="Assets/cloud.png" />
```

```
            </TeachingTip.HeroContent>
        </TeachingTip>
    </Button.Resources>
</Button>
```

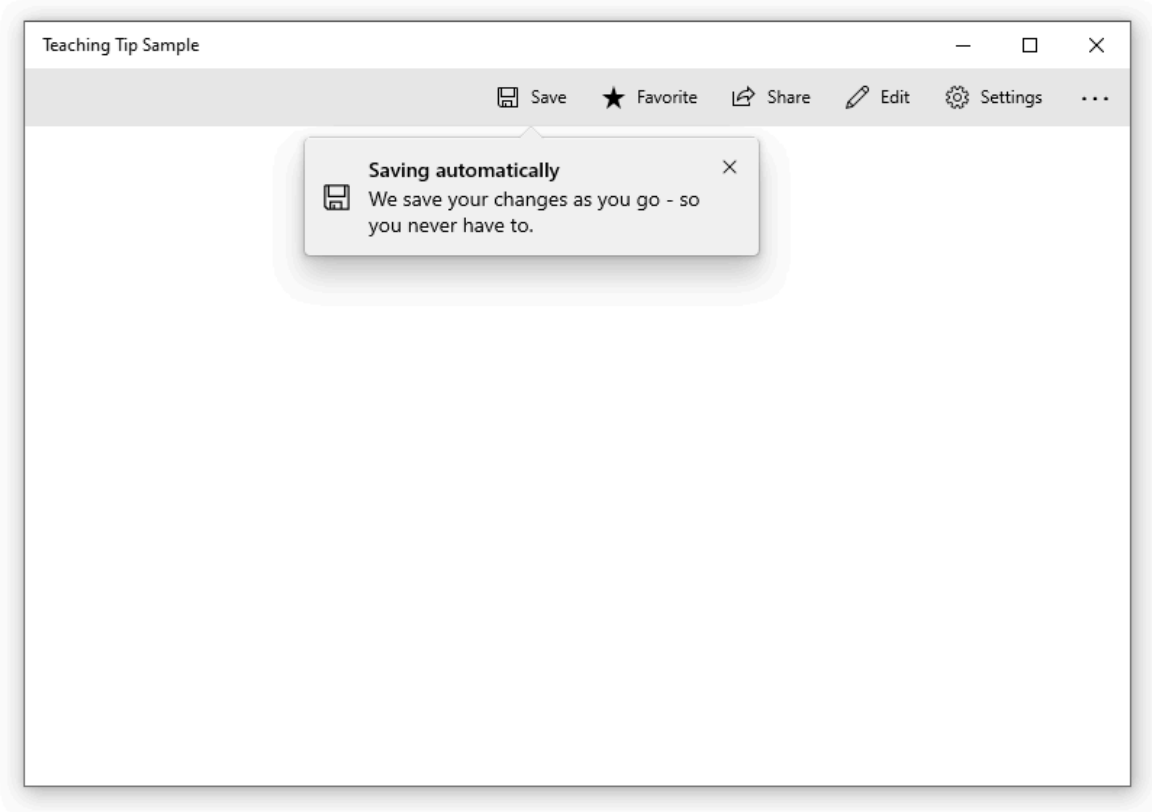


Add an icon

An icon can be added beside the title and subtitle using the IconSource property. Recommended icon sizes include 16px, 24px, and 32px.

XAML

```
<Button x:Name="SaveButton" Content="Save">
    <Button.Resources>
        <TeachingTip x:Name="AutoSaveTip"
            Target="{x:Bind SaveButton}"
            Title="Saving automatically"
            Subtitle="We save your changes as you go - so you never have
to."
            <TeachingTip.IconSource>
                <SymbolIconSource Symbol="Save" />
            </TeachingTip.IconSource>
        </TeachingTip>
    </Button.Resources>
</Button>
```



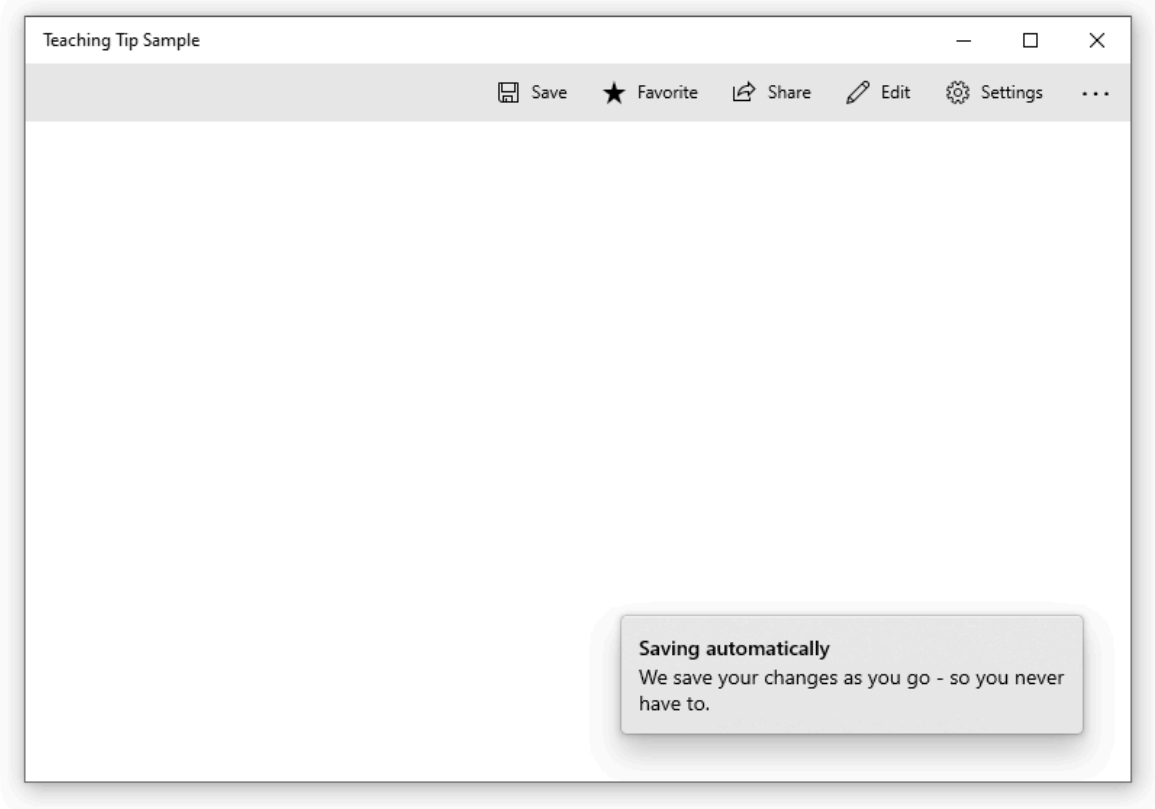
Enable light-dismiss

Light-dismiss functionality is disabled by default but it can enabled by setting the IsLightDismissEnabled property so that a teaching tip will dismiss, for example, when a user scrolls or interacts with other elements of the application. Because of this behavior, light-dismiss tips are the best solution when a tip needs to be placed in a scrollable area.

The close button will be automatically removed from a light-dismiss enabled teaching tip to identify its light-dismiss behavior to users.

XAML

```
<Button x:Name="SaveButton" Content="Save" />

<TeachingTip x:Name="AutoSaveTip"
    Title="Saving automatically"
    Subtitle="We save your changes as you go - so you never have to."
    IsLightDismissEnabled="True">
</TeachingTip>
```



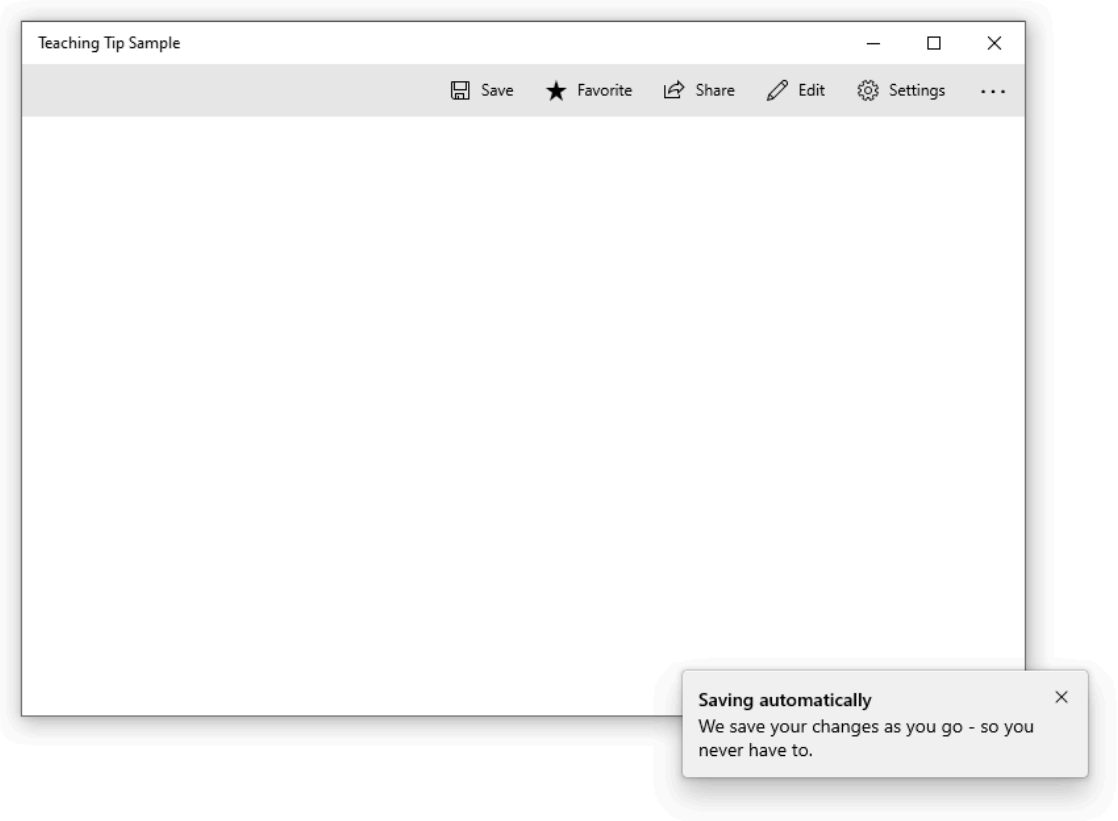
Escaping the XAML root bounds

Starting with Windows 10, version 1903 (Build 18362), a teaching tip can escape the bounds of the XAML root and the screen by setting the ShouldConstrainToRootBounds property. When this property is enabled, a teaching tip will not attempt to stay in the bounds of the XAML root nor the screen and will always position at the set `PreferredPlacement` mode. It is encouraged to enable the `IsLightDismissEnabled` property and set the `PreferredPlacement` mode nearest to the center of the XAML root to ensure the best experience for users.

On earlier versions of Windows, this property is ignored and the teaching tip always stays within the bounds of the XAML root.

XAML

```
<Button x:Name="SaveButton" Content="Save" />

<TeachingTip x:Name="AutoSaveTip"
    Title="Saving automatically"
    Subtitle="We save your changes as you go - so you never have to."
    PreferredPlacement="BottomRight"
    PlacementMargin="-80,-50,0,0"
    ShouldConstrainToRootBounds="False">
</TeachingTip>
```



Canceling and deferring close

The Closing event can be used to cancel and/or defer the close of a teaching tip. This can be used to keep the teaching tip open or allow time for an action or custom animation to occur. When the closing of a teaching tip is canceled, IsOpen will go back to true, however, it will stay false during the deferral. A programmatic close can also be canceled.

> **Note**
>
> If no placement option would allow a teaching tip to fully show, teaching tip will iterate through its event lifecycle to force a close rather than display without an accessible close button. If the app cancels the Closing event, the teaching tip may remain open without an accessible Close button.

XAML

```
<TeachingTip x:Name="EnableNewSettingsTip"
    Title="New ways to protect your privacy!"
    Subtitle="Please close this tip and review our updated privacy policy
and privacy settings."
```

```
    Closing="OnTipClosing">
</TeachingTip>
```

C#

```
private void OnTipClosing(muxc.TeachingTip sender,
muxc.TeachingTipClosingEventArgs args)
{
    if (args.Reason == muxc.TeachingTipCloseReason.CloseButton)
    {
        using(args.GetDeferral())
        {
            bool success = UpdateUserSettings(User thisUsersID);
            if(!success)
            {
                // We were not able to update the settings!
                // Don't close the tip and display the reason why.
                args.Cancel = true;
                ShowLastErrorMessage();
            }
        }
    }
}
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The TeachingTip for UWP apps requires WinUI 2. For more info, including installation instructions, see WinUI. APIs for this control exist in the Microsoft.UI.Xaml.Controls namespace.

- ✓ **WinUI 2 Apis:** TeachingTip class
- ✓ Open the WinUI 2 Gallery app and see the TeachingTip in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:TeachingTip />
```

Related articles

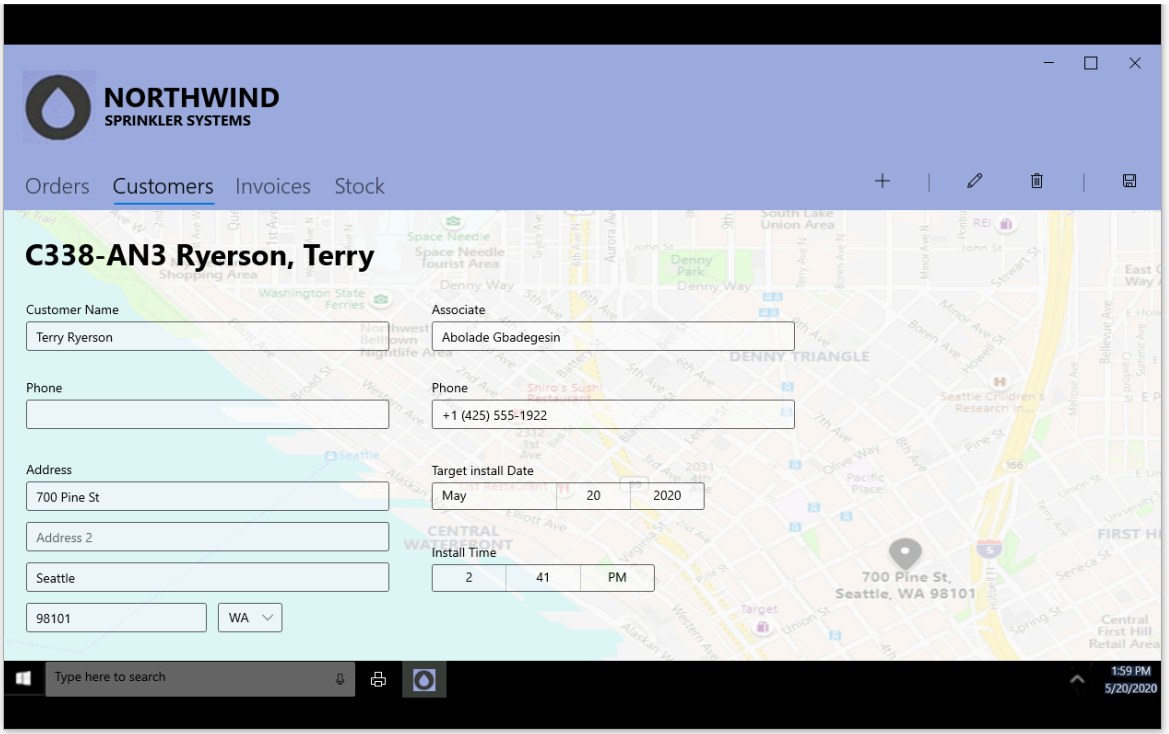
- Dialogs and flyouts

Forms

Article • 06/09/2022

A form is a group of controls that collect and submit data from users. Forms are typically used for settings pages, surveys, creating accounts, and much more.

This article discusses design guidelines for creating XAML layouts for forms.



When should you use a form?

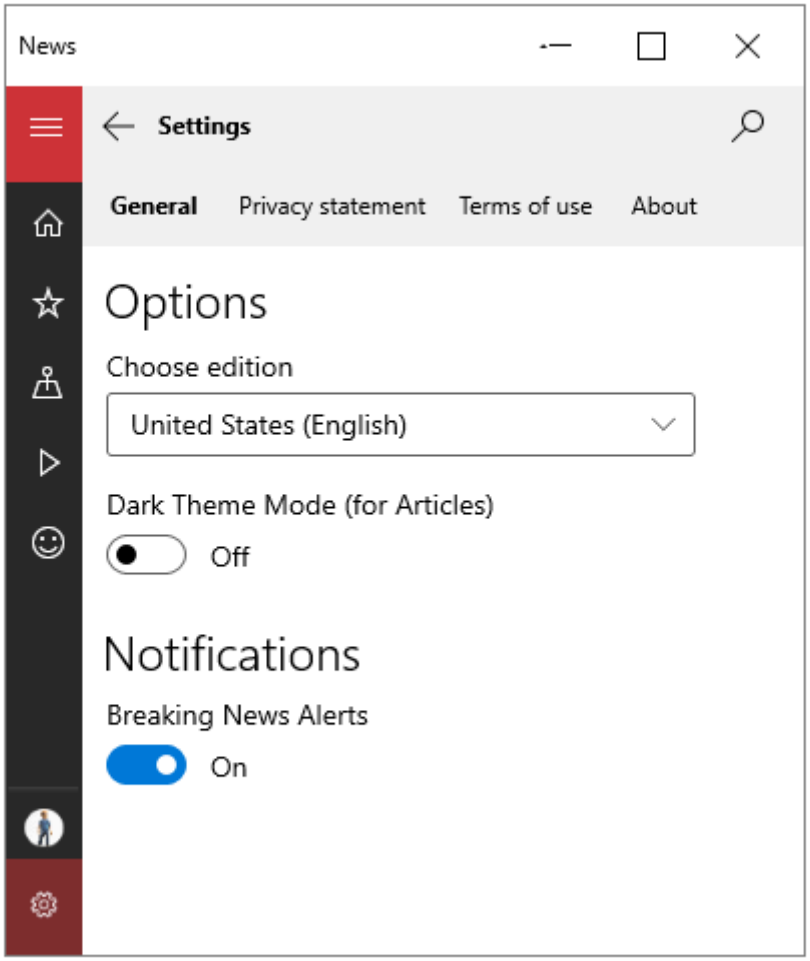
A form is a dedicated page for collecting data inputs that are clearly related to each other. You should use a form when you need to explicitly collect data from a user. You might create a form for a user to:

- Log into an account
- Sign up for an account
- Change app settings, such as privacy or display options
- Take a survey
- Purchase an item
- Give feedback

Types of forms

When thinking about how user input is submitted and displayed, there are two types of forms:

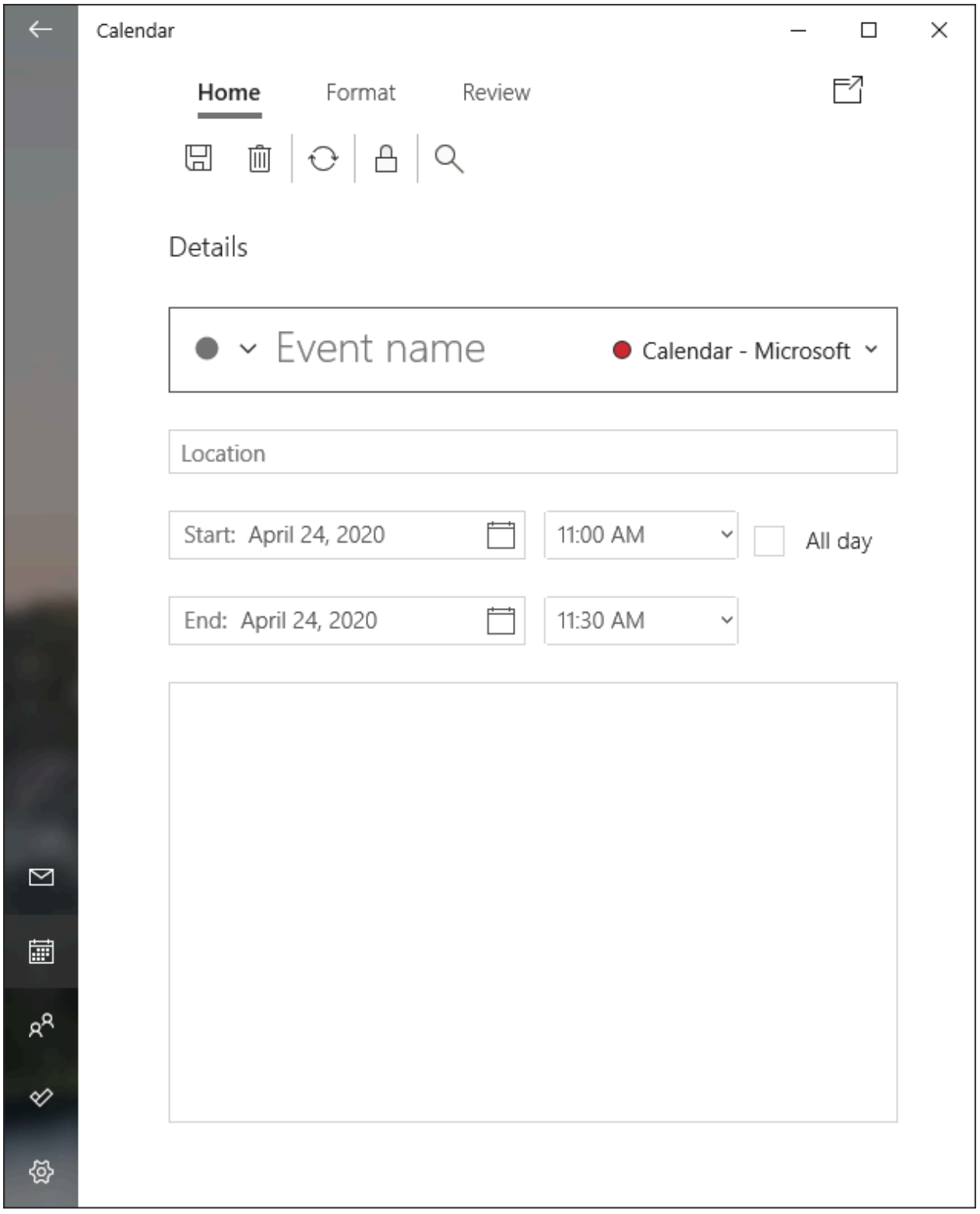
1. Instantly updating



Use an instantly updating form when you want users to immediately see the results of changing the values in the form. For example, in settings pages, the current selections are displayed, and any changes made to the selections are applied immediately. To acknowledge the changes in your app, you will need to add an event handler to each input control. If a user changes an input control, then your app can respond appropriately.

2. Submitting with button

The other type of form allows the user to choose when to submit data with a click of a button.



This type of form gives the user flexibility in responding. Typically, this type of form contains more free form input fields, and thus receives a greater variety of responses. To ensure valid user input and properly formatted data upon submission, consider the following recommendations:

- Make it impossible to submit invalid information by using the correct control (i.e., use a CalendarDatePicker rather than a TextBox for calendar dates). See more on selecting the appropriate input controls in your form in the Input Controls section later.

- When using TextBox controls, provide users a hint of the desired input format with the PlaceholderText property.
- Provide users with the appropriate on-screen keyboard by stating the expected input of a control with the InputScope property.
- Mark required input with an asterisk * on the label.
- Disable the submit button until all required information is filled in.
- If there is invalid data upon submission, mark the controls with invalid input with highlighted fields or borders, and require the user to resubmit the form.
- For other errors, such as failed network connection, make sure to display an appropriate error message to the user.

Layout

To facilitate the user experience and ensure that users are able to enter the correct input, consider the following recommendations for designing layouts for forms.

Labels

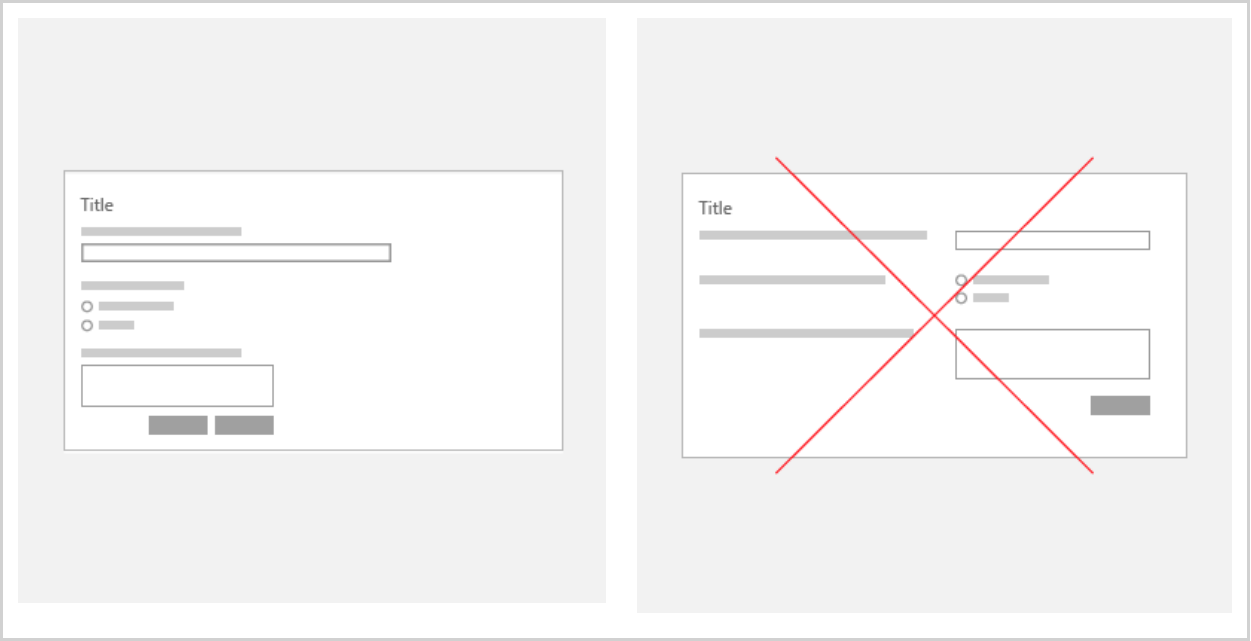
Labels should be left-aligned and placed above the input control. Many controls have a built-in Header property to display the label. For controls that don't have a Header property, or to label groups of controls, you can use a TextBlock instead.

To design for accessibility, label all individual and groups of controls for clarity for both human and screen readers.

For font styles, use the default Windows type ramp. Use `TitleTextBlockStyle` for page titles, `SubtitleTextBlockStyle` for group headings, and `BodyTextBlockStyle` for control labels.

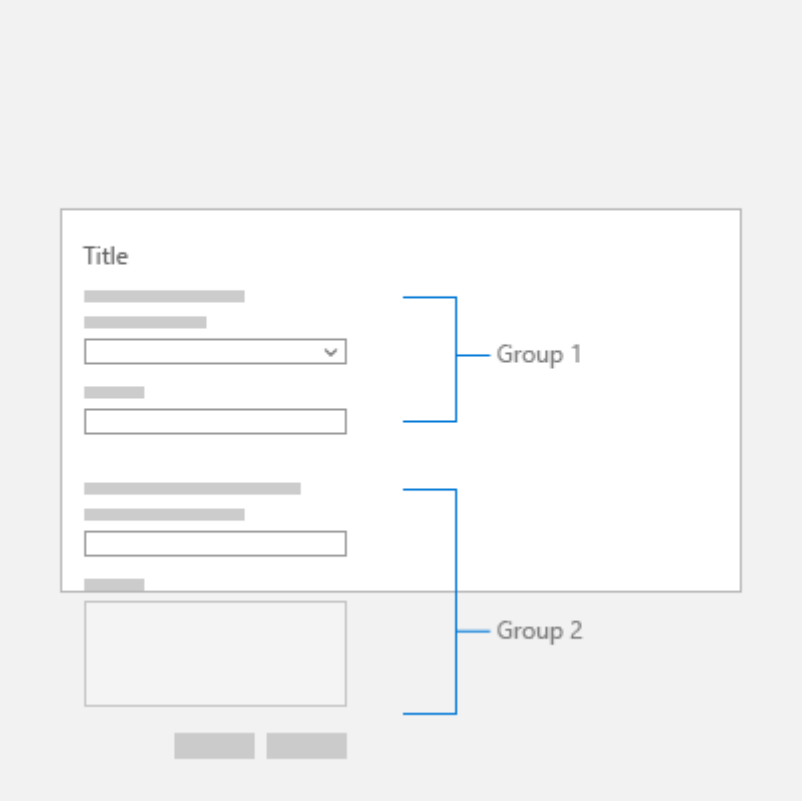
Expand table

Do	Don't



Spacing

To visually separate groups of controls from each other, use alignment, margins, and padding. Individual input controls are 80px in height and should be spaced 24px apart. Groups of input controls should be spaced 48px apart.

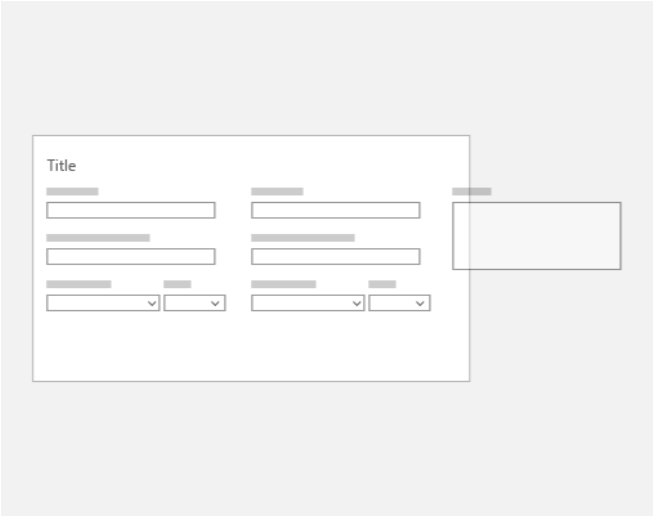


Columns

Creating columns can reduce unnecessary white space in forms, especially with larger screen sizes. However, if you would like to create a multi-column form, the number of

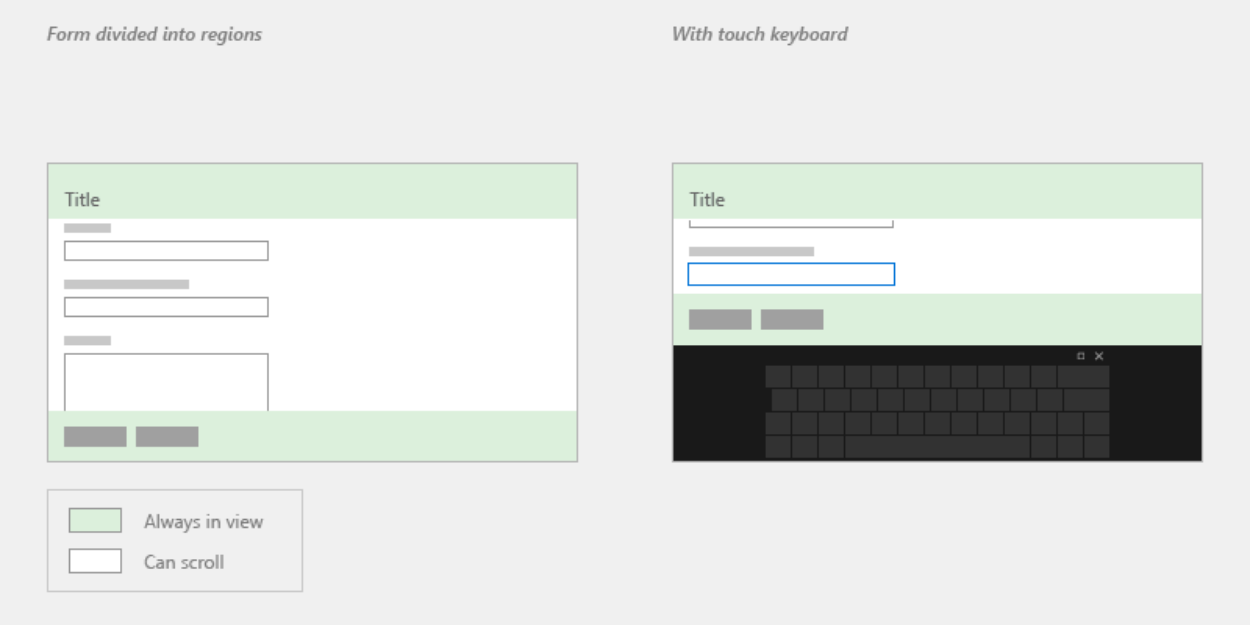
columns should depend on the number of input controls on the page and the screen size of the app window. Rather than overwhelm the screen with numerous input controls, consider creating multiple pages for your form.

Expand table

Do	Don't
	

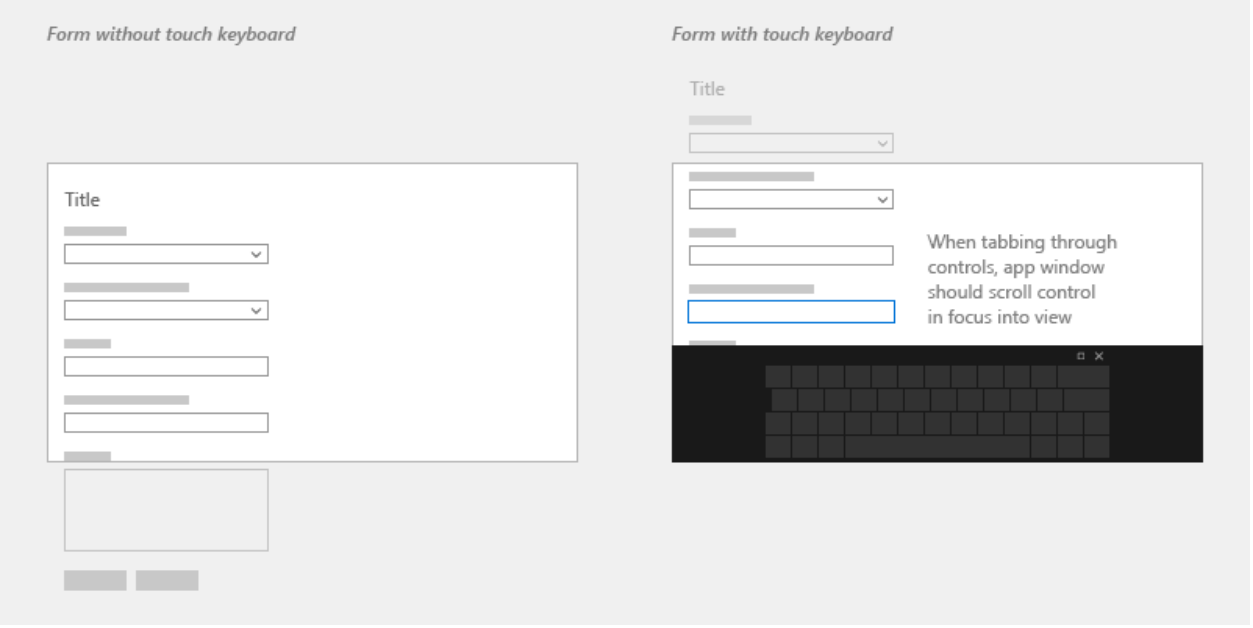
Responsive layout

Forms should resize as the screen or window size changes, so users don't overlook any input fields. For more information, see responsive design techniques. For example, you might want to keep specific regions of the form always in view, no matter the screen size.



Tab stops

Users can use the keyboard to navigate controls with tab stops. By default, the tab order of controls reflects the order in which they are created in XAML. To override the default behavior, change the **IsTabStop** or **TabIndex** properties of the control.



Input controls

Input controls are the UI elements that allow users to enter information into forms. Some common controls that can be added to forms are listed below, as well as information about when to use them.

Text input

Expand table

Control	Use	Example
TextBox	Capture one or multiple lines of text	Names, free form responses or feedback
PasswordBox	Collect private data by concealing the characters	Passwords, Social Security Numbers (SSN), PINs, credit card information
AutoSuggestBox	Show users a list of suggestions from a corresponding set of data as they type	Database search, mail to: line, previous queries
RichEditBox	Edit text files with formatted text, hyperlinks, and images	Upload file, preview, and edit in app

Selection

Expand table

Control	Use	Example
CheckBox	Select or deselect one or more action items	Agree to terms and conditions, add optional items, select all that apply
RadioButton	Select one option from two or more choices	Pick type, shipping method, etc.
ToggleSwitch	Choose one of two mutually exclusive options	On/off

> **Note**: If there are five or more selection items, use a list control.

Lists

Expand table

Control	Use	Example
ComboBox	Start in compact state and expand to show list of selectable items	Select from a long list of items, such as states or countries/regions
ListView	Categorize items and assign group headers, drag and drop items, curate content, and reorder items	Rank options
GridView	Arrange and browse image-based collections	Pick a photo, color, display theme

Numeric input

Expand table

Control	Use	Example
Slider	Select a number from a range of contiguous numerical values	Percentages, volume, playback speed
Rating	Rate with stars	Customer feedback

Date and Time

Expand table

Control	Use
CalendarView	Pick a single date or a range of dates from an always visible calendar
CalendarDatePicker	Pick a single date from a contextual calendar
DatePicker	Pick a single localized date when contextual info isn't important
TimePicker	Pick a single time value

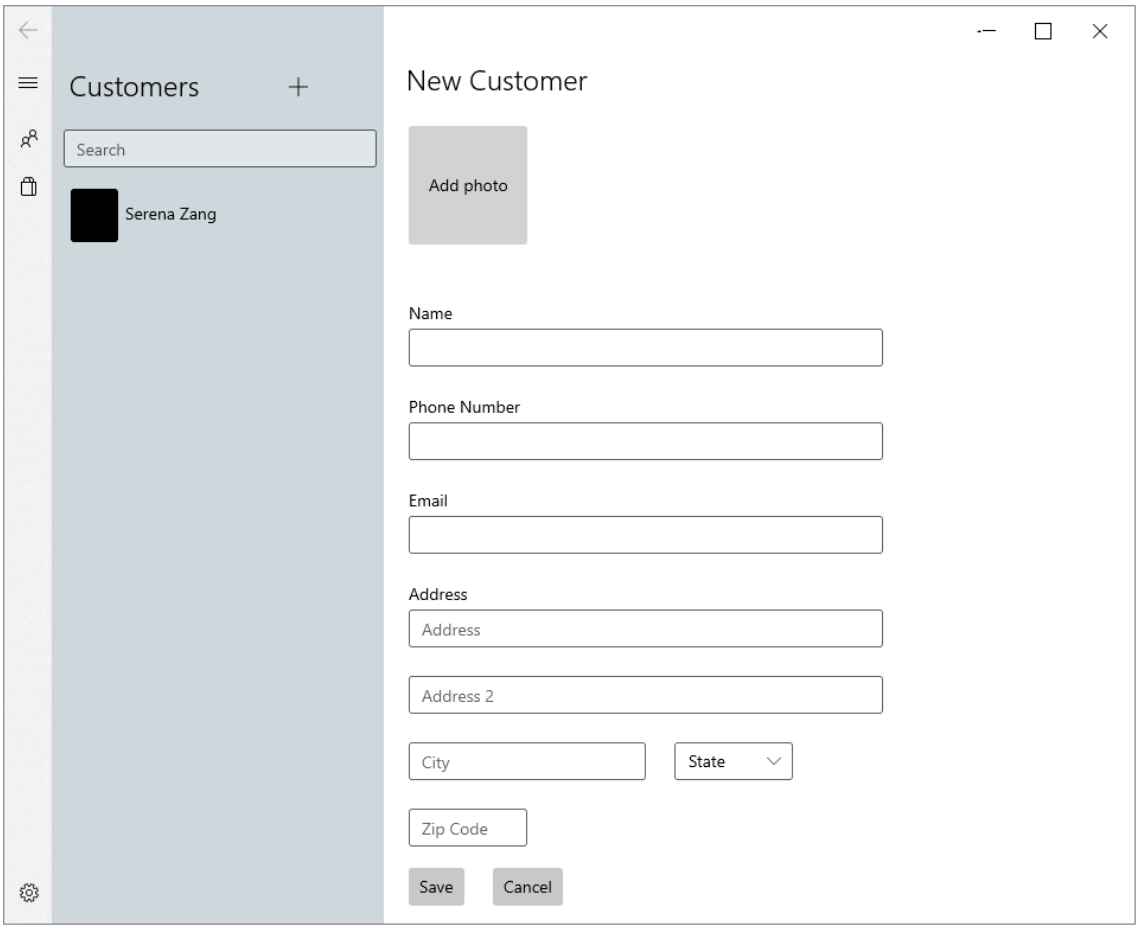
Additional Controls

For a complete list of UWP controls, see index of controls by function.

For more complex and custom UI controls, look at resources available from companies such as Telerik, SyncFusion, DevExpress, Infragistics, ComponentOne, and ActiPro.

One column form example

This example uses an Acrylic list/detail list view and NavigationView control.



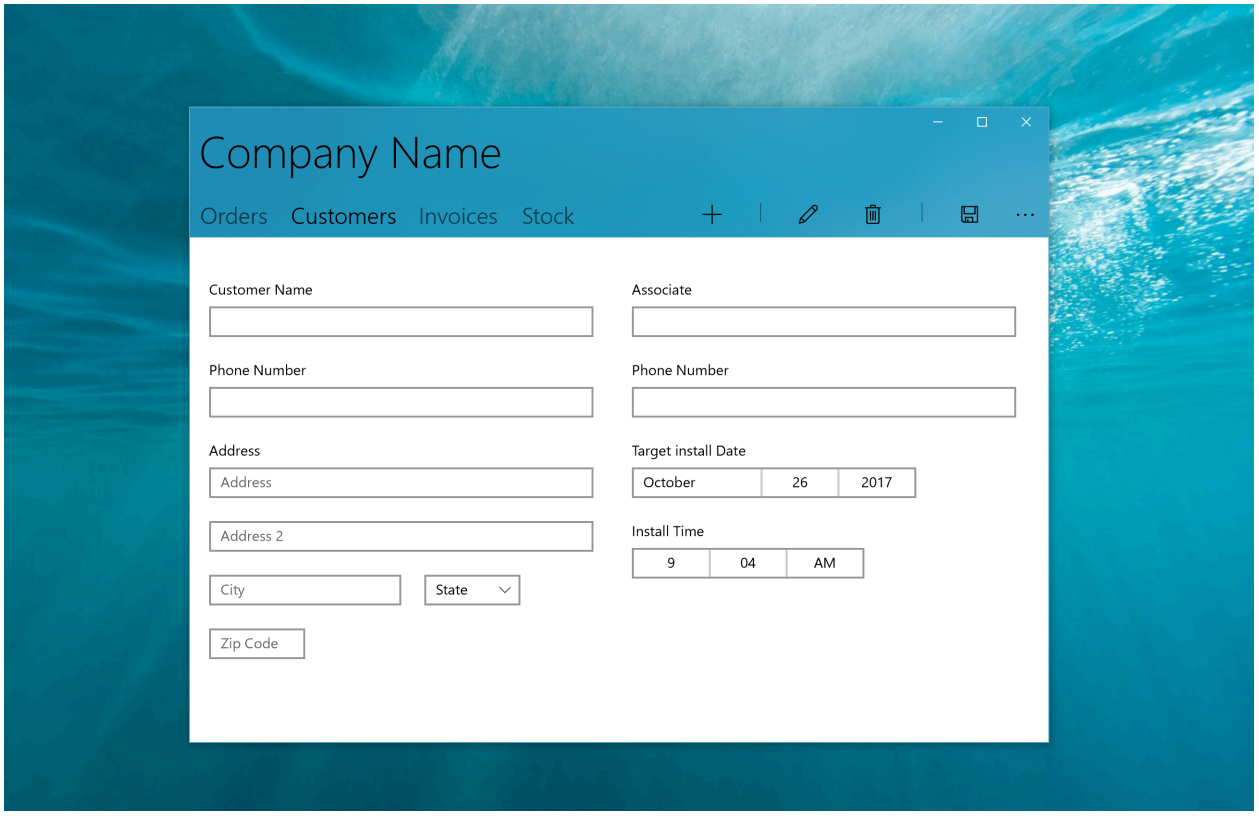
XAML

```
<StackPanel>
    <TextBlock Text="New Customer" Style="{StaticResource
TitleTextBlockStyle}"/>
    <Button Height="100" Width="100" Background="LightGray" Content="Add
photo" Margin="0,24" Click="AddPhotoButton_Click"/>
    <TextBox x:Name="Name" Header= "Name" Margin="0,24,0,0" MaxLength="32"
Width="400" HorizontalAlignment="Left" InputScope="PersonalFullName"/>
    <TextBox x:Name="PhoneNumber" Header="Phone Number" Margin="0,24,0,0"
MaxLength="15" Width="400" HorizontalAlignment="Left"
InputScope="TelephoneNumber" />
    <TextBox x:Name="Email" Header="Email" Margin="0,24,0,0" MaxLength="50"
Width="400" HorizontalAlignment="Left" InputScope="EmailNameOrAddress" />
    <TextBox x:Name="Address" Header="Address" PlaceholderText="Address"
Margin="0,24,0,0" MaxLength="50" Width="400" HorizontalAlignment="Left"
InputScope="EmailNameOrAddress" />
    <TextBox x:Name="Address2" Margin="0,24,0,0" PlaceholderText="Address 2"
MaxLength="50" Width="400" HorizontalAlignment="Left"
InputScope="EmailNameOrAddress" />
    <RelativePanel>
        <TextBox x:Name="City" PlaceholderText="City" Margin="0,24,0,0"
MaxLength="50" Width="200" HorizontalAlignment="Left"
InputScope="EmailNameOrAddress" />
```

```
        <ComboBox x:Name="State" PlaceholderText="State" Margin="24,24,0,0"
Width="100" RelativePanel.RightOf="City">
            <x:String>WA</x:String>
        </ComboBox>
    </RelativePanel>
    <TextBox x:Name="ZipCode" PlaceholderText="Zip Code" Margin="0,24,0,0"
MaxLength="6" Width="100" HorizontalAlignment="Left" InputScope="Number" />
    <StackPanel Orientation="Horizontal">
        <Button Content="Save" Margin="0,24" Click="SaveButton_Click"/>
        <Button Content="Cancel" Margin="24" Click="CancelButton_Click"/>
    </StackPanel>
</StackPanel>
```

Two column form example

This example uses the Pivot control, Acrylic background, and CommandBar in addition to input controls.



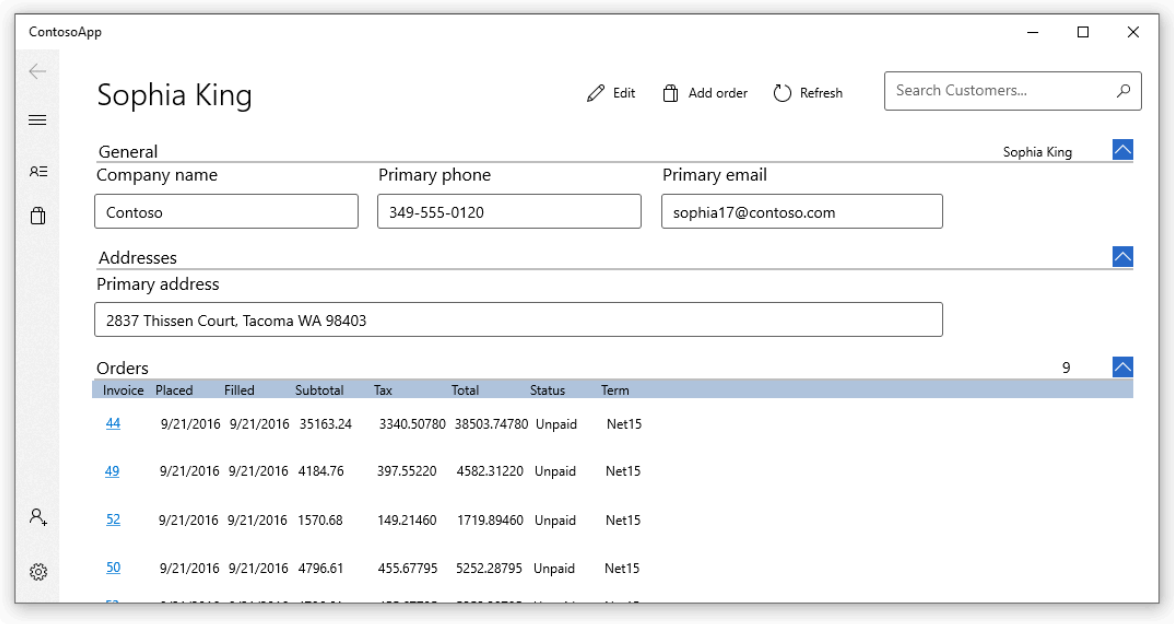
XAML

```
<Grid>
    <Pivot Background="{ThemeResource
SystemControlAccentAcrylicWindowAccentMediumHighBrush}" >
        <Pivot.TitleTemplate>
            <DataTemplate>
                <Grid>
                    <TextBlock Text="Company Name" Style="{ThemeResource
HeaderTextBlockStyle}"/>
                </Grid>
            </DataTemplate>
```

```
        </Pivot.TitleTemplate>
        <PivotItem Header="Orders" Margin="0"/>
        <PivotItem Header="Customers" Margin="0">
            <!--Form Example-->
            <Grid Background="White">
                <RelativePanel>
                    <StackPanel x:Name="Customer" Margin="20">
                        <TextBox x:Name="CustomerName" Header= "Customer
Name" Margin="0,24,0,0" MaxLength="320" Width="400"
HorizontalAlignment="Left" InputScope="PersonalFullName"/>
                        <TextBox x:Name="CustomerPhoneNumber" Header="Phone
Number" Margin="0,24,0,0" MaxLength="50" Width="400"
HorizontalAlignment="Left" InputScope="TelephoneNumber" />
                        <TextBox x:Name="Address" Header="Address"
PlaceholderText="Address" Margin="0,24,0,0" MaxLength="50" Width="400"
HorizontalAlignment="Left" InputScope="AlphanumericFullWidth" />
                        <TextBox x:Name="Address2" Margin="0,24,0,0"
PlaceholderText="Address 2" MaxLength="50" Width="400"
HorizontalAlignment="Left" InputScope="AlphanumericFullWidth" />
                        <RelativePanel>
                            <TextBox x:Name="City" PlaceholderText="City"
Margin="0,24,0,0" MaxLength="50" Width="200" HorizontalAlignment="Left"
InputScope="Text" />
                            <ComboBox x:Name="State" PlaceholderText="State"
Margin="24,24,0,0"  Width="100" RelativePanel.RightOf="City">
                                <x:String>WA</x:String>
                            </ComboBox>
                        </RelativePanel>
                        <TextBox x:Name="ZipCode" PlaceholderText="Zip Code"
Margin="0,24,0,0" MaxLength="6" Width="100" HorizontalAlignment="Left"
InputScope="Number" />
                    </StackPanel>
                    <StackPanel x:Name="Associate" Margin="20"
RelativePanel.RightOf="Customer">
                        <TextBox x:Name="AssociateName" Header= "Associate"
Margin="0,24,0,0" MaxLength="320" Width="400" HorizontalAlignment="Left"
InputScope="PersonalFullName"/>
                        <TextBox x:Name="AssociatePhoneNumber" Header="Phone
Number" Margin="0,24,0,0" MaxLength="50" Width="400"
HorizontalAlignment="Left" InputScope="TelephoneNumber" />
                        <DatePicker x:Name="TargetInstallDate"
Header="Target install Date" HorizontalAlignment="Left" Margin="0,24,0,0">
</DatePicker>
                        <TimePicker x:Name="InstallTime" Header="Install
Time" HorizontalAlignment="Left" Margin="0,24,0,0"></TimePicker>
                    </StackPanel>
                </RelativePanel>
            </Grid>
        </PivotItem>
        <PivotItem Header="Invoices"/>
        <PivotItem Header="Stock"/>
        <Pivot.RightHeader>
            <CommandBar OverflowButtonVisibility="Collapsed"
Background="Transparent">
                <AppBarButton Icon="Add"/>
```

```
                <AppBarSeparator/>
                <AppBarButton Icon="Edit" />
                <AppBarButton Icon="Delete"/>
                <AppBarSeparator/>
                <AppBarButton Icon="Save"/>
            </CommandBar>
        </Pivot.RightHeader>
    </Pivot>
</Grid>
```

Customer Orders Database Sample



To learn how to connect form input to an **Azure** database and see a fully implemented form, see the Customers Orders Database app sample.

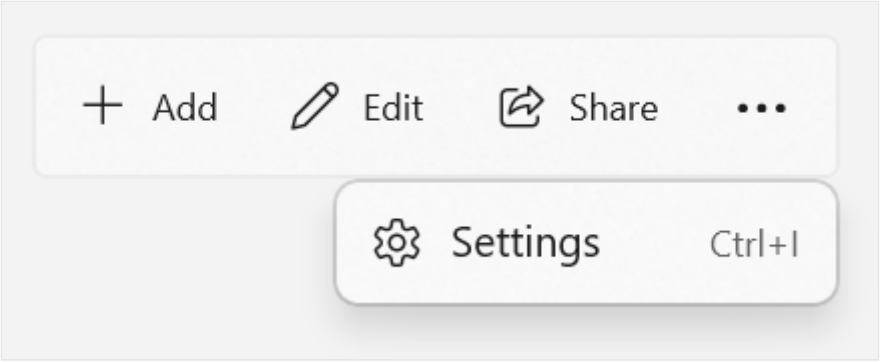
Related topics

- Input controls
- Typography

Icons in Windows apps

Article • 07/29/2024

Icons provide a visual shorthand for an action, concept, or product. By compressing meaning into a symbolic image, icons can cross language barriers and help conserve a valuable resource: screen space. Good icons harmonize with typography and with the rest of the design language. They don't mix metaphors, and they communicate only what's needed, as speedily and simply as possible.



Icons can appear within apps and outside them. Inside your app, you use icons to represent an action, such as copying text or going to the settings page.

This article describes icons within your app UI. To learn about icons that represent your app in Windows (app icons), see App icons.

Know when to use icons

Icons can save space, but when should you use them?

Use an icon for actions, like cut, copy, paste, and save, or for items on a navigation menu. Use an icon if it's easy for the user to understand what the icon means and it's simple enough to be clear at small sizes.

Don't use an icon if its meaning isn't clear, or if making it clear requires a complex shape.

Use the right type of icon

There are many ways to create an icon. You can use a symbol font like the Segoe Fluent Icons font. You can create your own vector-based image. You can even use a bitmap image, although we don't recommend it. Here's a summary of the ways that you can add an icon to your app.

To add an icon in your XAML app, you use either an IconElement or an IconSource.

This table shows the different kinds of icons that are derived from IconElement and IconSource.

Expand table

IconElement	IconSource	Description
AnimatedIcon	AnimatedIconSource	Represents an icon that displays and controls a visual that can animate in response to user interaction and visual state changes.
BitmapIcon	BitmapIconSource	Represents an icon that uses a bitmap as its content.
FontIcon	FontIconSource	Represents an icon that uses a glyph from the specified font.
IconSourceElement	N/A	Represents an icon that uses an IconSource as its content.
ImageIcon	ImageIconSource	Represents an icon that uses an image as its content.
PathIcon	PathIconSource	Represents an icon that uses a vector path as its content.
SymbolIcon	SymbolIconSource	Represents an icon that uses a glyph from the `SymbolThemeFontFamily` resource as its content.

IconElement vs. IconSource

IconElement is a FrameworkElement, so it can be added to the XAML object tree just like any other element of your app's UI. However, it can't be added to a ResourceDictionary and reused as a shared resource.

IconSource is similar to IconElement; however, because it is not a FrameworkElement, it can't be used as a standalone element in your UI, but it can be shared as a resource. IconSourceElement is a special icon element that wraps an IconSource so you can use it anywhere you need an IconElement. An example of these features is shown in the next section.

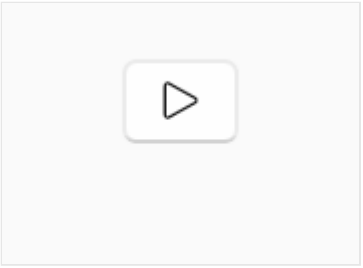
IconElement examples

You can use an IconElement-derived class as a standalone UI component.

This example shows how to set an icon glyph as the content of a Button. Set the button's FontFamily to `SymbolThemeFontFamily` and its content property to the Unicode value of the glyph that you want to use.

XAML

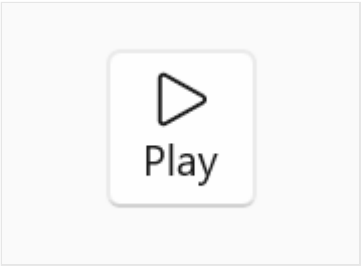
```
<Button FontFamily="{ThemeResource SymbolThemeFontFamily}"
        Content="&#xE768;"/>
```



You can also explicitly add one of the icon element objects listed previously, like SymbolIcon. This gives you more types of icons to choose from. It also lets you combine icons and other types of content, such as text, if you want.

XAML

```
<Button>
    <StackPanel>
        <SymbolIcon Symbol="Play"/>
        <TextBlock Text="Play" HorizontalAlignment="Center"/>
    </StackPanel>
</Button>
```



This example shows how you can define a FontIconSource in a ResourceDictionary, and then use an IconSourceElement to reuse the resource in different places of your app.

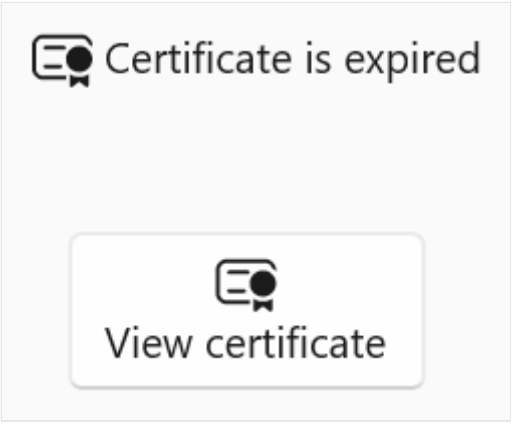
XAML

```
<!--App.xaml-->
<Application
   ...>
    <Application.Resources>
        <ResourceDictionary>
            ...
```

```
            <!-- Other app resources here -->

            <FontIconSource x:Key="CertIconSource" Glyph="&#xEB95;"/>

        </ResourceDictionary>
    </Application.Resources>
</Application>
```

XAML

```
<StackPanel Orientation="Horizontal">
    <IconSourceElement IconSource="{StaticResource CertIconSource}"/>
    <TextBlock Text="Certificate is expired" Margin="4,0,0,0"/>
</StackPanel>

<Button>
    <StackPanel>
        <IconSourceElement IconSource="{StaticResource CertIconSource}"/>
        <TextBlock Text="View certificate" HorizontalAlignment="Center"/>
    </StackPanel>
</Button>
```



Open the WinUI 3 Gallery app to see IconElement in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Icon properties

You often place icons in your UI by assigning one to an *icon* property of another XAML element. *Icon* properties that include *Source* in the name take an IconSource; otherwise, the property takes an IconElement.

This list shows some common elements that have an *icon* property.

Expand table

Commands/Actions	Navigation	Status/Other
AppBarButton.Icon	NavigationViewItem.Icon	AnimatedIcon.FallbackIconSource
AppBarToggleButton.Icon	SelectorBarItem.Icon	AnimatedIconSource.FallbackIconSource
AutoSuggestBox.QueryIcon	TabViewItem.IconSource	IconSourceElement.IconSource
MenuFlyoutItem.Icon		InfoBadge.IconSource
MenuFlyoutSubItem.Icon		InfoBar.IconSource
SwipeItem.IconSource		TeachingTip.IconSource
XamlUICommand.IconSource		

> **Tip**
>
> You can view these controls in the WinUI 3 Gallery app to see examples of how icons are used with them.

The remaining examples on this page show how to assign an icon to the `icon` property of a control.

FontIcon and SymbolIcon

The most common way to add icons to your app is to use the system icons provided by the icon fonts in Windows. Windows 11 introduces a new system icon font, Segoe Fluent Icons, which provides more than 1,000 icons designed for the Fluent Design language. It might not be intuitive to get an icon from a font, but Windows font display technology means these icons will look crisp and sharp on any display, at any resolution, and at any size.

> **Important**
>
> **Default font family**
>
> Rather than specifying a **FontFamily** directly, FontIcon and SymbolIcon use the font family defined by the `SymbolThemeFontFamily` XAML theme resource. By default, this resource uses the Segoe Fluent Icon font family. If your app is run on Windows 10, version 20H2 or earlier, the Segoe Fluent Icon font family is not available and the `SymbolThemeFontFamily` resource falls back to the **Segoe MDL2 Assets** font family instead.

Symbol enumeration

Many common glyphs from the `SymbolThemeFontFamily` are included in the Symbol enumeration. If the glyph you need is available as a Symbol, you can use a SymbolIcon anywhere you would use a FontIcon with the default font family.

You also use Symbol names to set an *`icon`* property in XAML using attribute syntax, like this

XAML

```
<AppBarButton Icon="Send" Label="Send"/>
```

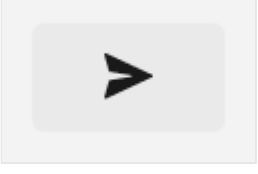
> **Tip**
>
> You can only use Symbol names to set an *`icon`* property using the shortened **attribute syntax**. All other icon types must be set using the longer **property element syntax**, as shown in other examples on this page.

Font icons

Only a small subset of Segoe Fluent Icon glyphs are available in the Symbol enumeration. To use any of the other available glyphs, use a FontIcon. This example shows how to create an AppBarButton with the `SendFill` icon.

XAML

```
<AppBarButton Label="Send">
    <AppBarButton.Icon>
        <FontIcon Glyph="&#xE725;"/>
    </AppBarButton.Icon>
</AppBarButton>
```

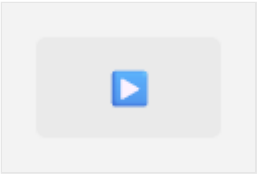


If you don't specify a FontFamily, or you specify a FontFamily that is not available on the system at runtime, the FontIcon falls back to the default font family defined by the `SymbolThemeFontFamily` theme resource.

You can also specify an icon using a Glyph value from any available font. This example shows how to use a Glyph from the Segoe UI Emoji font.

XAML

```
<AppBarToggleButton Label="FontIcon">
    <AppBarToggleButton.Icon>
        <FontIcon FontFamily="Segoe UI Emoji" Glyph="&#x25B6;"/>
    </AppBarToggleButton.Icon>
</AppBarToggleButton>
```



For more information and examples, see the FontIcon and SymbolIcon class documentation.

> **Tip**
>
> Use the Iconography page in the WinUI 3 Gallery app to view, search, and copy code for all the icons available in Segoe Fluent Icons.

Open the WinUI 3 Gallery app to the Iconography page

AnimatedIcon

Motion is an important part of the Fluent Design language. Animated icons draw attention to specific entry points, provide feedback from state to state, and add delight to an interaction.

You can use animated icons to implement lightweight, vector-based icons with motion using Lottie animations.

For more information and examples, see Animated icons and the AnimatedIcon class documentation.

Open the WinUI 3 Gallery app to see AnimatedIcon in action

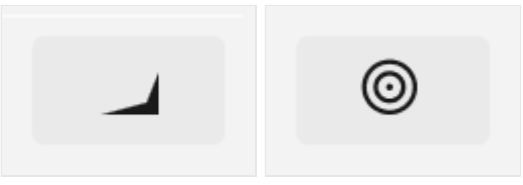
PathIcon

You can use a PathIcon to create custom icons that use vector-based shapes, so they always look sharp. However, creating a shape using a XAML Geometry is complicated because you have to individually specify each point and curve.

This example shows two different ways to define the Geometry used in a PathIcon.

XAML

```
<AppBarButton Label="PathIcon">
    <AppBarButton.Icon>
        <PathIcon Data="F1 M 16,12 20,2L 20,16 1,16"/>
    </AppBarButton.Icon>
</AppBarButton>

<AppBarButton Label="Circles">
    <AppBarButton.Icon>
        <PathIcon>
            <PathIcon.Data>
                <GeometryGroup>
                    <EllipseGeometry RadiusX="15" RadiusY="15"
Center="90,90" />
                    <EllipseGeometry RadiusX="50" RadiusY="50"
Center="90,90" />
                    <EllipseGeometry RadiusX="70" RadiusY="70"
Center="90,90" />
                    <EllipseGeometry RadiusX="100" RadiusY="100"
Center="90,90" />
                    <EllipseGeometry RadiusX="120" RadiusY="120"
Center="90,90" />
                </GeometryGroup>
            </PathIcon.Data>
        </PathIcon>
    </AppBarButton.Icon>
</AppBarButton>
```



To learn more about using Geometry classes to create custom shapes, see the class documentation and Move and draw commands for geometries. Also see the WPF Geometry documentation. The WinUI Geometry class doesn't have all the same features as the WPF class, but creating shapes is the same for both.

For more information and examples, see the PathIcon class documentation.

BitmapIcon and ImageIcon

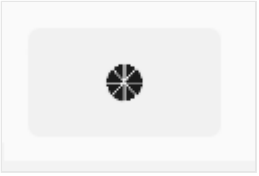
You can use a BitmapIcon or ImageIcon to create an icon from an image file (such as PNG, GIF, or JPEG), although we don't recommend it if another option is available. Bitmap images are created at a specific size, so they have to be scaled up or down depending on how large you want the icon to be and the resolution of the screen. When the image is scaled down (shrunk), it can appear blurry. When it's scaled up, it can appear blocky and pixelated.

BitmapIcon

By default, a BitmapIcon strips out all color information from the image and renders all non-transparent pixels in the Foreground color. To create a monochrome icon, use a solid image on a transparent background in PNG format. Other image formats will load apparently without error but result in a solid block of the Foreground color.

XAML

```
<AppBarButton Label="ImageIcon">
    <AppBarButton.Icon>
        <ImageIcon Source="ms-appx:///Assets/slices3.png"/>
    </AppBarButton.Icon>
</AppBarButton>
```



You can override the default behavior by setting the ShowAsMonochrome property to `false`. In this case, the BitmapIcon behaves the same as an ImageIcon for supported bitmap file types (SVG files are not supported).

XAML

```
<BitmapIcon UriSource="ms-appx:///Assets/slices3.jpg"
            ShowAsMonochrome="False"/>
```

For more information and examples, see the BitmapIcon class documentation.

Tip

Usage of BitmapIcon is similar to usage of BitmapImage; see the **BitmapImage** class for more information that's applicable to BitmapIcon, like setting the UriSource property in code.

ImageIcon

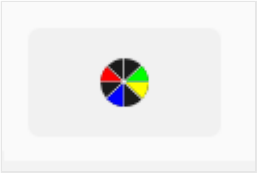
An ImageIcon shows the image provided by one of the ImageSource-derived classes. The most common is BitmapSource, but as mentioned previously, we don't recommend bitmap images for icons due to potential scaling issues.

Scalable Vector Graphics (SVG) resources are ideal for icons, because they always look sharp at any size or resolution. You can use an SVGImageSource with an ImageIcon, which supports secure static mode from the SVG specification but does not support animations or interactions. For more information, see SVGImageSource and SVG support.

An ImageIcon ignores the Foreground property, so it always shows the image in its original color. Because Foreground color is ignored, the icon doesn't respond to visual state changes when used in a button or other similar control.

XAML

```
<AppBarButton Label="ImageIcon">
    <AppBarButton.Icon>
        <ImageIcon Source="ms-appx:///Assets/slices3.svg"/>
    </AppBarButton.Icon>
</AppBarButton>
```



For more information and examples, see the ImageIcon class documentation.

Tip

Usage of ImageIcon is similar to the Image control; see the **Image** class for more information that's applicable to ImageIcon. One notable difference is that with ImageIcon, only the first frame of a multi-frame image (like an animated GIF) is used. To use animated icons, see **AnimatedIcon**.

Related articles

- Iconography in Windows
- App icons
- AnimatedIcon
- Command bar

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

AnimatedIcon

Article • 07/29/2024

An AnimatedIcon control plays animated images in response to user interaction and visual state changes.

Animated icons can draw attention to a UI component, such as the next button in a tutorial, or simply reflect the action associated with the icon in an entertaining and interesting way.

Custom animations can be created with Adobe AfterEffects and rendered with the Lottie-Windows library to use as an animated icon in your WinUI application. For more detail, see Use Lottie to create animated content for an AnimatedIcon later in this article.

The following example shows a basic animated search icon that was created in Adobe AfterEffects and rendered through Lottie.

Is this the right control?

Use the AnimatedIcon control when a control's icon needs to animate in response to user interaction with the control, such as when a user hovers over a button or clicks it.

Do not use an AnimatedIcon if the animation is not triggered by a visual state transition, and plays in a loop, plays one time only, or can be paused. Use AnimatedVisualPlayer instead.

Do not use AnimatedIcon for anything other than an icon, or where the control does not support an IconElement or IconElementSource property. Use AnimatedVisualPlayer instead.

When an animated icon is not required, use FontIcon, SymbolIcon, or BitmapIcon instead.

Differences between AnimatedIcon and AnimatedVisualPlayer

AnimatedIcon is an IconElement, which can be used anywhere an element or IconElement is required (such as NavigationViewItem.Icon), and is controlled through a

State property.

AnimatedVisualPlayer is a more general animation player, that is controlled through methods such as Play and Pause, and can be used anywhere in an application.

Use Lottie to create animated content for an AnimatedIcon

Defining an animation for an AnimatedIcon begins the same as the process to define an animation for an AnimatedVisualPlayer. You must create, or obtain, the Lottie file for the icon you want to add and run that file through LottieGen. LottieGen generates code for a C++/WinRT class that you can then instantiate and use with an AnimatedIcon.

> **Note**
>
> The AutoSuggestBox control uses the AnimatedVisuals.AnimatedFindVisualSource class, which was generated using the LottieGen tool.

You can also define markers in the animation definition to indicate playback time positions. You can then set the AnimatedIcon state to these markers. For example, if you have a playback position in the Lottie file marked "PointerOver", you can set the AnimatedIcon state to "PointerOver" and move the animation to that playback position.

Defining a color property in your Lottie animation named "Foreground" lets you to set the color using the AnimatedIcon.Foreground property.

Recommendations

- Please view the UX guidance for Icons for Windows Apps to ensure your icons match the design principles.
- Limit the number of animated icons on a single screen or view. Only animate icons to draw the user's attention to where they need to take action or when they are performing an action.

Create an animated icon

✓ **Important APIs:** AnimatedIcon class

Open the WinUI 3 Gallery app to see AnimatedIcon in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Add an AnimatedIcon to a Button

The following example demonstrates a back button that displays an animated back icon on a PointerEntered event.

- The `AnimatedBackVisualSource` is a class created with the LottieGen command line tool.
- The FallbackIconSource is used when animations can't be played, such as on older versions of Windows that don't support Lottie animations.
- If the user turns animations off in their system settings, AnimatedIcon displays the final frame of the state transition the controls was in.

XAML

```
<Button PointerEntered="Button_PointerEntered"
PointerExited="Button_PointerExited">
    <AnimatedIcon x:Name='BackAnimatedIcon'>
        <AnimatedIcon.Source>
            <animatedvisuals:AnimatedBackVisualSource/>
        </AnimatedIcon.Source>
        <AnimatedIcon.FallbackIconSource>
            <SymbolIconSource Symbol='Back'/>
        </AnimatedIcon.FallbackIconSource>
    </AnimatedIcon>
</Button>
```

cs

```
private void Button_PointerEntered(object sender, PointerRoutedEventArgs e)
{
    AnimatedIcon.SetState(this.BackAnimatedIcon, "PointerOver");
}

private void Button_PointerExited(object sender, PointerRoutedEventArgs e)
{
    AnimatedIcon.SetState(this.BackAnimatedIcon, "Normal");
}
```

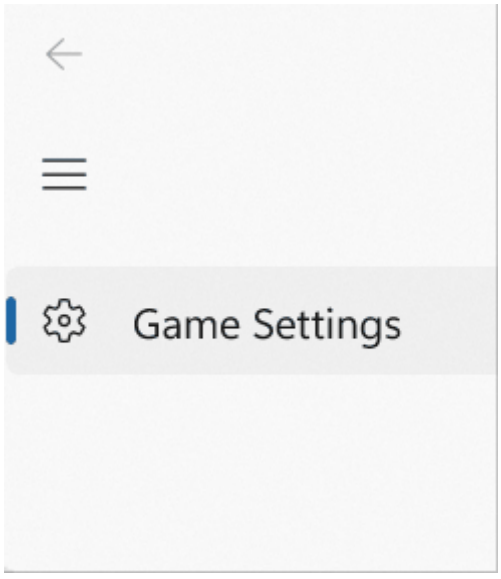
Add an AnimatedIcon to NavigationViewItem

The NavigationViewItem control automatically sets common states on an AnimatedIcon based on the state of the control, if those markers are defined in the Lottie animation.

For example, the following example shows how to set a custom animation (`GameSettingsIcon`) that was generated by the LottieGen tool:

XAML

```
<NavigationView.MenuItems>
    <NavigationViewItem Content = "Game Settings">
        <NavigationViewItem.Icon>
            <AnimatedIcon x:Name='GameSettingsIcon'>
                <AnimatedIcon.Source>
                    <animatedvisuals:AnimatedSettingsVisualSource/>
                </AnimatedIcon.Source>
                <AnimatedIcon.FallbackIconSource>
                    <FontIconSource FontFamily="Segoe MDL2 Assets"
Glyph="&#xE713;"/>
                </AnimatedIcon.FallbackIconSource>
            </AnimatedIcon>
        </NavigationViewItem.Icon>
    </NavigationViewItem>
</NavigationView.MenuItems>
```



The NavigationViewItem automatically sets the following states on the AnimatedIcon:

- Normal
- PointerOver
- Pressed
- Selected
- PressedSelected
- PointerOverSelected

If `GameSettingsIcon` has a marker defined for "NormalToPointerOver", the icon will be animated until the pointer moves over the NavigationViewItem. See the following section for more information on creating markers.

Define AnimatedIcon markers

To create the custom `GameSettingsIcon` in the previous example, run a Lottie JSON file (with markers) through the Windows LottieGen tool to generate the animation code as a C# class.

Once you run your Lottie file through LottieGen you can add the CodeGen output class to your project. See the LottieGen tutorial for more information.

Setting the AnimatedIcon state to a new value also sets a playback position in the Lottie animation for the transition from the old state to the new state. These playback positions are also identified with markers in the Lottie file. Specific markers for the start of the transition or the end of the transition can also be defined.

For example, the AutoSuggestBox control uses an AnimatedIcon that animates with the following states:

- Normal
- PointerOver
- Pressed

You can define markers in your Lottie file with those state names. You can also define markers as follows:

- Insert "To" between state names. For example, if you define a "NormalToPointerOver" marker, changing the AnimatedIcon state from "Normal" to "PointerOver" will cause it to move to this marker's playback position.
- Append "_Start" or "_End" to a marker name. For example defining markers "NormalToPointerOver_Start" and "NormalToPointerOver_End" and changing the AnimatedIcon state from "Normal" to "PointerOver" will cause it to jump to the _Start marker's playback position and then animate to the _End playback position.

The exact algorithm used to map AnimatedIcon State changes to marker playback positions:

- Check the provided file's markers for the markers "[PreviousState]To[NewState]_Start" and "[PreviousState]To[NewState]_End". If both are found play the animation from "[PreviousState]To[NewState]_Start" to "[PreviousState]To[NewState]_End".

- If "[PreviousState]To[NewState]_Start" is not found but "[PreviousState]To[NewState]_End" is found, then hard cut to the "[PreviousState]To[NewState]_End" playback position.
- If "[PreviousState]To[NewState]_End" is not found but "[PreviousState]To[NewState]_Start" is found, then hard cut to the "[PreviousState]To[NewState]_Start" playback position.
- Check if the provided IAnimatedVisualSource2's markers for the marker "[PreviousState]To[NewState]". If it is found then hard cut to the "[PreviousState]To[NewState]" playback position.
- Check if the provided IAnimatedVisualSource2's markers for the marker "[NewState]". If it is found, then hard cut to the "[NewState]" playback position.
- Check if the provided IAnimatedVisualSource2's markers has any marker which ends with "To[NewState]_End". If any marker is found which has that ending, hard cut to the first marker found with the appropriate ending's playback position.
- Check if "[NewState]" parses to a float. If it does, animated from the current position to the parsed float.
- Hard cut to playback position 0.0.

The following example shows the marker format in a Lottie JSON file. See the AnimatedIcon guidance for more detail.

JSON


```
"markers":[{"tm":0,"cm":"NormalToPointerOver_Start","dr":0},
{"tm":9,"cm":"NormalToPointerOver_End","dr":0},

{"tm":10,"cm":"NormalToPressed_Start","dr":0},
{"tm":19,"cm":"NormalToPressed_End","dr":0},

{"tm":20,"cm":"PointerOverToNormal_Start","dr":0},
{"tm":29,"cm":"PointerOverToNormal_End","dr":0},

{"tm":30,"cm":"PointerOverToPressed_Start","dr":0},
{"tm":39,"cm":"PointerOverToPressed_End","dr":0},

{"tm":40,"cm":"PressedToNormal_Start","dr":0},
{"tm":49,"cm":"PressedToNormal_End","dr":0},

{"tm":50,"cm":"PressedToPointerOver_Start","dr":0},
{"tm":69,"cm":"PressedToPointerOver_End","dr":0},

{"tm":90,"cm":"PressedToNormal_Start","dr":0},
{"tm":99,"cm":"PressedToNormal_End","dr":0},

{"tm":100,"cm":"PressedToPointerOver_Start","dr":0},
{"tm":101,"cm":"PressedToPointerOver_End","dr":0}]
```

Adding a Standalone AnimatedIcon

The following example is a button that the user clicks to Accept a prompt.

The `MyAcceptAnimation` class is created with the LottieGen tool.

The FallbackIconSource will be used rather than the animation when animations can't be played, such as on older versions of Windows that don't support Lottie animations.

If the end user turns off animations in their system settings, AnimatedIcon will display the final frame of the state transition the controls was in.

XML

```
<Button PointerEntered="HandlePointerEntered"
PointerExited="HandlePointerExited">
    <AnimatedIcon x:Name='AnimatedIcon1'>
        <local:MyAcceptAnimation/>
        <AnimatedIcon.FallbackIconSource>
            <SymbolIconSource Symbol='Accept'/>
        </AnimatedIcon.FallbackIconSource>
    </AnimatedIcon>
</Button>
```

cs

```
private void Button_PointerEntered(object sender, PointerRoutedEventArgs e)
{
    AnimatedIcon.SetState(this.AnimatedIcon1, "PointerOver");
}

private void Button_PointerExited(object sender, PointerRoutedEventArgs e)
{
    AnimatedIcon.SetState(this.StackPaAnimatedIcon1nel1, "Normal");
}
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

The AnimatedIcon for UWP apps requires WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in the Microsoft.UI.Xaml.Controls namespace.

- ✓ **WinUI 2 Apis:** AnimatedIcon class
- ✓ Open the WinUI 2 Gallery app and see the AnimatedIcon in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:AnimatedIcon />
```

Related articles

- Icons in Windows apps
- Motion for Windows apps

Images and image brushes

Article • 02/26/2025

To display an image, you can use either the **Image** object or the **ImageBrush** object. An Image object renders an image, and an ImageBrush object paints another object with an image.

Are these the right elements?

Use an **Image** element to display a stand-alone image in your app.

Use an **ImageBrush** to apply an image to another object. Uses for an ImageBrush include decorative effects for text, or backgrounds for controls or layout containers.

Create an image

✓ **Important APIs:** Image class, Source property, ImageBrush class, ImageSource property

Open the WinUI 3 Gallery app and see ImageBrushes in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Image

This example shows how to create an image by using the Image object.

XAML

```
<Image Width="200" Source="sunset.jpg" />
```

Here's the rendered Image object.

In this example, the Source property specifies the location of the image that you want to display. You can set the Source by specifying an absolute URL (for example, `http://contoso.com/myPicture.jpg`) or by specifying a URL that is relative to your app packaging structure. For our example, we put the "sunset.jpg" image file in the root folder of our project and declare project settings that include the image file as content.

ImageBrush

With the ImageBrush object, you can use an image to paint an area that takes a Brush object. For example, you can use an ImageBrush for the value of the Fill property of an Ellipse or the Background property of a Canvas.

The next example shows how to use an ImageBrush to paint an Ellipse.

XAML

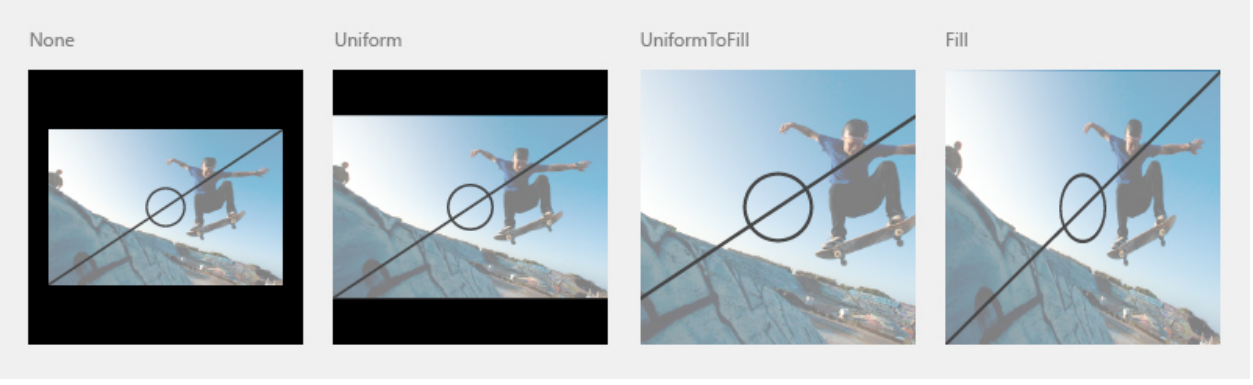
```
<Ellipse Height="200" Width="300">
   <Ellipse.Fill>
     <ImageBrush ImageSource="sunset.jpg" />
   </Ellipse.Fill>
</Ellipse>
```

Here's the Ellipse painted by the ImageBrush.

Stretch an image

If you don't set the Width or Height values of an **Image**, it is displayed with the dimensions of the image specified by the **Source**. Setting the **Width** and **Height** creates a containing rectangular area in which the image is displayed. You can specify how the image fills this containing area by using the Stretch property. The Stretch property accepts these values, which the Stretch enumeration defines:

- **None**: The image doesn't stretch to fill the output dimensions. Be careful with this Stretch setting: if the source image is larger than the containing area, your image will be clipped, and this usually isn't desirable because you don't have any control over the viewport like you do with a deliberate Clip.
- **Uniform**: The image is scaled to fit the output dimensions. But the aspect ratio of the content is preserved. This is the default value.
- **UniformToFill**: The image is scaled so that it completely fills the output area but preserves its original aspect ratio.
- **Fill**: The image is scaled to fit the output dimensions. Because the content's height and width are scaled independently, the original aspect ratio of the image might not be preserved. That is, the image might be distorted to completely fill the output area.



Crop an image

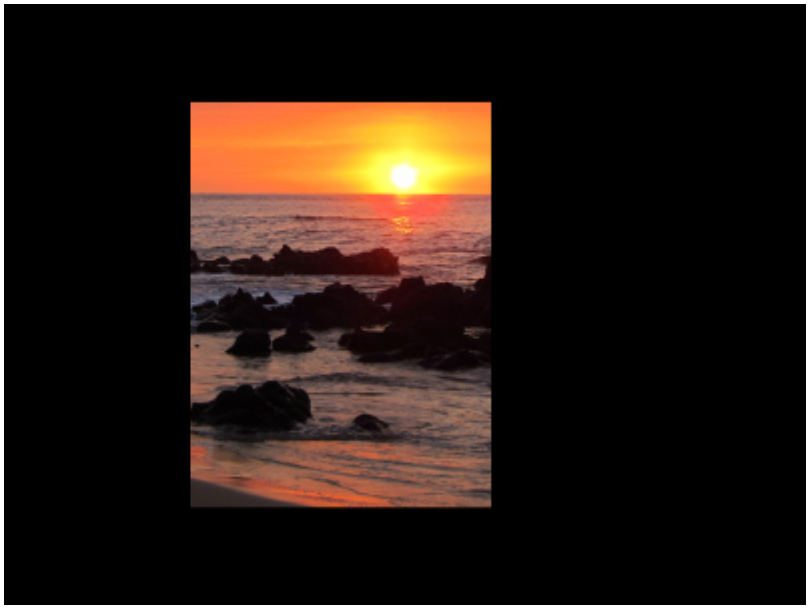
You can use the Clip property to clip an area from the image output. You set the Clip property to a Geometry. Currently, non-rectangular clipping is not supported.

The next example shows how to use a RectangleGeometry as the clip region for an image. In this example, we define an **Image** object with a height of 200. A **RectangleGeometry** defines a rectangle for the area of the image that will be displayed. The Rect property is set to "25,25,100,150", which defines a rectangle starting at position "25,25" with a width of 100 and a height of 150. Only the part of the image that is within the area of the rectangle is displayed.

XAML

```
<Image Source="sunset.jpg" Height="200">
    <Image.Clip>
        <RectangleGeometry Rect="25,25,100,150" />
    </Image.Clip>
</Image>
```

Here's the clipped image on a black background.



Apply an opacity

You can apply an Opacity to an image so that the image is rendered semi-translucent. The opacity values are from 0.0 to 1.0 where 1.0 is fully opaque and 0.0 is fully transparent. This example shows how to apply an opacity of 0.5 to an Image.

XAML

```
<Image Height="200" Source="sunset.jpg" Opacity="0.5" />
```

Here's the rendered image with an opacity of 0.5 and a black background showing through the partial opacity.

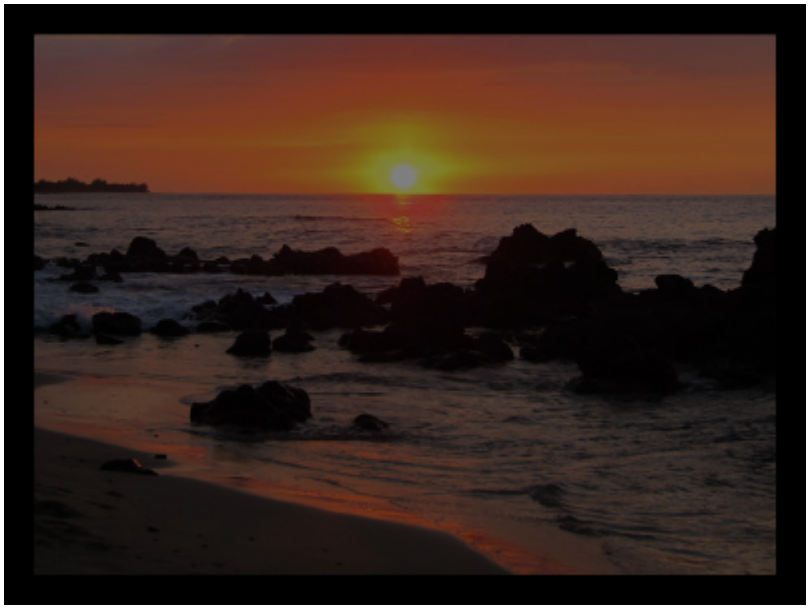


Image file formats

Image and **ImageBrush** can display these image file formats:

- Joint Photographic Experts Group (JPEG)
- Portable Network Graphics (PNG)
- bitmap (BMP)
- Graphics Interchange Format (GIF)
- Tagged Image File Format (TIFF)
- JPEG XR
- icons (ICO)

The APIs for Image, BitmapImage and BitmapSource don't include any dedicated methods for encoding and decoding of media formats. All of the encode and decode operations are built-in, and at most will surface aspects of encode or decode as part of event data for load events. If you want to do any special work with image encode or decode, which you might use if your app is doing image conversions or manipulation, you should use the APIs that are available in the Windows.Graphics.Imaging namespace. These APIs are also supported by the Windows Imaging Component (WIC) in Windows.

For more info about app resources and how to package image sources in an app, see Load images and assets tailored for scale, theme, high contrast, and others.

WriteableBitmap

A WriteableBitmap provides a BitmapSource that can be modified and that doesn't use the basic file-based decoding from the WIC. You can alter images dynamically and re-render the updated image. To define the buffer content of a **WriteableBitmap**, use the PixelBuffer property to access the buffer and use a stream or language-specific buffer type to fill it. For example code, see WriteableBitmap.

RenderTargetBitmap

The RenderTargetBitmap class can capture the XAML UI tree from a running app, and then represents a bitmap image source. After capture, that image source can be applied to other parts of the app, saved as a resource or app data by the user, or used for other scenarios. One particularly useful scenario is creating a runtime thumbnail of a XAML page for a navigation scheme. **RenderTargetBitmap** does have some limitations on the content that will appear in the captured image. For more info, see the API reference topic for RenderTargetBitmap.

Image sources and scaling

You should create your image sources at several recommended sizes, to ensure that your app looks great when Windows scales it. When specifying a **Source** for an **Image**, you can use a naming convention that will automatically reference the correct resource for the current scaling. For specifics of the naming convention and more info, see Quickstart: Using file or image resources.

For more info about how to design for scaling, see UX guidelines for layout and scaling.

Image and ImageBrush in code

It's typical to specify Image and ImageBrush elements using XAML rather than code. This is because these elements are often the output of design tools as part of a XAML UI definition.

If you define an Image or ImageBrush using code, use the default constructors, then set the relevant source property (Image.Source or ImageBrush.ImageSource). The source properties require a BitmapImage (not a URI) when you set them using code. If your source is a stream, use the SetSourceAsync method to initialize the value. If your source is a URI, which includes content in your app that uses the **ms-appx** or **ms-resource** schemes, use the BitmapImage constructor that takes a URI. You might also consider handling the ImageOpened event if there are any timing issues with retrieving or

decoding the image source, where you might need alternate content to display until the image source is available. For example code, see the WinUI Gallery sample.

> **Note**
>
> If you establish images using code, you can use automatic handling for accessing unqualified resources with current scale and culture qualifiers, or you can use ResourceManager and ResourceMap with qualifiers for culture and scale to obtain the resources directly. For more info see Resource management system.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls and Windows.UI.Xaml.Media namespaces.

- ✓ **UWP APIs:** Image class, Source property, ImageBrush class, ImageSource property
- ✓ Open the WinUI 2 Gallery app and see ImageBrushes in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls.

Starting in Windows 10, version 1607, the **Image** element supports animated GIF images. When you use a **BitmapImage** as the image **Source**, you can access BitmapImage APIs to control playback of the animated GIF image. For more info, see the Remarks on the BitmapImage class page.

> **Note**

Animated GIF support is available when your app is compiled for Windows 10, version 1607 and running on version 1607 (or later). When your app is compiled for or runs on previous versions, the first frame of the GIF is shown, but it is not animated.

Related articles

- Audio, video, and camera
- Image class
- ImageBrush class

Inking controls

Article • 06/03/2022

There are two different controls that facilitate inking in Windows apps: InkCanvas and InkToolbar.

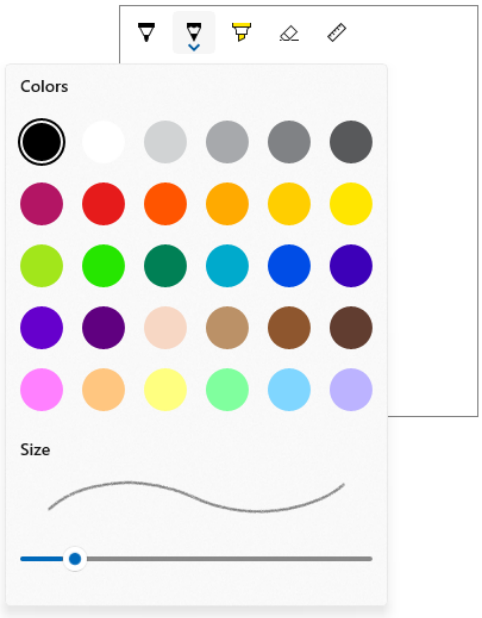
The InkCanvas control renders pen input as either an ink stroke (using default settings for color and thickness) or an erase stroke. This control is a transparent overlay that doesn't include any built-in UI for changing the default ink stroke properties.

> **Note**
>
> InkCanvas can be configured to support similar functionality for both mouse and touch input.

As the InkCanvas control does not include support for changing the default ink stroke settings, it can be paired with an InkToolbar control. The InkToolbar contains a customizable and extensible collection of buttons that activate ink-related features in an associated InkCanvas.

By default, the InkToolbar includes buttons for drawing, erasing, highlighting, and displaying a ruler. Depending on the feature, other settings and commands, such as ink color, stroke thickness, erase all ink, are provided in a flyout.

> **Note**
>
> InkToolbar supports pen and mouse input and can be configured to recognize touch input.



Is this the right control?

Use the InkCanvas when you need to enable basic inking features in your app without providing any ink settings to the user.

By default, strokes are rendered as ink when using the pen tip (a black ballpoint pen with a thickness of 2 pixels) and as an eraser when using the eraser tip. If an eraser tip is not present, the InkCanvas can be configured to process input from the pen tip as an erase stroke.

Pair the InkCanvas with an InkToolbar to provide a UI for activating ink features and setting basic ink properties such as stroke size, color, and shape of the pen tip.

> ⓘ **Note**
>
> For more extensive customization of ink stroke rendering on an InkCanvas, use the underlying **InkPresenter** object.

Ink toolbar overview

Built-in buttons

The InkToolbar includes the following built-in buttons:

Pens

- Ballpoint pen - draws a solid, opaque stroke with a circle pen tip. The stroke size is dependent on the pen pressure detected.
- Pencil - draws a soft-edged, textured, and semi-transparent stroke (useful for layered shading effects) with a circle pen tip. The stroke color (darkness) is dependent on the pen pressure detected.
- Highlighter – draws a semi-transparent stroke with a rectangle pen tip.

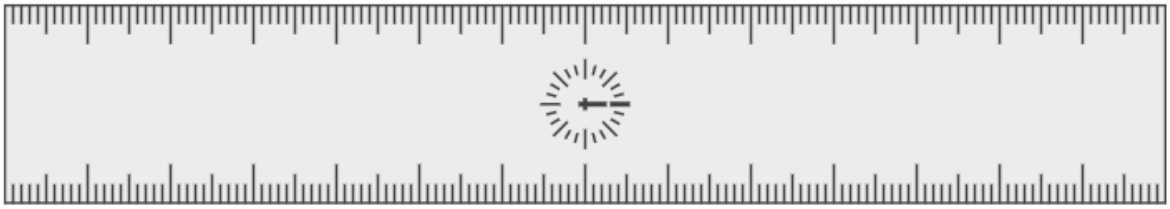
You can customize both the color palette and size attributes (min, max, default) in the flyout for each pen.

Tool

- Eraser – deletes any ink stroke touched. Note that the entire ink stroke is deleted, not just the portion under the eraser stroke.

Toggle

- Ruler – shows or hides the ruler. Drawing near the ruler edge causes the ink stroke to snap to the ruler.



Although this is the default configuration, you have complete control over which built-in buttons are included in the InkToolbar for your app.

Custom buttons

The InkToolbar consists of two distinct groups of button types:

1. A group of "tool" buttons containing the built-in drawing, erasing, and highlighting buttons. Custom pens and tools are added here.

> ⓘ **Note**
>
> Feature selection is mutually exclusive.

2. A group of "toggle" buttons containing the built-in ruler button. Custom toggles are added here.

> ⓘ **Note**

Features are not mutually exclusive and can be used concurrently with other active tools.

Depending on your application and the inking functionality required, you can add any of the following buttons (bound to your custom ink features) to the InkToolbar:

- Custom pen – a pen for which the ink color palette and pen tip properties, such as shape, rotation, and size, are defined by the host app.
- Custom tool – a non-pen tool, defined by the host app.
- Custom toggle – Sets the state of an app-defined feature to on or off. When turned on, the feature works in conjunction with the active tool.

> **Note**
>
> You cannot change the display order of the built-in buttons. The default display order is: Ballpoint pen, pencil, highlighter, eraser, and ruler. Custom pens are appended to the last default pen, custom tool buttons are added between the last pen button and the eraser button and custom toggle buttons are added after the ruler button. (Custom buttons are added in the order they are specified.)

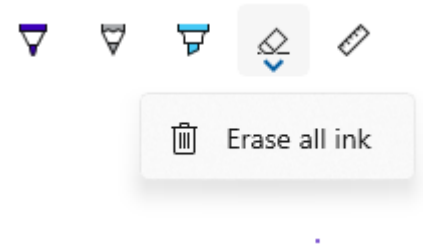
Although the InkToolbar can be a top level item, it is typically exposed through an "Inking" button or command. We recommend using EE56 glyph from the Segoe MLD2 Assets font as a top level icon.

InkToolbar Interaction

All built-in pen and tool buttons include a flyout menu where ink properties and pen tip shape and size can be set. An "extension glyph" is displayed on the button to indicate the existence of the flyout.

The flyout is shown when the button of an active tool is selected again. When the color or size is changed, the flyout is automatically dismissed and inking can be resumed. Custom pens and tools can use the default flyout or specify a custom flyout.

The eraser also has a flyout that provides the **Erase All Ink** command.



For information on customization and extensibility, check out SimpleInk sample.

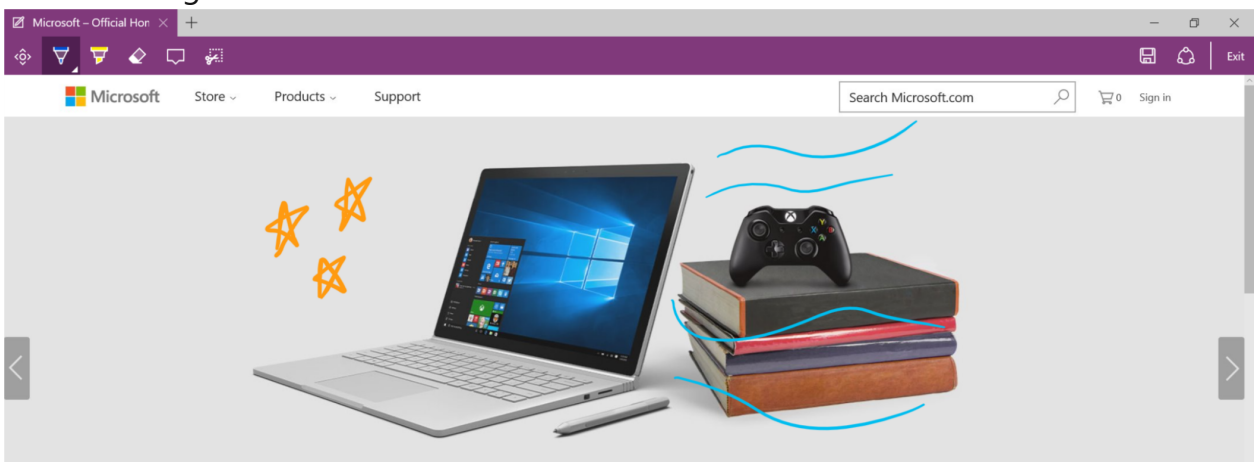
Recommendations

- The InkCanvas, and inking in general, is best experienced through an active pen. However, we recommend supporting inking with mouse and touch (including passive pen) input if required by your app.
- Use an InkToolbar control with the InkCanvas to provide basic inking features and settings. Both the InkCanvas and InkToolbar can be programmatically customized.
- The InkToolbar, and inking in general, is best experienced through an active pen. However, inking with mouse and touch can be supported if required by your app.
- If supporting inking with touch input, we recommend using the ED5F icon from the Segoe MLD2 Assets font for the toggle button, with a "Touch writing" tooltip.
- If providing stroke selection, we recommend using the EF20 icon from the Segoe MLD2 Assets font for the tool button, with a "Selection tool" tooltip.
- If using more than one InkCanvas, we recommend using a single InkToolbar to control inking across canvases.
- For best performance, we recommend altering the default flyout rather than creating a custom one for both default and custom tools.

Examples

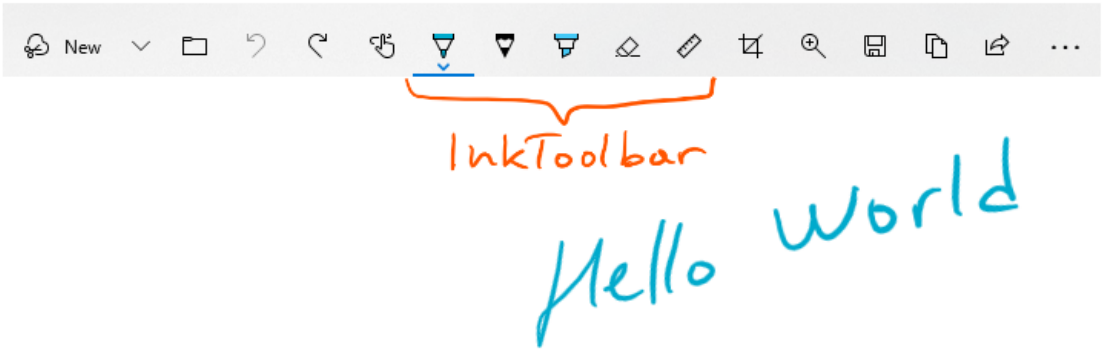
Microsoft Edge

Microsoft Edge uses the InkCanvas and InkToolbar for **Web Notes**.



Windows Ink Workspace

The InkCanvas and InkToolbar are also used for **Snip & Sketch** in the **Windows Ink Workspace**.



Create an InkCanvas and InkToolbar

- ✓ **UWP APIs:** InkCanvas class, InkToolbar class, InkPresenter class, Windows.UI.Input.Inking
- ✓ Open the WinUI 2 Gallery app and see the InkingControls in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

Adding an InkCanvas to your app requires just one line of markup:

XAML

```
<InkCanvas x:Name="myInkCanvas"/>
```

> ⓘ **Note**
>
> For detailed InkCanvas customization using InkPresenter, see the **"Pen interactions and Windows Ink in Windows apps"** article.

The InkToolbar control must be used in conjunction with an InkCanvas. Incorporating an InkToolbar (with all built-in tools) into your app requires one additional line of markup:

XAML

```
<InkToolbar TargetInkCanvas="{x:Bind myInkCanvas}"/>
```

This displays the following InkToolbar:

Get the sample code

- SimpleInk sample - Demonstrates 8 scenarios around the customization and extensibility capabilities of the InkCanvas and InkToolbar controls. Each scenario provides basic guidance on common inking situations and control implementations.
- ComplexInk sample - Demonstrates more advanced inking scenarios.
- WinUI 2 Gallery sample - See all the XAML controls in an interactive format.

Related articles

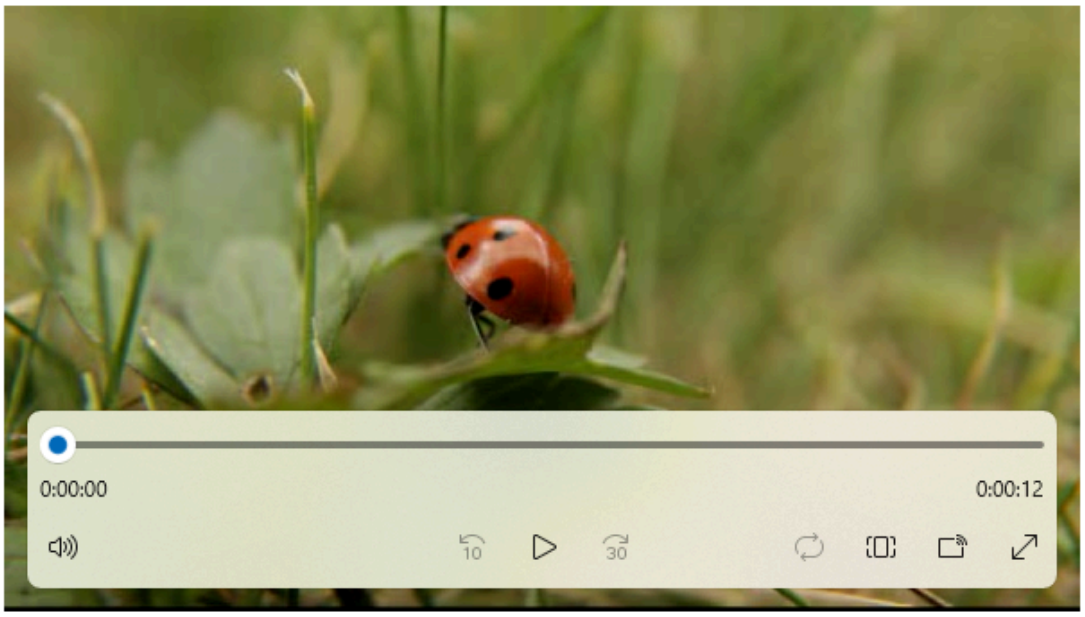
- Pen interactions and Windows Ink in Windows apps
- Recognize Windows Ink strokes as text and shapes
- Store and retrieve Windows Ink stroke data

Media players

Article • 02/26/2025

Media playback involves the viewing and listening of video and audio through inline (embedded in a page or with a group of other controls) or dedicated full-screen experiences.

Users expect a basic control set, such as play/pause, skip back, skip forward, which you can modify as required (including the media player's buttons, the background of the control bar, and control arrangement or layout).



Is this the right control?

Use a media player when you want to play audio or video in your app. To display a collection of images, use a flip view.

Recommendations

The media player supports both light and dark themes, but dark theme provides a better experience for most entertainment scenarios. The dark background provides better contrast, in particular for low-light conditions, and limits the control bar from interfering in the viewing experience.

When playing video content, encourage a dedicated viewing experience by promoting full-screen mode over inline mode. The full-screen viewing experience is optimal, and options are restricted in the inline mode.

If you have the screen real estate, go with the double-row layout. It provides more space for controls than the compact single-row layout and can be easier to navigate using a variety of inputs.

The default controls have been optimized for media playback, however you have the ability to add custom options you need to the media player in order to provide the best experience for your app. Visit Create custom transport controls to learn more about adding custom controls.

Create a media player

✓ **Important APIs**: MediaPlayerElement class, MediaTransportControls class

Open the WinUI 3 Gallery app and see the MediaPlayerElement in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Add media to your app by creating a MediaPlayerElement object in XAML and set the Source to a MediaSource that points to an audio or video file.

This XAML creates a MediaPlayerElement and sets its Source property to the URI of a video file that's local to the app. The `MediaPlayerElement` begins playing when the page loads. To suppress media from starting right away, you can set the AutoPlay property to `false`.

XAML

```
<MediaPlayerElement x:Name="mediaPlayerElement"
                    Source="ms-appx:///Videos/video1.mp4"
                    Width="400" AutoPlay="True"/>
```

This XAML creates a MediaPlayerElement with the built in transport controls enabled and the AutoPlay property set to `false.`

XAML

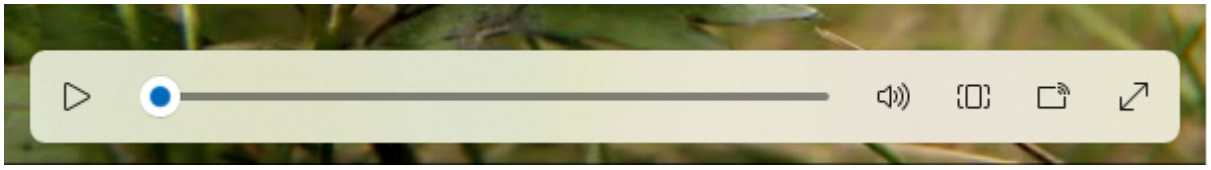
```
<MediaPlayerElement x:Name="mediaPlayerElement"
                    Source="ms-appx:///Videos/video1.mp4"
                    Width="400"
                    AutoPlay="False"
                    AreTransportControlsEnabled="True"/>
```

> **Important**
>
> Setting `MediaPlayerElement.Source` to a relative URI (ms-appx/ms-resource) only works in an app packaged with a Windows Application Packaging Project. If your app does not use a Windows Application Packaging Project, the recommended workaround is to convert the relative `ms-appx:///` URI to a fully resolved `file:///` URI. Also see the Set the media source and Open local media files sections later in this article.

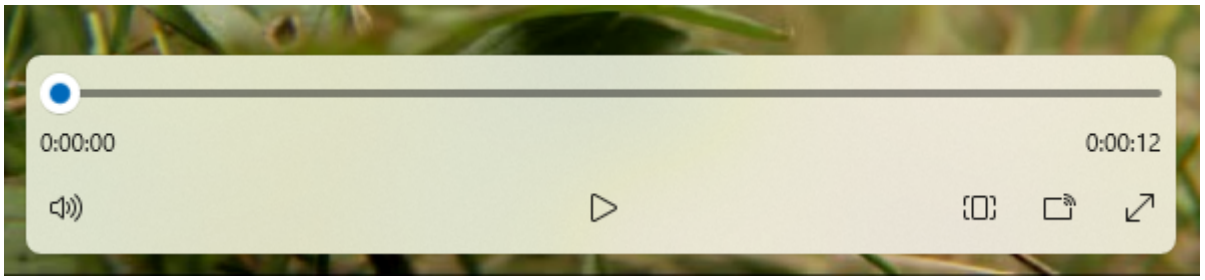
Media transport controls

MediaPlayerElement has built in transport controls that handle play, stop, pause, volume, mute, seeking/progress, closed captions, and audio track selection. To enable these controls, set AreTransportControlsEnabled to `true`. To disable them, set `AreTransportControlsEnabled` to `false`. The transport controls are represented by the MediaTransportControls class. You can use the transport controls as-is, or customize them in various ways. For more info, see the MediaTransportControls class reference and Create custom transport controls.

The transport controls support single- and double-row layouts. The first example here is a single-row layout, with the play/pause button located to the left of the media timeline. This layout is best reserved for inline media playback and compact screens.



The double-row controls layout (below) is recommended for most usage scenarios, especially on larger screens. This layout provides more space for controls and makes the timeline easier for the user to operate.



System media transport controls

MediaPlayerElement is automatically integrated with the system media transport controls. The system media transport controls are the controls that pop up when hardware media keys are pressed, such as the media buttons on keyboards. For more info, see SystemMediaTransportControls.

Set the media source

To play files on the network or files embedded with the app, set the Source property to a MediaSource with the path of the file.

> **Tip**
>
> To open files from the internet, you need to declare the **Internet (Client)** capability in your app's manifest (Package.appxmanifest). For more info about declaring capabilities, see App capability declarations.

This code attempts to set the Source property of the MediaPlayerElement defined in XAML to the path of a file entered into a TextBox.

XAML

```
<TextBox x:Name="txtFilePath" Width="400"
         FontSize="20"
         KeyUp="TxtFilePath_KeyUp"
         Header="File path"
         PlaceholderText="Enter file path"/>
```

C#

```
private void TxtFilePath_KeyUp(object sender, KeyRoutedEventArgs e)
{
    if (e.Key == Windows.System.VirtualKey.Enter)
    {
        TextBox tbPath = sender as TextBox;

        if (tbPath != null)
```

```
        {
            LoadMediaFromString(tbPath.Text);
        }
    }
}

private void LoadMediaFromString(string path)
{
    try
    {
        Uri pathUri = new Uri(path);
        mediaPlayerElement.Source = MediaSource.CreateFromUri(pathUri);
    }
    catch (Exception ex)
    {
        if (ex is FormatException)
        {
            // handle exception.
            // For example: Log error or notify user problem with file
        }
    }
}
```

To set the media source to a media file embedded in the app, initialize a Uri with the path prefixed with `ms-appx:///`, create a MediaSource with the Uri and then set the Source to the Uri. For example, for a file called **video1.mp4** that is in a **Videos** subfolder, the path would look like: `ms-appx:///Videos/video1.mp4`

> **Important**
>
> Setting `MediaPlayerElement.Source` to a relative URI (ms-appx/ms-resource) only works in an app packaged with a Windows Application Packaging Project.

This code sets the Source property of the MediaPlayerElement defined previously in XAML to `ms-appx:///Videos/video1.mp4`.

C#

```
private void LoadEmbeddedAppFile()
{
    try
    {
        Uri pathUri = new Uri("ms-appx:///Videos/video1.mp4");
        mediaPlayerElement.Source = MediaSource.CreateFromUri(pathUri);
    }
    catch (Exception ex)
    {
        if (ex is FormatException)
        {
```

```
            // handle exception.
            // For example: Log error or notify user problem with file
        }
    }
}
```

Open local media files

To open files on the local system or from OneDrive, you can use the FileOpenPicker to get the file and Source to set the media source, or you can programmatically access the user media folders.

If your app needs access without user interaction to the **Music** or **Video** folders, for example, if you are enumerating all the music or video files in the user's collection and displaying them in your app, then you need to declare the **Music Library** and **Video Library** capabilities. For more info, see Files and folders in the Music, Pictures, and Videos libraries.

The FileOpenPicker does not require special capabilities to access files on the local file system, such as the user's **Music** or **Video** folders, since the user has complete control over which file is being accessed. From a security and privacy standpoint, it is best to minimize the number of capabilities your app uses.

To open local media using FileOpenPicker

1. Call FileOpenPicker to let the user pick a media file.

   Use the FileOpenPicker class to select a media file. Set the FileTypeFilter to specify which file types the `FileOpenPicker` displays. Call PickSingleFileAsync to launch the file picker and get the file.

2. Use a MediaSource to set the chosen media file as the MediaPlayerElement.Source.

   To use the StorageFile returned from the FileOpenPicker, you need to call the CreateFromStorageFile method on MediaSource and set it as the Source of the MediaPlayerElement. Then call Play on the MediaPlayerElement.MediaPlayer to start the media.

This example shows how to use the FileOpenPicker to choose a file and set the file as the Source of a MediaPlayerElement.

XAML

```
<MediaPlayerElement x:Name="mediaPlayerElement"/>
...
```

```
<Button Content="Choose file" Click="Button_Click"/>
```

C#

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
    await SetLocalMedia();
}

async private System.Threading.Tasks.Task SetLocalMedia()
{
    var openPicker = new Windows.Storage.Pickers.FileOpenPicker();
    WinRT.Interop.InitializeWithWindow.Initialize(openPicker,
WinRT.Interop.WindowNative.GetWindowHandle(this));

    openPicker.FileTypeFilter.Add(".wmv");
    openPicker.FileTypeFilter.Add(".mp4");
    openPicker.FileTypeFilter.Add(".wma");
    openPicker.FileTypeFilter.Add(".mp3");

    var file = await openPicker.PickSingleFileAsync();

    // mediaPlayerElement is a MediaPlayerElement control defined in XAML
    if (file != null)
    {
        mediaPlayerElement.Source = MediaSource.CreateFromStorageFile(file);

        mediaPlayerElement.MediaPlayer.Play();
    }
}
```

Set the poster source

You can use the PosterSource property to provide your MediaPlayerElement with a visual representation before the media is loaded. A `PosterSource` is an image, such as a screen shot, movie poster, or album art, that is displayed in place of the media. The `PosterSource` is displayed in the following situations:

- When a valid source is not set. For example, Source is not set, `Source` was set to `null`, or the source is invalid (as is the case when a MediaFailed event occurs).
- While media is loading. For example, a valid source is set, but the MediaOpened event has not occurred yet.
- When media is streaming to another device.
- When the media is audio only.

Here's a MediaPlayerElement with its Source set to an album track, and it's PosterSource set to an image of the album cover.

XAML

```
<MediaPlayerElement Source="ms-appx:///Media/Track1.mp4" PosterSource="ms-appx:///Media/AlbumCover.png"/>
```

Keep the device's screen active

Typically, a device dims the display (and eventually turns it off) to save battery life when the user is away, but video apps need to keep the screen on so the user can see the video. To prevent the display from being deactivated when user action is no longer detected, such as when an app is playing video, you can call DisplayRequest.RequestActive. The DisplayRequest class lets you tell Windows to keep the display turned on so the user can see the video.

To conserve power and battery life, you should call DisplayRequest.RequestRelease to release the display request when it is no longer required. Windows automatically deactivates your app's active display requests when your app moves off screen, and re-activates them when your app comes back to the foreground.

Here are some situations when you should release the display request:

- Video playback is paused, for example, by user action, buffering, or adjustment due to limited bandwidth.
- Playback stops. For example, the video is done playing or the presentation is over.
- A playback error has occurred. For example, network connectivity issues or a corrupted file.

To keep the screen active

1. Create a global DisplayRequest variable. Initialize it to `null`.

   C#

   ```
   private DisplayRequest appDisplayRequest = null;
   ```

2. Call RequestActive to notify Windows that the app requires the display to remain on.

3. Call RequestRelease to release the display request whenever video playback is stopped, paused, or interrupted by a playback error. When your app no longer has any active display requests, Windows saves battery life by dimming the display (and eventually turning it off) when the device is not being used.

Each MediaPlayerElement.MediaPlayer has a PlaybackSession of type MediaPlaybackSession that controls various aspects of media playback such as PlaybackRate, PlaybackState and Position. Here, you use the PlaybackStateChanged event on MediaPlayer.PlaybackSession to detect situations when you should release the display request. Then, use the NaturalVideoHeight property to determine whether an audio or video file is playing, and keep the screen active only if video is playing.

XAML

```
<MediaPlayerElement x:Name="mediaPlayerElement" Source="ms-
appx:///Videos/video1.mp4"/>
```

C#

```
public sealed partial class MainWindow : Window
{
    public DisplayRequest appDisplayRequest = null;
    // using Microsoft.UI.Dispatching;
    private DispatcherQueue dispatcherQueue =
DispatcherQueue.GetForCurrentThread();

    public MainWindow()
    {
        this.InitializeComponent();
        mediaPlayerElement.MediaPlayer.PlaybackSession.PlaybackStateChanged
+=
            PlaybackSession_PlaybackStateChanged;
    }

    private void PlaybackSession_PlaybackStateChanged(MediaPlaybackSession
sender, object args)
    {
        MediaPlaybackSession playbackSession = sender as
MediaPlaybackSession;
        if (playbackSession != null && playbackSession.NaturalVideoHeight !=
0)
        {
            if (playbackSession.PlaybackState == MediaPlaybackState.Playing)
            {
                if (appDisplayRequest is null)
                {

dispatcherQueue.TryEnqueue(DispatcherQueuePriority.Normal, () =>
                    {
                        appDisplayRequest = new DisplayRequest();
                        appDisplayRequest.RequestActive();
                    });
                }
            }
            else // PlaybackState is Buffering, None, Opening, or Paused.
            {
```

```
                if (appDisplayRequest is not null)
                {
                    appDisplayRequest.RequestRelease();
                    appDisplayRequest = null;
                }
            }
        }
    }
}
```

Control the media player programmatically

MediaPlayerElement provides numerous properties, methods, and events for controlling audio and video playback through the MediaPlayerElement.MediaPlayer property. For a full listing of properties, methods, and events, see the MediaPlayer reference page.

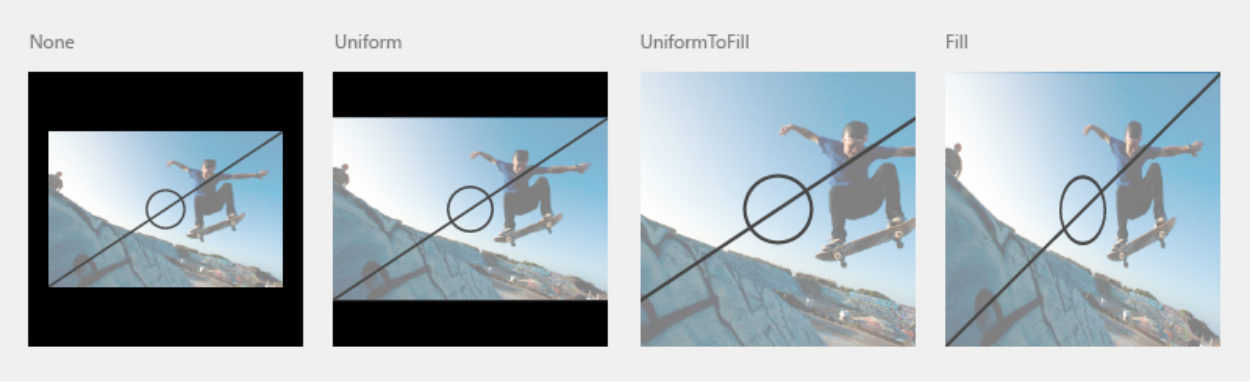
Advanced media playback scenarios

For more complex media playback scenarios like playing a playlist, switching between audio languages, or creating custom metadata tracks, set the MediaPlayerElement.Source to a MediaPlaybackItem or a MediaPlaybackList. See the Media playback page for more information on how to enable various advanced media functionality.

Resize and stretch video

Use the Stretch property to change how the video content and/or the PosterSource fills the container it's in. This resizes and stretches the video depending on the Stretch value. The `Stretch` states are similar to picture size settings on many TV sets. You can hook this up to a button and allow the user to choose which setting they prefer.

- None displays the native resolution of the content in its original size.This can result in some of the video being cropped or black bars at the edges of the video.
- Uniform fills up as much of the space as possible while preserving the aspect ratio and the video content. This can result in horizontal or vertical black bars at the edges of the video. This is similar to wide-screen modes.
- UniformToFill fills up the entire space while preserving the aspect ratio. This can result in some of the video being cropped. This is similar to full-screen modes.
- Fill fills up the entire space, but does not preserve the aspect ratio. None of video is cropped, but stretching may occur. This is similar to stretch modes.



Here, an AppBarButton is used to cycle through the Stretch options. A `switch` statement checks the current state of the Stretch property and sets it to the next value in the `Stretch` enumeration. This lets the user cycle through the different stretch states.

XAML

```
<AppBarButton Icon="Trim"
              Label="Resize Video"
              Click="PictureSize_Click" />
```

C#

```
private void PictureSize_Click(object sender, RoutedEventArgs e)
{
    switch (mediaPlayerElement.Stretch)
    {
        case Stretch.Fill:
            mediaPlayerElement.Stretch = Stretch.None;
            break;
        case Stretch.None:
            mediaPlayerElement.Stretch = Stretch.Uniform;
            break;
        case Stretch.Uniform:
            mediaPlayerElement.Stretch = Stretch.UniformToFill;
            break;
        case Stretch.UniformToFill:
            mediaPlayerElement.Stretch = Stretch.Fill;
            break;
        default:
            break;
    }
}
```

Enable low-latency playback

Set the RealTimePlayback property to `true` on a MediaPlayerElement.MediaPlayer to enable the media player element to reduce the initial latency for playback. This is critical

for two-way communications apps, and can be applicable to some gaming scenarios. Be aware that this mode is more resource intensive and less power-efficient.

This example creates a MediaPlayerElement and sets RealTimePlayback to `true`.

```
C#

MediaPlayerElement mediaPlayerElement = new MediaPlayerElement();
mediaPlayerElement.MediaPlayer.RealTimePlayback = true;
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** MediaPlayerElement class, MediaTransportControls class
- ✓ Open the WinUI 2 Gallery app and see the MediaPlayerElement in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

If you are designing for the 10-foot experience, go with the double-row layout. It provides more space for controls than the compact single-row layout and it is easier to navigate using a gamepad for 10-foot. See the Designing for Xbox and TV article for more information about optimizing your application for the 10-foot experience.

`MediaPlayerElement` is only available in Windows 10, version 1607 and later. If you are developing an app for an earlier version of Windows 10, you need to use the

MediaElement control instead. All recommendations made here apply to `MediaElement` as well.

Related articles

- Command design basics for Windows apps
- Content design basics for Windows apps

Create custom transport controls

Article • 05/13/2022

MediaPlayerElement has customizable XAML transport controls to manage control of audio and video content within a Windows app. Here, we demonstrate how to customize the MediaTransportControls template. We'll show you how to work with the overflow menu, add a custom button and modify the slider.

> **Important APIs**: MediaPlayerElement, MediaPlayerElement.AreTransportControlsEnabled, MediaTransportControls

Before starting, you should be familiar with the MediaPlayerElement and the MediaTransportControls classes. For more info, see the MediaPlayerElement control guide.

> **Tip**
>
> The examples in this topic are based on the **Media Transport Controls sample**. You can download the sample to view and run the completed code.

> **Note**
>
> **MediaPlayerElement** is only available in Windows 10, version 1607 and later. If you are developing an app for an earlier version of Windows 10 you will need to use MediaElement instead. All of the examples on this page work with **MediaElement** as well.

When should you customize the template?

MediaPlayerElement has built-in transport controls that are designed to work well without modification in most video and audio playback apps. They're provided by the MediaTransportControls class and include buttons to play, stop, and navigate media, adjust volume, toggle full screen, cast to a second device, enable captions, switch audio tracks, and adjust the playback rate. MediaTransportControls has properties that let you control whether each button is shown and enabled. You can also set the IsCompact property to specify whether the controls are shown in one row or two.

However, there may be scenarios where you need to further customize the look of the control or change its behavior. Here are some examples:

- Change the icons, slider behavior, and colors.
- Move less commonly used command buttons into an overflow menu.
- Change the order in which commands drop out when the control is resized.
- Provide a command button that's not in the default set.

> **Note**
>
> The buttons visible on screen will drop out of the built-in transport controls in a predefined order if there is not enough room on screen. To change this ordering or put commands that don't fit into an overflow menu, you will need to customize the controls.

You can customize the appearance of the control by modifying the default template. To modify the control's behavior or add new commands, you can create a custom control that's derived from MediaTransportControls.

> **Tip**
>
> Customizable control templates are a powerful feature of the XAML platform, but there are also consequences that you should take into consideration. When you customize a template, it becomes a static part of your app and therefore will not receive any platform updates that are made to the template by Microsoft. If template updates are made by Microsoft, you should take the new template and re-modify it in order to get the benefits of the updated template.

Template structure

The **ControlTemplate** is part of the default style. You can copy this default style into your project to modify it. The ControlTemplate is divided into sections similar to other XAML control templates.

- The first section of the template contains the **Style** definitions for the various components of the MediaTransportControls.
- The second section defines the various visual states that are used by the MediaTransportControls.
- The third section contains the **Grid** that holds that various MediaTransportControls elements together and defines how the components are laid out.

> **Note**
>
> For more info about modifying templates, see Control templates. You can use a text editor or similar editors in your IDE to open the XAML files in (*Program Files*)\Windows Kits\10\DesignTime\CommonConfiguration\Neutral\UAP\(*SDK version*)\Generic. The default style and template for each control is defined in the **generic.xaml** file. You can find the MediaTransportControls template in generic.xaml by searching for "MediaTransportControls".

In the following sections, you learn how to customize several of the main elements of the transport controls:

- **Slider**: allows a user to scrub through their media and also displays progress
- **CommandBar**: contains all of the buttons. For more info, see the Anatomy section of the MediaTransportControls reference topic.

Customize the transport controls

If you want to modify only the appearance of the MediaTransportControls, you can create a copy of the default control style and template, and modify that. However, if you also want to add to or modify the functionality of the control, you need to create a new class that derives from MediaTransportControls.

Re-template the control

To customize the MediaTransportControls default style and template

1. Copy the default style from MediaTransportControls styles and templates into a ResourceDictionary in your project.
2. Give the Style an x:Key value to identify it, like this.

XAML

```
<Style TargetType="MediaTransportControls" x:Key="myTransportControlsStyle">
    <!-- Style content ... -->
</Style>
```

3. Add a MediaPlayerElement with MediaTransportControls to your UI.
4. Set the Style property of the MediaTransportControls element to your custom Style resource, as shown here.

XAML

```
<MediaPlayerElement AreTransportControlsEnabled="True">
    <MediaPlayerElement.TransportControls>
        <MediaTransportControls Style="{StaticResource
myTransportControlsStyle}"/>
    </MediaPlayerElement.TransportControls>
</MediaPlayerElement>
```

For more info about modifying styles and templates, see Styling controls and Control templates.

Create a derived control

To add to or modify the functionality of the transport controls, you must create a new class that's derived from MediaTransportControls. A derived class called `CustomMediaTransportControls` is shown in the Media Transport Controls sample and the remaining examples on this page.

To create a new class derived from MediaTransportControls

1. Add a new class file to your project.
    - In Visual Studio, select Project > Add Class. The Add New Item dialog opens.
    - In the Add New Item dialog, enter a name for the class file, then click Add. (In the Media Transport Controls sample, the class is named `CustomMediaTransportControls`.)
2. Modify the class code to derive from the MediaTransportControls class.

C#

```
public sealed class CustomMediaTransportControls : MediaTransportControls
{
}
```

3. Copy the default style for **MediaTransportControls** into a ResourceDictionary in your project. This is the style and template you modify. (In the Media Transport Controls sample, a new folder called "Themes" is created, and a ResourceDictionary file called generic.xaml is added to it.)
4. Change the **TargetType** of the style to the new custom control type. (In the sample, the TargetType is changed to `local:CustomMediaTransportControls`.)

XAML

```
xmlns:local="using:CustomMediaTransportControls">
...
<Style TargetType="local:CustomMediaTransportControls">
```

5. Set the **DefaultStyleKey** of your custom class. This tells your custom class to use a Style with a TargetType of `local:CustomMediaTransportControls`.

C#

```
public sealed class CustomMediaTransportControls : MediaTransportControls
{
    public CustomMediaTransportControls()
    {
        this.DefaultStyleKey = typeof(CustomMediaTransportControls);
    }
}
```

6. Add a **MediaPlayerElement** to your XAML markup and add the custom transport controls to it. One thing to note is that the APIs to hide, show, disable, and enable the default buttons still work with a customized template.

XAML

```
<MediaPlayerElement Name="MediaPlayerElement1"
AreTransportControlsEnabled="True" Source="video.mp4">
    <MediaPlayerElement.TransportControls>
        <local:CustomMediaTransportControls x:Name="customMTC"

IsFastForwardButtonVisible="True"
                                            IsFastForwardEnabled="True"
                                            IsFastRewindButtonVisible="True"
                                            IsFastRewindEnabled="True"

IsPlaybackRateButtonVisible="True"
                                            IsPlaybackRateEnabled="True"
                                            IsCompact="False">
        </local:CustomMediaTransportControls>
    </MediaPlayerElement.TransportControls>
</MediaPlayerElement>
```

You can now modify the control style and template to update the look of your custom control, and the control code to update its behavior.

Working with the overflow menu

You can move MediaTransportControls command buttons into an overflow menu, so that less commonly used commands are hidden until the user needs them.

In the MediaTransportControls template, the command buttons are contained in a **CommandBar** element. The command bar has the concept of primary and secondary commands. The primary commands are the buttons that appear in the control by default and are always visible (unless you disable the button, hide the button or there is not enough room). The secondary commands are shown in an overflow menu that appears when a user clicks the ellipsis (...) button. For more info, see the App bars and command bars article.

To move an element from the command bar primary commands to the overflow menu, you need to edit the XAML control template.

To move a command to the overflow menu:

1. In the control template, find the CommandBar element named `MediaControlsCommandBar`.
2. Add a **SecondaryCommands** section to the XAML for the CommandBar. Put it after the closing tag for the **PrimaryCommands**.

XAML

```
<CommandBar x:Name="MediaControlsCommandBar" ... >
  <CommandBar.PrimaryCommands>
...
    <AppBarButton x:Name='PlaybackRateButton'
                  Style='{StaticResource AppBarButtonStyle}'
                  MediaTransportControlsHelper.DropoutOrder='4'
                  Visibility='Collapsed'>
      <AppBarButton.Icon>
        <FontIcon Glyph="&#xEC57;"/>
      </AppBarButton.Icon>
    </AppBarButton>
...
  </CommandBar.PrimaryCommands>
<!-- Add secondary commands (overflow menu) here -->
  <CommandBar.SecondaryCommands>
    ...
  </CommandBar.SecondaryCommands>
</CommandBar>
```

3. To populate the menu with commands, cut and paste the XAML for the desired **AppBarButton** objects from the PrimaryCommands to the SecondaryCommands. In this example, we move the `PlaybackRateButton` to the overflow menu.

4. Add a label to the button and remove the styling information, as shown here. Because the overflow menu is comprised of text buttons, you must add a text label to the button and also remove the style that sets the height and width of the button. Otherwise, it won't appear correctly in the overflow menu.

XAML

```
<CommandBar.SecondaryCommands>
    <AppBarButton x:Name='PlaybackRateButton'
                  Label='Playback Rate'>
    </AppBarButton>
</CommandBar.SecondaryCommands>
```

> **Important**
>
> You must still make the button visible and enable it in order to use it in the overflow menu. In this example, the PlaybackRateButton element isn't visible in the overflow menu unless the IsPlaybackRateButtonVisible property is true. It's not enabled unless the IsPlaybackRateEnabled property is true. Setting these properties is shown in the previous section.

Adding a custom button

One reason you might want to customize MediaTransportControls is to add a custom command to the control. Whether you add it as a primary command or a secondary command, the procedure for creating the command button and modifying its behavior is the same. In the Media Transport Controls sample, a "rating" button is added to the primary commands.

To add a custom command button

1. Create an AppBarButton object and add it to the CommandBar in the control template.

XAML

```
<AppBarButton x:Name="LikeButton"
              Icon="Like"
              Style="{StaticResource AppBarButtonStyle}"
              MediaTransportControlsHelper.DropoutOrder="3"
              VerticalAlignment="Center" />
```

You must add it to the CommandBar in the appropriate location. (For more information, see the Working with the overflow menu section.) How it's positioned in the UI is determined by where the button is in the markup. For example, if you want this button to appear as the last element in the primary commands, add it at the very end of the primary commands list.

You can also customize the icon for the button. For more information, see the **AppBarButton** reference.

2. In the **OnApplyTemplate** override, get the button from the template and register a handler for its **Click** event. This code goes in the `CustomMediaTransportControls` class.

C#

```
public sealed class CustomMediaTransportControls :  MediaTransportControls
{
    // ...

    protected override void OnApplyTemplate()
    {
        // Find the custom button and create an event handler for its Click
event.
        var likeButton = GetTemplateChild("LikeButton") as Button;
        likeButton.Click += LikeButton_Click;
        base.OnApplyTemplate();
    }

    //...
}
```

3. Add code to the Click event handler to perform the action that occurs when the button is clicked. Here's the complete code for the class.

C#

```
public sealed class CustomMediaTransportControls : MediaTransportControls
{
    public event EventHandler< EventArgs> Liked;

    public CustomMediaTransportControls()
    {
        this.DefaultStyleKey = typeof(CustomMediaTransportControls);
    }

    protected override void OnApplyTemplate()
    {
        // Find the custom button and create an event handler for its Click
event.
```

```
        var likeButton = GetTemplateChild("LikeButton") as Button;
        likeButton.Click += LikeButton_Click;
        base.OnApplyTemplate();
    }

    private void LikeButton_Click(object sender, RoutedEventArgs e)
    {
        // Raise an event on the custom control when 'like' is clicked.
        var handler = Liked;
        if (handler != null)
        {
            handler(this, EventArgs.Empty);
        }
    }
}
```

Modifying the slider

The "seek" control of the MediaTransportControls is provided by a **Slider** element. One way you can customize it is to change the granularity of the seek behavior.

The default seek slider is divided into 100 parts, so the seek behavior is limited to that many sections. You can change the granularity of the seek slider by getting the Slider from the XAML visual tree in your **MediaOpened** event handler on **MediaPlayerElement.MediaPlayer**. This example shows how to use **VisualTreeHelper** to get a reference to the Slider, then change the default step frequency of the slider from 1% to 0.1% (1000 steps) if the media is longer than 120 minutes. The MediaPlayerElement is named `MediaPlayerElement1`.

C#

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
  MediaPlayerElement1.MediaPlayer.MediaOpened +=
MediaPlayerElement_MediaPlayer_MediaOpened;
  base.OnNavigatedTo(e);
}

private void MediaPlayerElement_MediaPlayer_MediaOpened(object sender,
RoutedEventArgs e)
{
  FrameworkElement transportControlsTemplateRoot =
(FrameworkElement)VisualTreeHelper.GetChild(MediaPlayerElement1.TransportCon
trols, 0);
  Slider sliderControl =
(Slider)transportControlsTemplateRoot.FindName("ProgressSlider");
  if (sliderControl != null &&
MediaPlayerElement1.NaturalDuration.TimeSpan.TotalMinutes > 120)
  {
```

```
    // Default is 1%. Change to 0.1% for more granular seeking.
    sliderControl.StepFrequency = 0.1;
  }
}
```

Related articles

- Media playback

Draw shapes

Article • 02/26/2025

Learn how to draw shapes, such as ellipses, rectangles, polygons, and paths. The **Path** class is the way to visualize a fairly complex vector-based drawing language in a XAML UI; for example, you can draw Bezier curves.

✓ **Important APIs**: Path class, Microsoft.UI.Xaml.Shapes namespace, Microsoft.UI.Xaml.Media namespace

Two sets of classes define a region of space in XAML UI: **Shape** classes and **Geometry** classes. The main difference between these classes is that a **Shape** has a brush associated with it and can be rendered to the screen, and a **Geometry** simply defines a region of space and is not rendered unless it helps contribute information to another UI property. You can think of a **Shape** as a **UIElement** with its boundary defined by a **Geometry**. This topic covers mainly the **Shape** classes.

The **Shape** classes are **Line**, **Ellipse**, **Rectangle**, **Polygon**, **Polyline**, and **Path**. **Path** is interesting because it can define an arbitrary geometry, and the **Geometry** class is involved here because that's one way to define the parts of a **Path**.

Fill and Stroke for shapes

For a **Shape** to render to the app canvas, you must associate a **Brush** with it. Set the **Fill** property of the **Shape** to the **Brush** you want. For more info about brushes, see Using brushes.

A **Shape** can also have a **Stroke**, which is a line that is drawn around the shape's perimeter. A **Stroke** also requires a **Brush** that defines its appearance, and should have a non-zero value for **StrokeThickness**. **StrokeThickness** is a property that defines the perimeter's thickness around the shape edge. If you don't specify a **Brush** value for **Stroke**, or if you set **StrokeThickness** to 0, then the border around the shape is not drawn.

Ellipse

An **Ellipse** is a shape with a curved perimeter. To create a basic **Ellipse**, specify a **Width**, **Height**, and a **Brush** for the **Fill**.

The next example creates an **Ellipse** with a **Width** of 200 and a **Height** of 200, and uses a **SteelBlue** colored **SolidColorBrush** as its **Fill**.

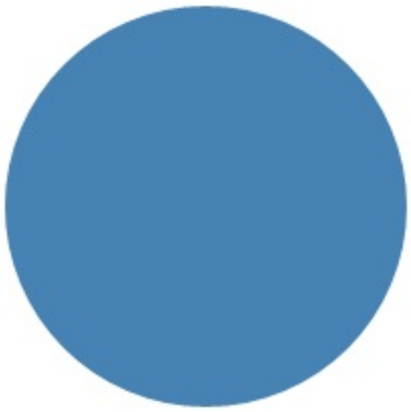
XAML

```
<Ellipse Fill="SteelBlue" Height="200" Width="200" />
```

C#

```
var ellipse1 = new Ellipse();
ellipse1.Fill = new SolidColorBrush(Colors.SteelBlue);
ellipse1.Width = 200;
ellipse1.Height = 200;

// When you create a XAML element in code, you have to add
// it to the XAML visual tree. This example assumes you have
// a panel named 'layoutRoot' in your XAML file, like this:
// <Grid x:Name="layoutRoot>
layoutRoot.Children.Add(ellipse1);
```

Here's the rendered **Ellipse**.



In this case the **Ellipse** is what most people would consider a circle, but that's how you declare a circle shape in XAML: use an **Ellipse** with equal **Width** and **Height**.

When an **Ellipse** is positioned in a UI layout, its size is assumed to be the same as a rectangle with that **Width** and **Height**; the area outside the perimeter does not have rendering but still is part of its layout slot size.

A set of 6 **Ellipse** elements are part of the control template for the **ProgressRing** control, and 2 concentric **Ellipse** elements are part of a **RadioButton**.

Rectangle

A **Rectangle** is a four-sided shape with its opposite sides being equal. To create a basic **Rectangle**, specify a **Width**, a **Height**, and a **Fill**.

You can round the corners of a **Rectangle**. To create rounded corners, specify a value for the **RadiusX** and **RadiusY** properties. These properties specify the x-axis and y-axis of an ellipse that defines the curve of the corners. The maximum allowed value of **RadiusX** is the **Width** divided by two and the maximum allowed value of **RadiusY** is the **Height** divided by two.

The next example creates a **Rectangle** with a **Width** of 200 and a **Height** of 100. It uses a **Blue** value of **SolidColorBrush** for its **Fill** and a **Black** value of **SolidColorBrush** for its **Stroke**. We set the **StrokeThickness** to 3. We set the **RadiusX** property to 50 and the **RadiusY** property to 10, which gives the **Rectangle** rounded corners.

XAML

```
<Rectangle Fill="Blue"
           Width="200"
           Height="100"
           Stroke="Black"
           StrokeThickness="3"
           RadiusX="50"
           RadiusY="10" />
```

C#

```
var rectangle1 = new Rectangle();
rectangle1.Fill = new SolidColorBrush(Colors.Blue);
rectangle1.Width = 200;
rectangle1.Height = 100;
rectangle1.Stroke = new SolidColorBrush(Colors.Black);
rectangle1.StrokeThickness = 3;
rectangle1.RadiusX = 50;
rectangle1.RadiusY = 10;

// When you create a XAML element in code, you have to add
// it to the XAML visual tree. This example assumes you have
// a panel named 'layoutRoot' in your XAML file, like this:
// <Grid x:Name="layoutRoot>
layoutRoot.Children.Add(rectangle1);
```

Here's the rendered **Rectangle**.

There are some scenarios for UI definitions where instead of using a **Rectangle**, a **Border** might be more appropriate. If your intention is to create a rectangle shape around other content, it might be better to use **Border** because it can have child content and will automatically size around that content, rather than using the fixed dimensions for height and width like **Rectangle** does. A **Border** also has the option of having rounded corners if you set the **CornerRadius** property.

On the other hand, a **Rectangle** is probably a better choice for control composition. A **Rectangle** shape is seen in many control templates because it's used as a "FocusVisual" part for focusable controls. Whenever the control is in a "Focused" visual state, this rectangle is made visible, in other states it's hidden.

Polygon

A **Polygon** is a shape with a boundary defined by an arbitrary number of points. The boundary is created by connecting a line from one point to the next, with the last point connected to the first point. The **Points** property defines the collection of points that make up the boundary. In XAML, you define the points with a comma-separated list. In code-behind you use a **PointCollection** to define the points and you add each individual point as a **Point** value to the collection.

You don't need to explicitly declare the points such that the start point and end point are both specified as the same **Point** value. The rendering logic for a **Polygon** assumes that you are defining a closed shape and will connect the end point to the start point implicitly.

The next example creates a **Polygon** with 4 points set to `(10,200)`, `(60,140)`, `(130,140)`, and `(180,200)`. It uses a **LightBlue** value of **SolidColorBrush** for its **Fill**, and has no value for **Stroke** so it has no perimeter outline.

XAML

```
<Polygon Fill="LightBlue"
         Points="10,200,60,140,130,140,180,200" />
```

C#

```
var polygon1 = new Polygon();
polygon1.Fill = new SolidColorBrush(Colors.LightBlue);

var points = new PointCollection();
points.Add(new Windows.Foundation.Point(10, 200));
points.Add(new Windows.Foundation.Point(60, 140));
points.Add(new Windows.Foundation.Point(130, 140));
```

```
points.Add(new Windows.Foundation.Point(180, 200));
polygon1.Points = points;

// When you create a XAML element in code, you have to add
// it to the XAML visual tree. This example assumes you have
// a panel named 'layoutRoot' in your XAML file, like this:
// <Grid x:Name="layoutRoot>
layoutRoot.Children.Add(polygon1);
```

Here's the rendered Polygon.

> **Tip**
>
> A Point value is often used as a type in XAML for scenarios other than declaring the vertices of shapes. For example, a **Point** is part of the event data for touch events, so you can know exactly where in a coordinate space the touch action occurred. For more info about **Point** and how to use it in XAML or code, see the API reference topic for Point.

Line

A Line is simply a line drawn between two points in coordinate space. A **Line** ignores any value provided for Fill, because it has no interior space. For a **Line**, make sure to specify values for the Stroke and StrokeThickness properties, because otherwise the **Line** won't render.

You don't use Point values to specify a Line shape, instead you use discrete Double values for X1, Y1, X2 and Y2. This enables minimal markup for horizontal or vertical lines. For example, `<Line Stroke="Red" X2="400"/>` defines a horizontal line that is 400 pixels long. The other X,Y properties are 0 by default, so in terms of points this XAML would draw a line from `(0,0)` to `(400,0)`. You could then use a TranslateTransform to move the entire **Line**, if you wanted it to start at a point other than (0,0).

XAML

```
<Line Stroke="Red" X2="400"/>
```

C#

```
var line1 = new Line();
line1.Stroke = new SolidColorBrush(Colors.Red);
line1.X2 = 400;

// When you create a XAML element in code, you have to add
// it to the XAML visual tree. This example assumes you have
// a panel named 'layoutRoot' in your XAML file, like this:
// <Grid x:Name="layoutRoot>
layoutRoot.Children.Add(line1);
```

Polyline

A **Polyline** is similar to a **Polygon** in that the boundary of the shape is defined by a set of points, except the last point in a **Polyline** is not connected to the first point.

> ⓘ **Note**
>
> You could explicitly have an identical start point and end point in the **Points** set for the **Polyline**, but in that case you probably could have used a **Polygon** instead.

If you specify a **Fill** of a **Polyline**, the **Fill** paints the interior space of the shape, even if the start point and end point of the **Points** set for the **Polyline** do not intersect. If you do not specify a **Fill**, then the **Polyline** is similar to what would have rendered if you had specified several individual **Line** elements where the start points and end points of consecutive lines intersected.

As with a **Polygon**, the **Points** property defines the collection of points that make up the boundary. In XAML, you define the points with a comma-separated list. In code-behind, you use a **PointCollection** to define the points and you add each individual point as a **Point** structure to the collection.

This example creates a **Polyline** with four points set to `(10,200)`, `(60,140)`, `(130,140)`, and `(180,200)`. A **Stroke** is defined but not a **Fill**.

XAML

```
<Polyline Stroke="Black"
          StrokeThickness="4"
          Points="10,200,60,140,130,140,180,200" />
```

C#

```
var polyline1 = new Polyline();
polyline1.Stroke = new SolidColorBrush(Colors.Black);
polyline1.StrokeThickness = 4;

var points = new PointCollection();
points.Add(new Windows.Foundation.Point(10, 200));
points.Add(new Windows.Foundation.Point(60, 140));
points.Add(new Windows.Foundation.Point(130, 140));
points.Add(new Windows.Foundation.Point(180, 200));
polyline1.Points = points;

// When you create a XAML element in code, you have to add
// it to the XAML visual tree. This example assumes you have
// a panel named 'layoutRoot' in your XAML file, like this:
// <Grid x:Name="layoutRoot>
layoutRoot.Children.Add(polyline1);
```

Here's the rendered Polyline. Notice that the first and last points are not connected by the Stroke outline as they are in a Polygon.

Path

A Path is the most versatile Shape because you can use it to define an arbitrary geometry. But with this versatility comes complexity. Let's now look at how to create a basic **Path** in XAML.

You define the geometry of a path with the Data property. There are two techniques for setting **Data**:

- You can set a string value for Data in XAML. In this form, the **Path.Data** value is consuming a serialization format for graphics. You typically don't text-edit this value in string form after it is first established. Instead, you use design tools that enable you to work in a design or drawing metaphor on a surface. Then you save or export the output, and this gives you a XAML file or XAML string fragment with **Path.Data** information.
- You can set the Data property to a single Geometry object. This can be done in code or in XAML. That single **Geometry** is typically a GeometryGroup, which acts as a container that can composite multiple geometry definitions into a single object for purposes of the object model. The most common reason for doing this

is because you want to use one or more of the curves and complex shapes that can be defined as Segments values for a PathFigure, for example BezierSegment.

This example shows a Path that might have resulted from using Blend for Visual Studio to produce just a few vector shapes and then saving the result as XAML. The total **Path** consists of a Bezier curve segment and a line segment. The example is mainly intended to give you some examples of what elements exist in the Path.Data serialization format and what the numbers represent.

This Data begins with the move command, indicated by "M", which establishes an absolute start point for the path.

The first segment is a cubic Bezier curve that begins at `(100,200)` and ends at `(400,175)`, which is drawn by using the two control points `(100,25)` and `(400,350)`. This segment is indicated by the "C" command in the Data attribute string.

The second segment begins with an absolute horizontal line command "H", which specifies a line drawn from the preceding subpath endpoint `(400,175)` to a new endpoint `(280,175)`. Because it's a horizontal line command, the value specified is an x-coordinate.

XAML

```
<Path Stroke="DarkGoldenRod"
      StrokeThickness="3"
      Data="M 100,200 C 100,25 400,350 400,175 H 280" />
```

Here's the rendered Path.

The next example shows a usage of the other technique we discussed: a GeometryGroup with a PathGeometry. This example exercises some of the contributing geometry types that can be used as part of a **PathGeometry**: PathFigure and the various elements that can be a segment in PathFigure.Segments.

XAML

```
<Path Stroke="Black" StrokeThickness="1" Fill="#CCCCFF">
    <Path.Data>
        <GeometryGroup>
```

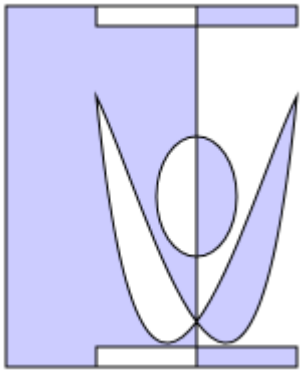
```
            <RectangleGeometry Rect="50,5 100,10" />
            <RectangleGeometry Rect="5,5 95,180" />
            <EllipseGeometry Center="100, 100" RadiusX="20" RadiusY="30"/>
            <RectangleGeometry Rect="50,175 100,10" />
            <PathGeometry>
                <PathGeometry.Figures>
                    <PathFigureCollection>
                        <PathFigure IsClosed="true" StartPoint="50,50">
                            <PathFigure.Segments>
                                <PathSegmentCollection>
                                    <BezierSegment Point1="75,300"
Point2="125,100" Point3="150,50"/>
                                    <BezierSegment Point1="125,300"
Point2="75,100"  Point3="50,50"/>
                                </PathSegmentCollection>
                            </PathFigure.Segments>
                        </PathFigure>
                    </PathFigureCollection>
                </PathGeometry.Figures>
            </PathGeometry>
        </GeometryGroup>
    </Path.Data>
</Path>
```

C#

```
var path1 = new Microsoft.UI.Xaml.Shapes.Path();
path1.Fill = new SolidColorBrush(Windows.UI.Color.FromArgb(255, 204, 204,
255));
path1.Stroke = new SolidColorBrush(Colors.Black);
path1.StrokeThickness = 1;

var geometryGroup1 = new GeometryGroup();
var rectangleGeometry1 = new RectangleGeometry();
rectangleGeometry1.Rect = new Rect(50, 5, 100, 10);
var rectangleGeometry2 = new RectangleGeometry();
rectangleGeometry2.Rect = new Rect(5, 5, 95, 180);
geometryGroup1.Children.Add(rectangleGeometry1);
geometryGroup1.Children.Add(rectangleGeometry2);

var ellipseGeometry1 = new EllipseGeometry();
ellipseGeometry1.Center = new Point(100, 100);
ellipseGeometry1.RadiusX = 20;
ellipseGeometry1.RadiusY = 30;
geometryGroup1.Children.Add(ellipseGeometry1);

var pathGeometry1 = new PathGeometry();
var pathFigureCollection1 = new PathFigureCollection();
var pathFigure1 = new PathFigure();
pathFigure1.IsClosed = true;
pathFigure1.StartPoint = new Windows.Foundation.Point(50, 50);
pathFigureCollection1.Add(pathFigure1);
pathGeometry1.Figures = pathFigureCollection1;
```

```
var pathSegmentCollection1 = new PathSegmentCollection();
var pathSegment1 = new BezierSegment();
pathSegment1.Point1 = new Point(75, 300);
pathSegment1.Point2 = new Point(125, 100);
pathSegment1.Point3 = new Point(150, 50);
pathSegmentCollection1.Add(pathSegment1);

var pathSegment2 = new BezierSegment();
pathSegment2.Point1 = new Point(125, 300);
pathSegment2.Point2 = new Point(75, 100);
pathSegment2.Point3 = new Point(50, 50);
pathSegmentCollection1.Add(pathSegment2);
pathFigure1.Segments = pathSegmentCollection1;

geometryGroup1.Children.Add(pathGeometry1);
path1.Data = geometryGroup1;

// When you create a XAML element in code, you have to add
// it to the XAML visual tree. This example assumes you have
// a panel named 'layoutRoot' in your XAML file, like this:
// <Grid x:Name="layoutRoot">
layoutRoot.Children.Add(path1);
```

Here's the rendered **Path**.



Using **PathGeometry** may be more readable than populating a **Path.Data** string. On the other hand, **Path.Data** uses a syntax compatible with Scalable Vector Graphics (SVG) image path definitions so it may be useful for porting graphics from SVG, or as output from a tool like Blend.

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that

use **WinUI 2**. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for these shapes exist in the Windows.UI.Xaml.Shapes namespace.

- ✓ **UWP APIs:** Path class, Windows.UI.Xaml.Shapes namespace, Windows.UI.Xaml.Media namespace

Web view

Article • 03/16/2023

A web view control embeds a view into your app that renders web content using the Microsoft Edge Legacy rendering engine. Hyperlinks can also appear and function in a web view control.

> **ⓘ Important**
>
> The `WebView2` control is available as part of WinUI3. `WebView2` uses Microsoft Edge (Chromium) as the rendering engine to display web content in apps. For more info, see Introduction to Microsoft Edge WebView2, Getting started with WebView2 in WinUI 3 (Preview), and WebView2 in the WinUI API reference.

Is this the right control?

Use a web view control to display richly formatted HTML content from a remote web server, dynamically generated code, or content files in your app package. Rich content can also contain script code and communicate between the script and your app's code.

Recommendations

- Make sure that the website loaded is formatted correctly for the device and uses colors, typography, and navigation that are consistent with the rest of your app.
- Input fields should be appropriately sized. Users may not realize that they can zoom in to enter text.
- If a web view doesn't look like the rest of your app, consider alternative controls or ways to accomplish relevant tasks. If your web view matches the rest of your app, users will see it all as one seamless experience.

Create a web view

- ✓ **UWP APIs:** WebView class
- ✓ Open the WinUI 2 Gallery app and see the WebView in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

Modify the appearance of a web view

WebView is not a Control subclass, so it doesn't have a control template. However, you can set various properties to control some visual aspects of the web view.

- To constrain the display area, set the Width and Height properties.
- To translate, scale, skew, and rotate a web view, use the RenderTransform property.
- To control the opacity of the web view, set the Opacity property.
- To specify a color to use as the web page background when the HTML content does not specify a color, set the DefaultBackgroundColor property.

Get the web page title

You can get the title of the HTML document currently displayed in the web view by using the DocumentTitle property.

Input events and tab order

Although WebView is not a Control subclass, it will receive keyboard input focus and participate in the tab sequence. It provides a Focus method, and GotFocus and LostFocus events, but it has no tab-related properties. Its position in the tab sequence is the same as its position in the XAML document order. The tab sequence includes all elements in the web view content that can receive input focus.

As indicated in the Events table on the WebView class page, web view doesn't support most of the user input events inherited from UIElement, such as KeyDown, KeyUp, and PointerPressed. Instead, you can use InvokeScriptAsync with the JavaScript **eval** function to use the HTML event handlers, and to use **window.external.notify** from the HTML event handler to notify the application using WebView.ScriptNotify.

Navigating to content

Web view provides several APIs for basic navigation: GoBack, GoForward, Stop, Refresh, CanGoBack, and CanGoForward. You can use these to add typical web browsing capabilities to your app.

To set the initial content of the web view, set the Source property in XAML. The XAML parser automatically converts the string to a Uri.

XAML

```
<!-- Source file is on the web. -->
<WebView x:Name="webView1" Source="http://www.contoso.com"/>

<!-- Source file is in local storage. -->
```

```
<WebView x:Name="webView2" Source="ms-appdata:///local/intro/welcome.html"/>

<!-- Source file is in the app package. -->
<WebView x:Name="webView3" Source="ms-appx-web:///help/about.html"/>
```

The Source property can be set in code, but rather than doing so, you typically use one of the **Navigate** methods to load content in code.

To load web content, use the Navigate method with a **Uri** that uses the http or https scheme.

C#

```
webView1.Navigate(new Uri("http://www.contoso.com"));
```

To navigate to a URI with a POST request and HTTP headers, use the NavigateWithHttpRequestMessage method. This method supports only HttpMethod.Post and HttpMethod.Get for the HttpRequestMessage.Method property value.

To load uncompressed and unencrypted content from your app's LocalFolder or TemporaryFolder data stores, use the **Navigate** method with a **Uri** that uses the ms-appdata scheme. The web view support for this scheme requires you to place your content in a subfolder under the local or temporary folder. This enables navigation to URIs such as ms-appdata:///local/*folder*/*file*.html and ms-appdata:///temp/*folder*/*file*.html . (To load compressed or encrypted files, see NavigateToLocalStreamUri.)

Each of these first-level subfolders is isolated from the content in other first-level subfolders. For example, you can navigate to ms-appdata:///temp/folder1/file.html, but you can't have a link in this file to ms-appdata:///temp/folder2/file.html. However, you can still link to HTML content in the app package using the **ms-appx-web scheme**, and to web content using the **http** and **https** URI schemes.

C#

```
webView1.Navigate(new Uri("ms-appdata:///local/intro/welcome.html"));
```

To load content from the your app package, use the **Navigate** method with a **Uri** that uses the ms-appx-web scheme.

C#

```
webView1.Navigate(new Uri("ms-appx-web:///help/about.html"));
```

You can load local content through a custom resolver using the NavigateToLocalStreamUri method. This enables advanced scenarios such as downloading and caching web-based content for offline use, or extracting content from a compressed file.

Responding to navigation events

The web view control provides several events that you can use to respond to navigation and content loading states. The events occur in the following order for the root web view content: NavigationStarting, ContentLoading, DOMContentLoaded, NavigationCompleted

NavigationStarting - Occurs before the web view navigates to new content. You can cancel navigation in a handler for this event by setting the WebViewNavigationStartingEventArgs.Cancel property to true.

C#

```
webView1.NavigationStarting += webView1_NavigationStarting;

private void webView1_NavigationStarting(object sender,
WebViewNavigationStartingEventArgs args)
{
    // Cancel navigation if URL is not allowed. (Implementation of
IsAllowedUri not shown.)
    if (!IsAllowedUri(args.Uri))
        args.Cancel = true;
}
```

ContentLoading - Occurs when the web view has started loading new content.

C#

```
webView1.ContentLoading += webView1_ContentLoading;

private void webView1_ContentLoading(WebView sender,
WebViewContentLoadingEventArgs args)
{
    // Show status.
    if (args.Uri != null)
    {
        statusTextBlock.Text = "Loading content for " + args.Uri.ToString();
```

```
    }
}
```

DOMContentLoaded - Occurs when the web view has finished parsing the current HTML content.

C#

```
webView1.DOMContentLoaded += webView1_DOMContentLoaded;

private void webView1_DOMContentLoaded(WebView sender,
WebViewDOMContentLoadedEventArgs args)
{
    // Show status.
    if (args.Uri != null)
    {
        statusTextBlock.Text = "Content for " + args.Uri.ToString() + " has
finished loading";
    }
}
```

NavigationCompleted - Occurs when the web view has finished loading the current content or if navigation has failed. To determine whether navigation has failed, check the IsSuccess and WebErrorStatus properties of the WebViewNavigationCompletedEventArgs class.

C#

```
webView1.NavigationCompleted += webView1_NavigationCompleted;

private void webView1_NavigationCompleted(WebView sender,
WebViewNavigationCompletedEventArgs args)
{
    if (args.IsSuccess == true)
    {
        statusTextBlock.Text = "Navigation to " + args.Uri.ToString() + "
completed successfully.";
    }
    else
    {
        statusTextBlock.Text = "Navigation to: " + args.Uri.ToString() +
                               " failed with error " +
args.WebErrorStatus.ToString();
    }
}
```

Similar events occur in the same order for each **iframe** in the web view content:

- FrameNavigationStarting - Occurs before a frame in the web view navigates to new content.
- FrameContentLoading - Occurs when a frame in the web view has started loading new content.
- FrameDOMContentLoaded - Occurs when a frame in the web view has finished parsing its current HTML content.
- FrameNavigationCompleted - Occurs when a frame in the web view has finished loading its content.

Responding to potential problems

You can respond to potential problems with the content such as long running scripts, content that web view can't load, and warnings of unsafe content.

Your app might appear unresponsive while scripts are running. The LongRunningScriptDetected event occurs periodically while the web view executes JavaScript and provides an opportunity to interrupt the script. To determine how long the script has been running, check the ExecutionTime property of the WebViewLongRunningScriptDetectedEventArgs. To halt the script, set the event args StopPageScriptExecution property to **true**. The halted script will not execute again unless it is reloaded during a subsequent web view navigation.

The web view control cannot host arbitrary file types. When an attempt is made to load content that the web view can't host, the UnviewableContentIdentified event occurs. You can handle this event and notify the user, or use the Launcher class to redirect the file to an external browser or another app.

Similarly, the UnsupportedUriSchemeIdentified event occurs when a URI scheme that's not supported is invoked in the web content, such as fbconnect:// or mailto://. You can handle this event to provide custom behavior instead of allowing the default system launcher to launch the URI.

The UnsafeContentWarningDisplayingevent occurs when the web view shows a warning page for content that was reported as unsafe by the SmartScreen Filter. If the user chooses to continue the navigation, subsequent navigation to the page will not display the warning nor fire the event.

Handling special cases for web view content

You can use the ContainsFullScreenElement property and ContainsFullScreenElementChanged event to detect, respond to, and enable full-screen experiences in web content, such as full-screen video playback. For example, you may

use the ContainsFullScreenElementChanged event to resize the web view to occupy the entirety of your app view, or, as the following example illustrates, put a windowed app in full screen mode when a full screen web experience is desired.

C#

```
// Assume webView is defined in XAML
webView.ContainsFullScreenElementChanged +=
webView_ContainsFullScreenElementChanged;

private void webView_ContainsFullScreenElementChanged(WebView sender, object
args)
{
    var applicationView = ApplicationView.GetForCurrentView();

    if (sender.ContainsFullScreenElement)
    {
        applicationView.TryEnterFullScreenMode();
    }
    else if (applicationView.IsFullScreenMode)
    {
        applicationView.ExitFullScreenMode();
    }
}
```

You can use the NewWindowRequested event to handle cases where hosted web content requests a new window to be displayed, such as a popup window. You can use another WebView control to display the contents of the requested window.

Use PermissionRequested event to enable web features that require special capabilities. These currently include geolocation, IndexedDB storage, and user audio and video (for example, from a microphone or webcam). If your app accesses user location or user media, you still are required to declare this capability in the app manifest. For example, an app that uses geolocation needs the following capability declarations at minimum in Package.appxmanifest:

XML

```
  <Capabilities>
    <Capability Name="internetClient" />
    <DeviceCapability Name="location" />
  </Capabilities>
```

In addition to the app handling the PermissionRequested event, the user will have to approve standard system dialogs for apps requesting location or media capabilities in order for these features to be enabled.

Here is an example of how an app would enable geolocation in a map from Bing:

```
C#
```

```
// Assume webView is defined in XAML
webView.PermissionRequested += webView_PermissionRequested;

private void webView_PermissionRequested(WebView sender,
WebViewPermissionRequestedEventArgs args)
{
    if (args.PermissionRequest.PermissionType ==
WebViewPermissionType.Geolocation &&
        args.PermissionRequest.Uri.Host == "www.bing.com")
    {
        args.PermissionRequest.Allow();
    }
}
```

If your app requires user input or other asynchronous operations to respond to a permission request, use the Defer method of WebViewPermissionRequest to create a WebViewDeferredPermissionRequest that can be acted upon at a later time. See WebViewPermissionRequest.Defer.

If users must securely log out of a website hosted in a web view, or in other cases when security is important, call the static method ClearTemporaryWebDataAsync to clear out all locally cached content from a web view session. This prevents malicious users from accessing sensitive data.

Interacting with web view content

You can interact with the content of the web view by using the InvokeScriptAsync method to invoke or inject script into the web view content, and the ScriptNotify event to get information back from the web view content.

To invoke JavaScript inside the web view content, use the InvokeScriptAsync method. The invoked script can return only string values.

For example, if the content of a web view named `webView1` contains a function named `setDate` that takes 3 parameters, you can invoke it like this.

```
C#
```

```
string[] args = {"January", "1", "2000"};
string returnValue = await webView1.InvokeScriptAsync("setDate", args);
```

You can use **InvokeScriptAsync** with the JavaScript **eval** function to inject content into the web page.

Here, the text of a XAML text box (`nameTextBox.Text`) is written to a div in an HTML page hosted in `webView1`.

C#

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
    string functionString =
String.Format("document.getElementById('nameDiv').innerText = 'Hello,
{0}';", nameTextBox.Text);
    await webView1.InvokeScriptAsync("eval", new string[] { functionString
});
}
```

Scripts in the web view content can use **window.external.notify** with a string parameter to send information back to your app. To receive these messages, handle the ScriptNotify event.

To enable an external web page to fire the **ScriptNotify** event when calling window.external.notify, you must include the page's URI in the **ApplicationContentUriRules** section of the app manifest. (You can do this in Microsoft Visual Studio on the Content URIs tab of the Package.appxmanifest designer.) The URIs in this list must use HTTPS, and may contain subdomain wildcards (for example, `https://*.microsoft.com`) but they cannot contain domain wildcards (for example, `https://*.com` and `https://*.*`). The manifest requirement does not apply to content that originates from the app package, uses an ms-local-stream:// URI, or is loaded using NavigateToString.

Accessing the Windows Runtime in a web view

You can use the AddWebAllowedObject method to inject an instance of a native class from a Windows Runtime component into the JavaScript context of the web view. This allows full access to the native methods, properties, and events of that object in the JavaScript content of that web view. The class must be decorated with the AllowForWeb attribute.

For example, this code injects an instance of `MyClass` imported from a Windows Runtime component into a web view.

C#

```
private void webView_NavigationStarting(WebView sender,
WebViewNavigationStartingEventArgs args)
{
    if (args.Uri.Host == "www.contoso.com")
    {
        webView.AddWebAllowedObject("nativeObject", new MyClass());
    }
}
```

For more info, see WebView.AddWebAllowedObject.

In addition, trusted JavaScript content in a web view can be allowed to directly access Windows Runtime APIs. This provides powerful native capabilities for web apps hosted in a web view. To enable this feature, the URI for trusted content must be allowlisted in the ApplicationContentUriRules of the app in Package.appxmanifest, with WindowsRuntimeAccess specifically set to "all".

This example shows a section of the app manifest. Here, a local URI is given access to the Windows Runtime.

XML

```
  <Applications>
    <Application Id="App"
      ...

      <uap:ApplicationContentUriRules>
        <uap:Rule Match="ms-appx-web:///Web/App.html"
WindowsRuntimeAccess="all" Type="include"/>
      </uap:ApplicationContentUriRules>
    </Application>
  </Applications>
```

Options for web content hosting

You can use the WebView.Settings property (of type WebViewSettings) to control whether JavaScript and IndexedDB are enabled. For example, if you use a web view to display strictly static content, you might want to disable JavaScript for best performance.

Capturing web view content

To enable sharing web view content with other apps, use the CaptureSelectedContentToDataPackageAsync method, which returns the selected content as a DataPackage. This method is asynchronous, so you must use a deferral to

prevent your DataRequested event handler from returning before the asynchronous call is complete.

To get a preview image of the web view's current content, use the CapturePreviewToStreamAsync method. This method creates an image of the current content and writes it to the specified stream.

Threading behavior

By default, web view content is hosted on the UI thread on devices in the desktop device family, and off the UI thread on all other devices. You can use the WebView.DefaultExecutionMode static property to query the default threading behavior for the current client. If necessary, you can use the WebView(WebViewExecutionMode) constructor to override this behavior.

> **Note** There might be performance issues when hosting content on the UI thread on mobile devices, so be sure to test on all target devices when you change DefaultExecutionMode.

A web view that hosts content off the UI thread is not compatible with parent controls that require gestures to propagate up from the web view control to the parent, such as FlipView, ScrollViewer, and other related controls. These controls will not be able to receive gestures initiated in the off-thread web view. In addition, printing off-thread web content is not directly supported – you should print an element with WebViewBrush fill instead.

Get the sample code

- WinUI 2 Gallery sample - See all the XAML controls in an interactive format.

Related topics

- WebView class
- Introduction to Microsoft Edge WebView2
- Getting started with WebView2 in WinUI 3 (Preview)
- WebView2

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

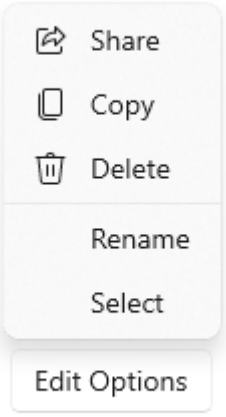
Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Menus and context menus

Article • 10/24/2022

Menus and context menus are similar in how they look and what they can contain. They both display an organized list of commands or options and save space by hiding until the user needs them. However there are differences between them, such as what you should use to create them and how they are accessed by a user.



Is this the right control?

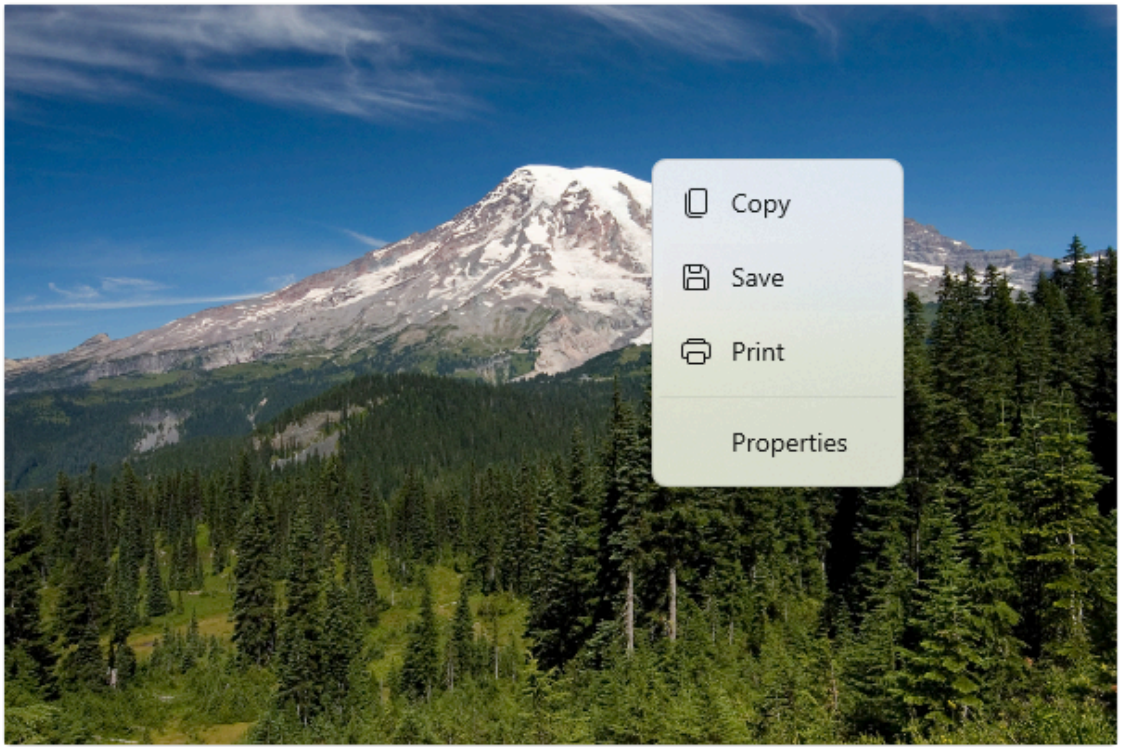
Menus and context menus are for organizing commands and saving space by hiding those commands until the user needs them. To display arbitrary content, such as a notification or confirmation request, use a dialog or a flyout.

If a particular command will be used frequently and you have the space available, consider placing it directly in its own element so that users don't have to go through a menu to get to it.

When should you use a menu or a context menu?

- If the host element is a button or some other command element whose primary role is to present additional commands, use a menu.
- If the host element is some other type of element that has another primary purpose (such as presenting text or an image), use a context menu.

If you want to add commands (such as Cut, Copy, and Paste) to a text or image element, use a context menu instead of a menu. In this scenario, the primary role of the text element is to present and edit text; additional commands (such as Cut, Copy, and Paste) are secondary and belong in a context menu.



Context menus

Context menus have the following characteristics:

- Are attached to a single element and display secondary commands.
- Are invoked by right clicking (or an equivalent action, such as pressing and holding with your finger).
- Are associated with an element via its ContextFlyout property.

In cases where your context menu will include common commands (such as Copy, Cut, Paste, Delete, Share, or text selection commands), use command bar flyout and group these common commands together as primary commands so that they will be shown as a single, horizontal row in the context menu.

In cases where your context menu will not include common commands, either command bar flyout or menu flyout can be used to show a context menu. We recommend using CommandBarFlyout because it provides more functionality than MenuFlyout and, if desired, can achieve the same behavior and look of MenuFlyout by using only secondary commands.

Menus

Menus have the following characteristics:

- Have a single entry point (a File menu at the top of the screen, for example) that is always displayed.
- Are usually attached to a button or a parent menu item.
- Are invoked by left-clicking (or an equivalent action, such as tapping with your finger).
- Are associated with an element via its Flyout or FlyoutBase.AttachedFlyout properties, or grouped in a menu bar at the top of the app window.

When the user invokes a command element (such as a button) whose primary role is to present additional commands, use menu flyout to host a single top-level menu to be shown inline as a flyout attached to the on-canvas UI element. Each MenuFlyout can host menu items and sub-menus. For apps that might need more organization or grouping, use a menu bar as a quick and simple way to show a set of multiple top-level menus in a horizontal row.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the **Windows App SDK** and **WinUI 3**, but are generally applicable to UWP apps that use **WinUI 2**. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The CommandBarFlyout and MenuBar controls for UWP apps are included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for these controls exist in both the Windows.UI.Xaml.Controls and Microsoft.UI.Xaml.Controls namespaces.

- ✓ **UWP APIs:** AppBarButton class, AppBarSeparator class, AppBarToggleButton class, CommandBarFlyout class, ContextFlyout property, FlyoutBase.AttachedFlyout property, MenuBar class, MenuFlyout class, TextCommandBarFlyout class
- ✓ **WinUI 2 Apis:** CommandBarFlyout class, MenuBar class, TextCommandBarFlyout class

- ✔ Open the WinUI 2 Gallery app and see the MenuBar or CommandBarFlyout in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for these controls that uses rounded corners. For more info, see Corner radius.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:CommandBarFlyout />
<muxc:MenuBar />
```

Related articles

- Command design basics for Windows apps
- Contextual commanding for collections and lists
- Menu flyout and menu bar
- Command bar flyout
- Command bar

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

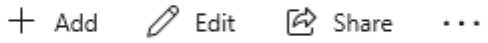
Open a documentation issue

Provide product feedback

Command bar

Article • 02/26/2025

Command bars provide users with easy access to your app's most common tasks. Command bars can provide access to app-level or page-specific commands and can be used with any navigation pattern.



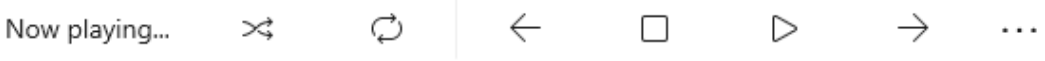
Is this the right control?

The CommandBar control is a general-purpose, flexible, light-weight control that can display both complex content, such as images or text blocks, as well as simple commands such as AppBarButton, AppBarToggleButton, and AppBarSeparator controls.

> **Note**
>
> XAML provides both the **AppBar** control and the **CommandBar** control. You should use the AppBar only when you are upgrading a Universal Windows 8 app that uses the AppBar, and need to minimize changes. For new apps in Windows 10, we recommend using the CommandBar control instead. This document assumes you are using the CommandBar control.

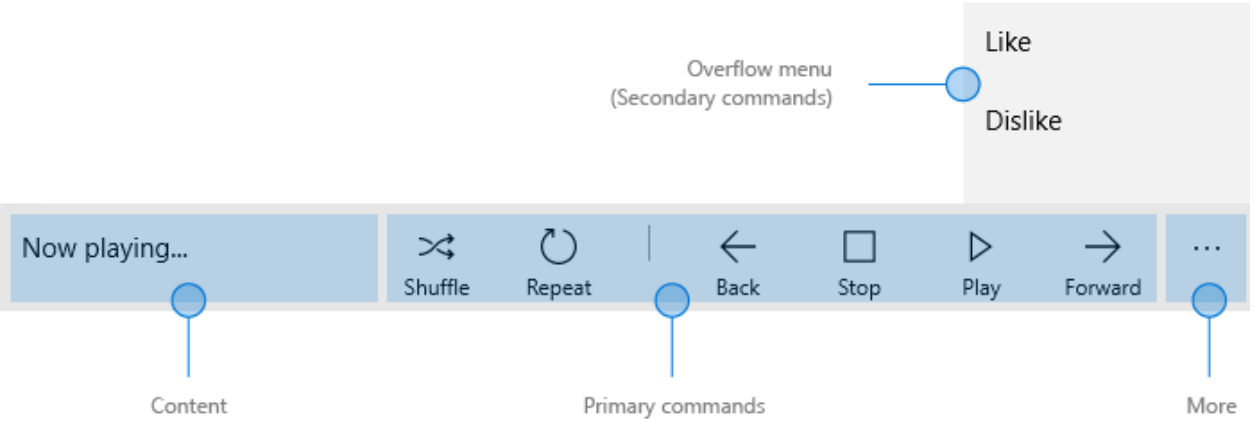
Anatomy

By default, the command bar shows a row of icon buttons and an optional "see more" button, which is represented by an ellipsis [...]. Here's the command bar created by the example code shown later. It's shown in its closed compact state.



The command bar can also be shown in a closed minimal state that looks like this. See the Open and closed states section for more info.

Here's the same command bar in its open state. The labels identify the main parts of the control.



The command bar is divided into 4 main areas:

- The content area is aligned to the left side of the bar. It is shown if the Content property is populated.
- The primary command area is aligned to the right side of the bar. It is shown if the PrimaryCommands property is populated.
- The "see more" [...] button is shown on the right of the bar. Pressing the "see more" [...] button reveals primary command labels and opens the overflow menu if there are secondary commands. The button will not be visible when no primary command labels or secondary labels are present. To change default behavior, use the OverflowButtonVisibility property.
- The overflow menu is shown only when the command bar is open and the SecondaryCommands property is populated. When space is limited, primary commands will move into the SecondaryCommands area. To change default behavior, use the IsDynamicOverflowEnabled property.

The layout is reversed when the FlowDirection is **RightToLeft**.

Placement

Command bars can be placed at the top of the app window, at the bottom of the app window, and inline, by embedding them in a layout control such as `Grid.row`.

- For small handheld devices, we recommend positioning command bars at the bottom of the screen for easy reachability.
- For devices with larger screens, placing command bars near the top of the window makes them more noticeable and discoverable.

Use the DiagonalSizeInInches API to determine physical screen size.

Command bars can be placed in the following screen regions on single-view screens (left example) and on multi-view screens (right example). Inline command bars can be placed anywhere in the action space.

> **Touch devices**: If the command bar must remain visible to a user when the touch keyboard, or Soft Input Panel (SIP), appears then you can assign the command bar to the BottomAppBar property of a Page and it will move to remain visible when the SIP is present. Otherwise, you should place the command bar inline and positioned relative to your app content.

Create a command bar

✓ **Important APIs:** CommandBar class, AppBarButton class, AppBarToggleButton class, AppBarSeparator class

Open the WinUI 3 Gallery app and see the CommandBar in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This example creates the command bar shown previously.

XAML

```
<CommandBar>
    <AppBarToggleButton Icon="Shuffle" Label="Shuffle"
Click="AppBarButton_Click" />
    <AppBarToggleButton Icon="RepeatAll" Label="Repeat"
Click="AppBarButton_Click"/>
    <AppBarSeparator/>
    <AppBarButton Icon="Back" Label="Back" Click="AppBarButton_Click"/>
    <AppBarButton Icon="Stop" Label="Stop" Click="AppBarButton_Click"/>
    <AppBarButton Icon="Play" Label="Play" Click="AppBarButton_Click"/>
    <AppBarButton Icon="Forward" Label="Forward"
Click="AppBarButton_Click"/>

    <CommandBar.SecondaryCommands>
        <AppBarButton Label="Like" Click="AppBarButton_Click"/>
        <AppBarButton Label="Dislike" Click="AppBarButton_Click"/>
    </CommandBar.SecondaryCommands>

    <CommandBar.Content>
        <TextBlock Text="Now playing..." Margin="12,14"/>
    </CommandBar.Content>
</CommandBar>
```

Commands and content

The CommandBar control has 3 properties you can use to add commands and content: PrimaryCommands, SecondaryCommands, and Content.

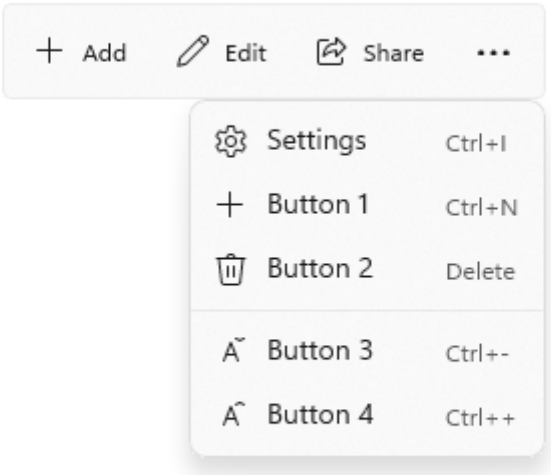
Commands

By default, command bar items are added to the **PrimaryCommands** collection. You should add commands in order of their importance so that the most important commands are always visible. When the command bar width changes, such as when

users resize their app window, primary commands dynamically move between the command bar and the overflow menu at breakpoints. To change this default behavior, use the IsDynamicOverflowEnabled property.

On the smallest screens (320 epx width), a maximum of 4 primary commands fit in the command bar.

You can also add commands to the **SecondaryCommands** collection, which are shown in the overflow menu.



You can programmatically move commands between the PrimaryCommands and SecondaryCommands as needed.

- *If there is a command that would appear consistently across pages, it's best to keep that command in a consistent location.*
- *We recommended placing Accept, Yes, and OK commands to the left of Reject, No, and Cancel. Consistency gives users the confidence to move around the system and helps them transfer their knowledge of app navigation from app to app.*

App bar buttons

Both the PrimaryCommands and SecondaryCommands can be populated only with types that implement the ICommandBarElement interface, which includes AppBarButton, AppBarToggleButton, and AppBarSeparator command elements.

If you'd like to include a different type of element in your PrimaryCommands or SecondaryCommands, you can use the AppBarElementContainer class. This will serve as a wrapper for your element and will enable the element to display in a CommandBar.

The app bar button controls are characterized by an icon and text label. These controls are optimized for use in a command bar, and their appearance changes depending on whether the control is used in the command bar or the overflow menu.

Icons

The size of the icons when shown in the primary command area is 20x20px; in the overflow menu, icons are displayed at 16x16px. If you use SymbolIcon, FontIcon, or PathIcon, the icon will automatically scale to the correct size with no loss of fidelity when the command enters the secondary command area.

For more information and examples of setting the icon, see the documentation for the AppBarButton class.

Labels

The AppBarButton IsCompact property determines whether the label is shown. In a CommandBar control, the command bar overwrites the button's IsCompact property automatically as the command bar is opened and closed.

To position app bar button labels, use CommandBar's DefaultLabelPosition property.

XAML

```
<CommandBar DefaultLabelPosition="Right">
    <AppBarToggleButton Icon="Edit" Label="Edit"/>
    <AppBarToggleButton Icon="Share" Label="Share"/>
</CommandBar>
```

On larger windows, consider moving labels to the right of app bar button icons to improve legibility. Labels on the bottom require users to open the command bar to reveal labels, while labels on the right are visible even when command bar is closed.

In overflow menus, labels are positioned to the right of icons by default, and **LabelPosition** is ignored. You can adjust the styling by setting the CommandBarOverflowPresenterStyle property to a Style that targets the CommandBarOverflowPresenter.

Button labels should be short, preferably a single word. Longer labels below an icon will wrap to multiple lines, increasing the overall height of the opened command bar. You

can include a soft-hyphen character (0x00AD) in the text for a label to hint at the character boundary where a word break should occur. In XAML, you express this using an escape sequence, like this:

XAML

```
<AppBarButton Icon="Back" Label="Areally&#x00AD;longlabel"/>
```

When the label wraps at the hinted location, it looks like this.

SplitButton

You can display a SplitButton in a CommandBar using the built-in `SplitButtonCommandBarStyle` and the AppBarElementContainer class. `SplitButtonCommandBarStyle` provides visuals for a SplitButton to look and feel like an AppBarButton, while `AppBarElementContainer` is a wrapper class that provides the functionality that SplitButton needs to act like an AppBarButton.

When you wrap a SplitButton in an `AppBarElementContainer` and place it in a CommandBar, the `SplitButtonCommandBarStyle` resource is applied automatically.

This sample code creates and displays a SplitButton inside of a CommandBar:

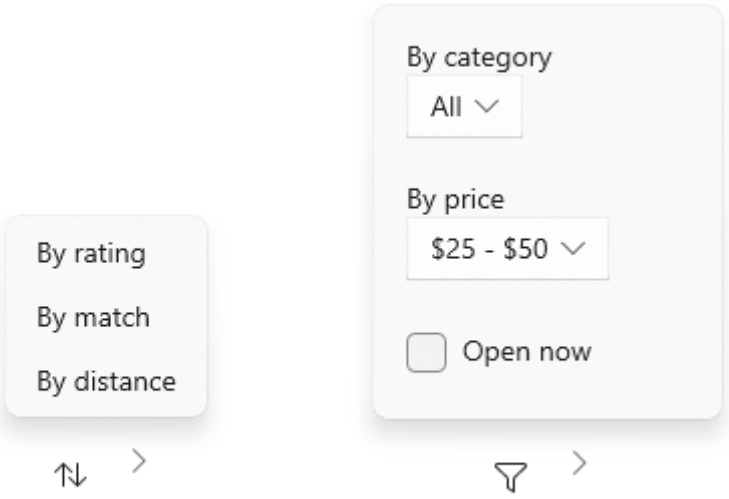
XAML

```
<CommandBar>
    <AppBarButton Icon="Copy" ToolTipService.ToolTip="Copy" Label="Copy"/>
    <AppBarElementContainer>
        <muxc:SplitButton ToolTipService.ToolTip="Insert" Content="Insert">
            <muxc:SplitButton.Flyout>
                <MenuFlyout Placement="RightEdgeAlignedTop">
                    <MenuFlyoutItem Text="Insert above"/>
                    <MenuFlyoutItem Text="Insert between"/>
                    <MenuFlyoutItem  Text="Insert below"/>
                </MenuFlyout>
            </muxc:SplitButton.Flyout>
        </muxc:SplitButton>
    </AppBarElementContainer>
```

```
    <AppBarButton Label="Select all"/>
    <AppBarButton Label="Delete" Icon="Delete"/>
</CommandBar>
```

Menus and flyouts

Consider logical groupings for the commands, such as placing Reply, Reply All, and Forward in a Respond menu. While typically an app bar button activates a single command, an app bar button can be used to show a MenuFlyout or Flyout with custom content.



Other content

You can add any XAML elements to the content area by setting the **Content** property. If you want to add more than one element, you need to place them in a panel container and make the panel the single child of the Content property.

When dynamic overflow is enabled, content will not clip because primary commands can move into the overflow menu. Otherwise, primary commands take precedence and may cause the content to be clipped.

When the ClosedDisplayMode is **Compact**, the content can be clipped if it is larger than the compact size of the command bar. You should handle the Opening and Closed events to show or hide parts of the UI in the content area so that they aren't clipped. See the Open and closed states section for more info.

Open and closed states

The command bar can be open or closed. When it's open, it shows primary command buttons with text labels and it opens the overflow menu (if there are secondary commands). The command bar opens the overflow menu upwards (above the primary commands) or downwards (below the primary commands). The default direction is up, but if there's not enough space to open the overflow menu upwards, the command bar opens it downwards.

A user can switch between these states by pressing the "see more" [...] button. You can switch between them programmatically by setting the IsOpen property.

You can use the Opening, Opened, Closing, and Closed events to respond to the command bar being opened or closed.

- The Opening and Closing events occur before the transition animation begins.
- The Opened and Closed events occur after the transition completes.

In this example, the Opening and Closing events are used to change the opacity of the command bar. When the command bar is closed, it's semi-transparent so the app background shows through. When the command bar is opened, the command bar is made opaque so the user can focus on the commands.

XAML

```
<CommandBar Opening="CommandBar_Opening"
            Closing="CommandBar_Closing">
    <AppBarButton Icon="Accept" Label="Accept"/>
    <AppBarButton Icon="Edit" Label="Edit"/>
    <AppBarButton Icon="Save" Label="Save"/>
    <AppBarButton Icon="Cancel" Label="Cancel"/>
</CommandBar>
```

C#

```
private void CommandBar_Opening(object sender, object e)
{
    CommandBar cb = sender as CommandBar;
    if (cb != null) cb.Background.Opacity = 1.0;
}

private void CommandBar_Closing(object sender, object e)
{
    CommandBar cb = sender as CommandBar;
    if (cb != null) cb.Background.Opacity = 0.5;
}
```

IsSticky

If a user interacts with other parts of an app when a command bar is open, then the command bar will automatically close. This is called *light dismiss*. You can control light dismiss behavior by setting the IsSticky property. When `IsSticky="true"`, the bar remains open until the user presses the "see more" [...] button or selects an item from the overflow menu.

We recommend avoiding sticky command bars because they don't conform to users' expectations for light dismiss and keyboard focus behavior.

Display Mode

You can control how the command bar is shown in its closed state by setting the ClosedDisplayMode property. There are 3 closed display modes to choose from:

- **Compact**: The default mode. Shows content, primary command icons without labels, and the "see more" [...] button.
- **Minimal**: Shows only a thin bar that acts as the "see more" [...] button. The user can press anywhere on the bar to open it.
- **Hidden**: The command bar is not shown when it's closed. This can be useful for showing contextual commands with an inline command bar. In this case, you must open the command bar programmatically by setting the **IsOpen** property or changing the ClosedDisplayMode to **Minimal** or **Compact**.

Here, a command bar is used to hold simple formatting commands for a RichEditBox. When the edit box doesn't have focus, the formatting commands can be distracting, so they're hidden. When the edit box is being used, the command bar's ClosedDisplayMode is changed to Compact so the formatting commands are visible.

XAML

```
<StackPanel Width="300"
            GotFocus="EditStackPanel_GotFocus"
            LostFocus="EditStackPanel_LostFocus">
    <CommandBar x:Name="FormattingCommandBar" ClosedDisplayMode="Hidden">
        <AppBarButton Icon="Bold" Label="Bold"
ToolTipService.ToolTip="Bold"/>
        <AppBarButton Icon="Italic" Label="Italic"
ToolTipService.ToolTip="Italic"/>
        <AppBarButton Icon="Underline" Label="Underline"
ToolTipService.ToolTip="Underline"/>
    </CommandBar>
    <RichEditBox Height="200"/>
</StackPanel>
```

C#

```
private void EditStackPanel_GotFocus(object sender, RoutedEventArgs e)
{
    FormattingCommandBar.ClosedDisplayMode =
AppBarClosedDisplayMode.Compact;
}

private void EditStackPanel_LostFocus(object sender, RoutedEventArgs e)
{
    FormattingCommandBar.ClosedDisplayMode = AppBarClosedDisplayMode.Hidden;
}
```

> **Note**
>
> The implementation of the editing commands is beyond the scope of this example. For more info, see the RichEditBox article.

Although the Minimal and Hidden modes are useful in some situations, keep in mind that hiding all actions could confuse users.

Changing the ClosedDisplayMode to provide more or less of a hint to the user affects the layout of surrounding elements. In contrast, when the CommandBar transitions between closed and open, it does not affect the layout of other elements.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- **UWP APIs:** CommandBar class, AppBarButton class, AppBarToggleButton class, AppBarSeparator class

- ✓ Open the WinUI 2 Gallery app and see the CommandBar in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Automatic styling of a SplitButton in a CommandBar requires that you use the SplitButton control from WinUI 2.6 or later.

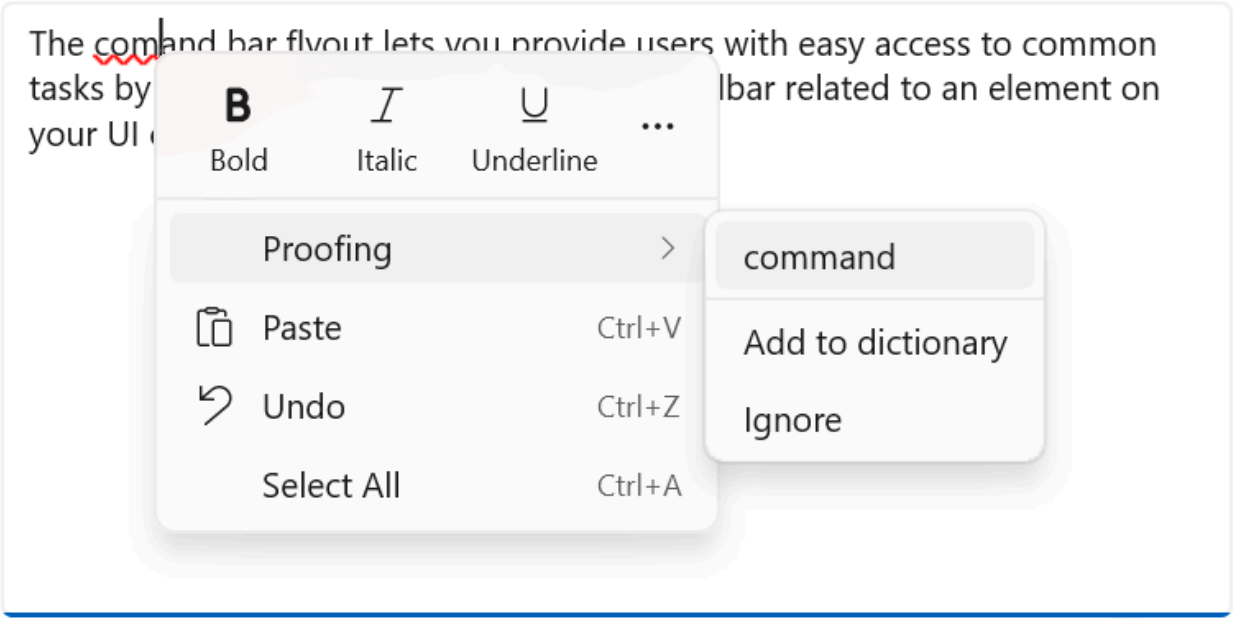
Related articles

- Command design basics for Windows apps
- Command bar flyout
- Menu flyout and menu bar
- MenuFlyout class
- CommandBar class

Command bar flyout

Article • 02/26/2025

The command bar flyout lets you provide users with easy access to common tasks by showing commands in a floating toolbar related to an element on your UI canvas.



Like CommandBar, CommandBarFlyout has **PrimaryCommands** and **SecondaryCommands** properties you can use to add commands. You can place commands in either collection, or both. When and how the primary and secondary commands are displayed depends on the display mode.

The command bar flyout has two display modes: *collapsed* and *expanded*.

- In the collapsed mode, only the primary commands are shown. If your command bar flyout has both primary and secondary commands, a "see more" button, which is represented by an ellipsis [...], is displayed. This lets the user get access to the secondary commands by transitioning to expanded mode.
- In the expanded mode, both the primary and secondary commands are shown. (If the control has only secondary items, they are shown in a way similar to the MenuFlyout control.)

Is this the right control?

Use the command bar flyout control to show a collection of commands to the user, such as buttons and menu items, in the context of an element on the app canvas.

Command bar flyout is the recommended control for creating context menus. This allows the common commands (such as Copy, Cut, Paste, Delete, Share or text selection commands) that are most contextually relevant for the context menu's scenario to be added as primary commands so that they will be shown as a single, horizontal row in the command bar flyout. The TextCommandBarFlyout is already configured appropriately to automatically display text commands in TextBox, TextBlock, RichEditBox, RichTextBlock, and PasswordBox controls. A CommandBarFlyout can be used to replace the default text commands on text controls.

To show contextual commands on list items follow the guidance in Contextual commanding for collections and lists.

Proactive vs. reactive invocation

There are typically two ways to invoke a flyout or menu that's associated with an element on your UI canvas: *proactive invocation* and *reactive invocation*.

In proactive invocation, commands appear automatically when the user interacts with the item that the commands are associated with. For example, text formatting commands might pop up when the user selects text in a text box. In this case, the command bar flyout does not take focus. Instead, it presents relevant commands close to the item the user is interacting with. If the user doesn't interact with the commands, they are dismissed.

In reactive invocation, commands are shown in response to an explicit user action to request the commands; for example, a right-click. This corresponds to the traditional concept of a context menu.

You can use the CommandBarFlyout in either way, or even a mixture of the two.

Create a command bar flyout

✓ **Important APIs:** CommandBarFlyout class, TextCommandBarFlyout class, AppBarButton class, AppBarToggleButton class, AppBarSeparator class

Open the WinUI 3 Gallery app and see the CommandBarFlyout in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This example shows how to create a command bar flyout and use it both proactively and reactively. When the image is tapped, the flyout is shown in its collapsed mode. When shown as a context menu, the flyout is shown in its expanded mode. In either case, the user can expand or collapse the flyout after it's opened.

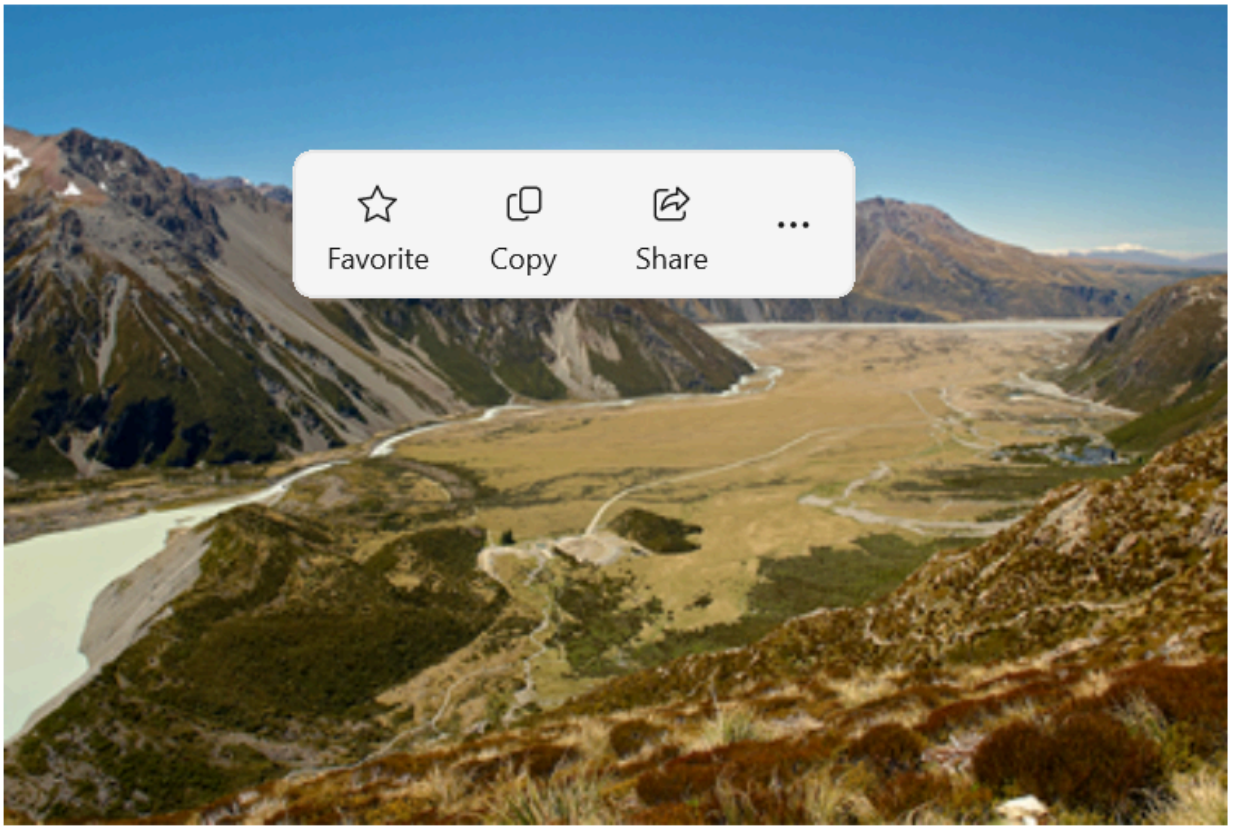
XAML

```
<Grid>
    <Grid.Resources>
        <CommandBarFlyout x:Name="ImageCommandsFlyout">
            <AppBarButton Label="Favorite" Icon="OutlineStar"
ToolTipService.ToolTip="Favorite"/>
            <AppBarButton Label="Copy" Icon="Copy"
ToolTipService.ToolTip="Copy"/>
            <AppBarButton Label="Share" Icon="Share"
ToolTipService.ToolTip="Share"/>
            <CommandBarFlyout.SecondaryCommands>
                <AppBarButton Label="Rotate" Icon="Rotate"/>
                <AppBarButton Label="Delete" Icon="Delete"/>
            </CommandBarFlyout.SecondaryCommands>
        </CommandBarFlyout>
    </Grid.Resources>

    <Image Source="Assets/image1.png" Width="300"
           Tapped="Image_Tapped"
           FlyoutBase.AttachedFlyout="{x:Bind ImageCommandsFlyout}"
           ContextFlyout="{x:Bind ImageCommandsFlyout}"/>
</Grid>
```

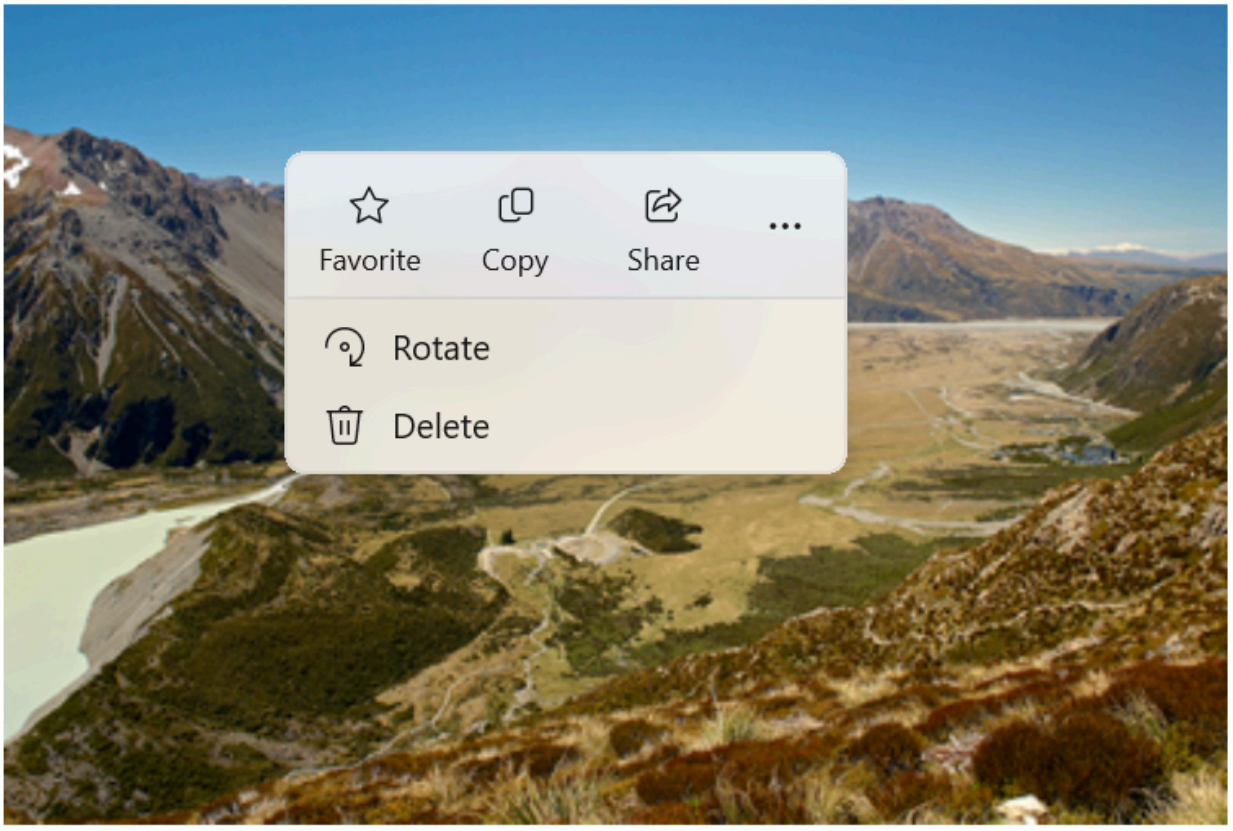
C#

```
private void Image_Tapped(object sender, TappedRoutedEventArgs e)
{
    var flyout = FlyoutBase.GetAttachedFlyout((FrameworkElement)sender);
    var options = new FlyoutShowOptions()
    {
        // Position shows the flyout next to the pointer.
        // "Transient" ShowMode makes the flyout open in its collapsed
state.
        Position = e.GetPosition((FrameworkElement)sender),
        ShowMode = FlyoutShowMode.Transient
    };
    flyout?.ShowAt((FrameworkElement)sender, options);
}
```

Here's the command bar flyout in its collapsed state.



Here's the same command bar flyout in its expanded state showing secondary commands.



Show commands proactively

When you show contextual commands proactively, only the primary commands should be shown by default (the command bar flyout should be collapsed). Place the most important commands in the primary commands collection, and additional commands that would traditionally go in a context menu into the secondary commands collection.

To proactively show commands, you typically handle the Click or Tapped event to show the command bar flyout. Set the flyout's ShowMode to **Transient** or **TransientWithDismissOnPointerMoveAway** to open the flyout in its collapsed mode without taking focus.

Text controls have a **SelectionFlyout** property. When you assign a flyout to this property, it is automatically shown when text is selected.

Show commands reactively

When you show contextual commands reactively, as a context menu, the secondary commands are shown by default (the command bar flyout should be expanded). In this case, the command bar flyout might have both primary and secondary commands, or secondary commands only.

To show commands in a context menu, you typically assign the flyout to the ContextFlyout property of a UI element. This way, opening the flyout is handled by the element, and you don't need to do anything more.

If you handle showing the flyout yourself (for example, on a RightTapped event), set the flyout's ShowMode to **Standard** to open the flyout in its expanded mode and give it focus.

> **Tip**
>
> For more info about options when showing a flyout and how to control placement of the flyout, see **Flyouts**.

Show an always expanded CommandBarFlyout

When you have primary and secondary commands in a CommandBarFlyout, the "see more" [...] button is displayed by default, and can be used to expand and collapse the secondary commands. If you'd like to keep your CommandBarFlyout in expanded mode and show the secondary commands at all times, you can use the `CommandBarFlyout.AlwaysExpanded` property.

When the `AlwaysExpanded` property is set to `true`, the "see more" button is not shown, and the user is not able to toggle the expanded state of the control. The CommandBarFlyout will still dismiss as usual when a secondary command is clicked or the user clicks outside of the flyout.

This property only has an effect if the CommandBarFlyout has secondary commands. If there are no secondary commands, the CommandBarFlyout will always be in collapsed mode.

> **Tip**
>
> You can still collapse and expand the CommandBarFlyout programmatically by setting the **IsOpen** property even when the `AlwaysExpanded` property is set to `true`.

Commands and content

The CommandBarFlyout control has 2 properties you can use to add commands and content: PrimaryCommands and SecondaryCommands.

By default, command bar items are added to the **PrimaryCommands** collection. These commands are shown in the command bar and are visible in both the collapsed and expanded modes. Unlike CommandBar, primary commands do not automatically overflow to the secondary commands and might be truncated.

You can also add commands to the **SecondaryCommands** collection. Secondary commands are shown in the menu portion of the control and are visible only in the expanded mode.

If there are **common commands** (such as Copy, Cut, Paste, Delete, Share or text selection commands) that are important to the scenario, it is recommended to add them as primary commands rather than secondary commands.

App bar buttons

You can populate the PrimaryCommands and SecondaryCommands directly with AppBarButton, AppBarToggleButton, and AppBarSeparator controls.

The app bar button controls are characterized by an icon and text label. These controls are optimized for use in a command bar, and their appearance changes depending on whether the control is shown in the command bar or the overflow menu.

- **In Windows App SDK 1.5 and later:** App bar buttons used as primary commands are shown in the command bar with both the text label and icon (if both are set).

  XAML

  ```
  <AppBarButton Icon="Copy" Label="Copy"/>
  ```

- **In Windows App SDK 1.4 and earlier:** App bar buttons used as primary commands are shown in the command bar with only their icon; the text label is not shown. We recommend that you use a tooltip to show a text description of the command, as shown here.

  XAML

  ```
  <AppBarButton Icon="Copy" ToolTipService.ToolTip="Copy"/>
  ```

- App bar buttons used as secondary commands are shown in the menu, with both the label and icon visible.

Icons

Consider providing menu item icons for:

- The most commonly used items.
- Menu items whose icon is standard or well known.
- Menu items whose icon well illustrates what the command does.

Don't feel obligated to provide icons for commands that don't have a standard visualization. Cryptic icons aren't helpful, create visual clutter, and prevent users from focusing on the important menu items.

Other content

You can add other controls to a command bar flyout by wrapping them in an AppBarElementContainer. This lets you add controls like DropDownButton or SplitButton, or add containers like StackPanel to create more complex UI.

In order to be added to the primary or secondary command collections of a command bar flyout, an element must implement the ICommandBarElement interface. AppBarElementContainer is a wrapper that implements this interface so you can add an element to a command bar even if it doesn't implement the interface itself.

Here, an AppBarElementContainer is used to add extra elements to a command bar flyout. A SplitButton is added to the primary commands to enable text alignment. A StackPanel is added to the secondary commands to allow a more complex layout for zoom controls.

> **Tip**
>
> By default, elements designed for the app canvas might not look right in a command bar. When you add an element using AppBarElementContainer, there are some steps you should take to make the element match other command bar elements:
>
> - Override the default brushes with **lightweight styling** to make the element's background and border match the app bar buttons.
> - Adjust the size and position of the element.
> - Wrap icons in a Viewbox with a Width and Height of 16px.

> **Note**
>
> This example shows only the command bar flyout UI, it does not implement any of the commands that are shown. For more info about implementing the commands, see Buttons and Command design basics.

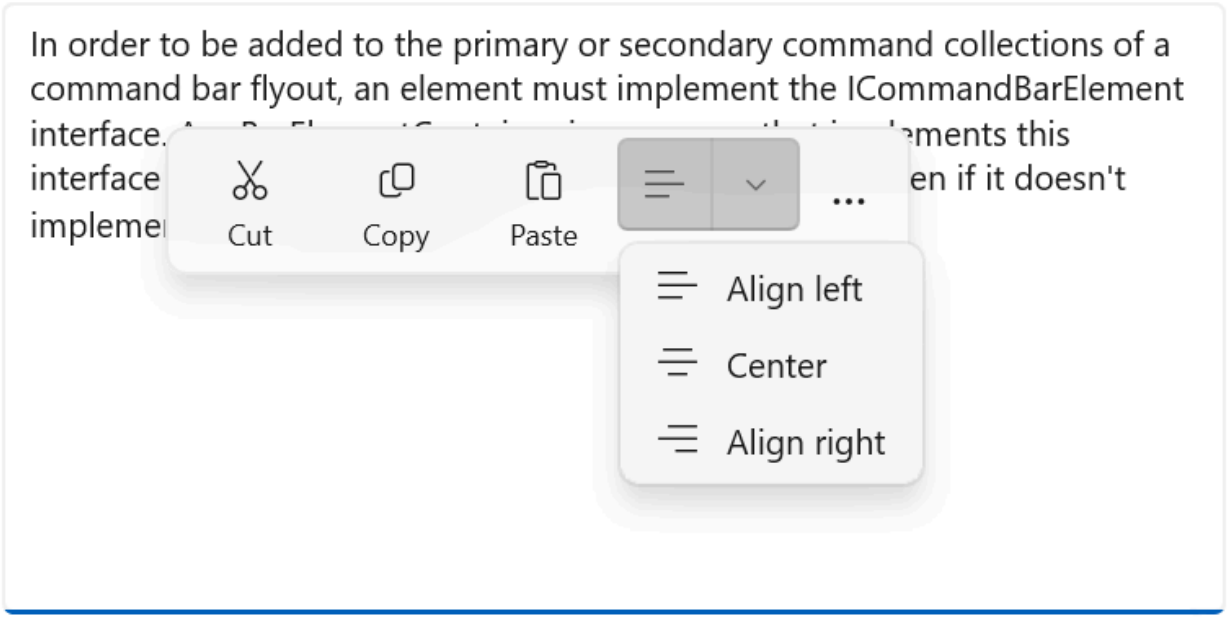
XAML

```
<CommandBarFlyout>
    <AppBarButton Icon="Cut" Label="Cut" ToolTipService.ToolTip="Cut"/>
    <AppBarButton Icon="Copy" Label="Copy" ToolTipService.ToolTip="Copy"/>
    <AppBarButton Icon="Paste" Label="Paste"
ToolTipService.ToolTip="Paste"/>
    <!-- Alignment controls -->
    <AppBarElementContainer>
         <SplitButton ToolTipService.ToolTip="Alignment">
            <SplitButton.Resources>
                <!-- Override default brushes to make the SplitButton
                     match other command bar elements. -->
                <Style TargetType="SplitButton">
                    <Setter Property="Height" Value="38"/>
                </Style>
                <SolidColorBrush x:Key="SplitButtonBackground"
                                 Color="Transparent"/>
                <SolidColorBrush x:Key="SplitButtonBackgroundPressed"
                                 Color="{ThemeResource
SystemListMediumColor}"/>
                <SolidColorBrush x:Key="SplitButtonBackgroundPointerOver"
```

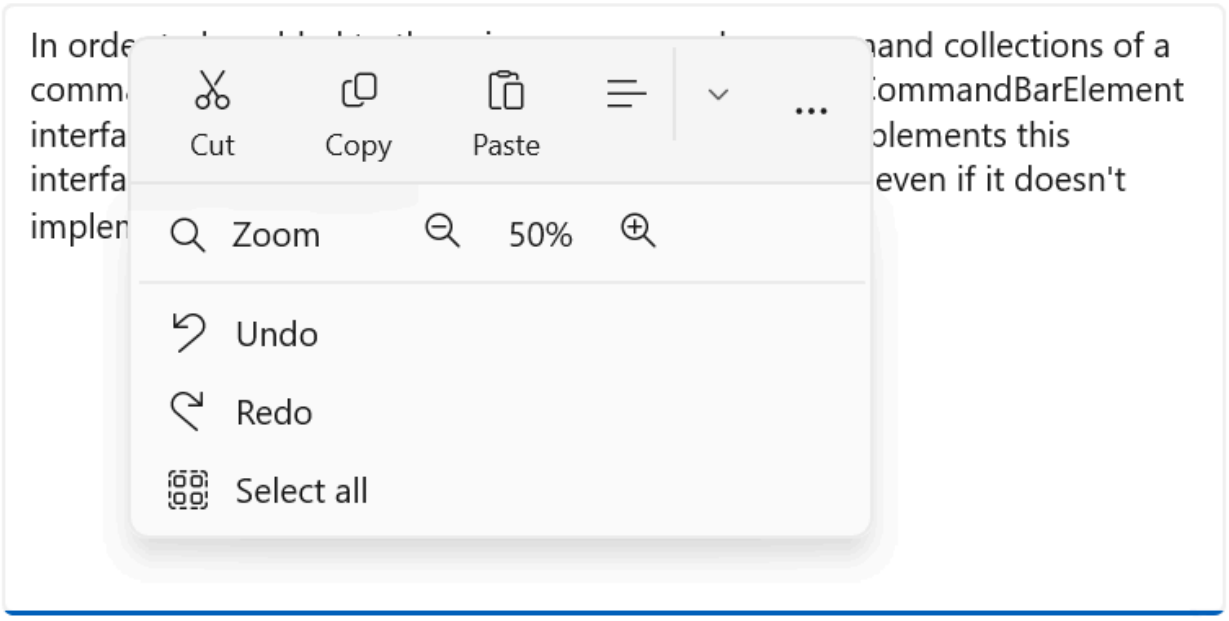
```
                                Color="{ThemeResource
SystemListLowColor}"/>
                <SolidColorBrush x:Key="SplitButtonBorderBrush"
Color="Transparent"/>
                <SolidColorBrush x:Key="SplitButtonBorderBrushPointerOver"
                                 Color="Transparent"/>
                <SolidColorBrush x:Key="SplitButtonBorderBrushChecked"
                                 Color="Transparent"/>
            </SplitButton.Resources>
            <SplitButton.Content>
                <Viewbox Width="16" Height="16" Margin="0,2,0,0">
                    <SymbolIcon Symbol="AlignLeft"/>
                </Viewbox>
            </SplitButton.Content>
            <SplitButton.Flyout>
                <MenuFlyout>
                    <MenuFlyoutItem Icon="AlignLeft" Text="Align left"/>
                    <MenuFlyoutItem Icon="AlignCenter" Text="Center"/>
                    <MenuFlyoutItem Icon="AlignRight" Text="Align right"/>
                </MenuFlyout>
            </SplitButton.Flyout>
        </SplitButton>
    </AppBarElementContainer>
    <!-- end Alignment controls -->
    <CommandBarFlyout.SecondaryCommands>
        <!-- Zoom controls -->
        <AppBarElementContainer>
            <AppBarElementContainer.Resources>
                <!-- Override default brushes to make the Buttons
                     match other command bar elements. -->
                <SolidColorBrush x:Key="ButtonBackground"
                                 Color="Transparent"/>
                <SolidColorBrush x:Key="ButtonBackgroundPressed"
                                 Color="{ThemeResource
SystemListMediumColor}"/>
                <SolidColorBrush x:Key="ButtonBackgroundPointerOver"
                                 Color="{ThemeResource
SystemListLowColor}"/>
                <SolidColorBrush x:Key="ButtonBorderBrush"
                                 Color="Transparent"/>
                <SolidColorBrush x:Key="ButtonBorderBrushPointerOver"
                                 Color="Transparent"/>
                <SolidColorBrush x:Key="ButtonBorderBrushChecked"
                                 Color="Transparent"/>
                <Style TargetType="TextBlock">
                    <Setter Property="VerticalAlignment" Value="Center"/>
                </Style>
                <Style TargetType="Button">
                    <Setter Property="Height" Value="40"/>
                    <Setter Property="Width" Value="40"/>
                </Style>
            </AppBarElementContainer.Resources>
            <Grid Margin="12,-4">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto"/>
```

```
                    <ColumnDefinition Width="76"/>
                    <ColumnDefinition Width="Auto"/>
                </Grid.ColumnDefinitions>
                <Viewbox Width="16" Height="16" Margin="0,2,0,0">
                    <SymbolIcon Symbol="Zoom"/>
                </Viewbox>
                <TextBlock Text="Zoom" Margin="10,0,0,0" Grid.Column="1"/>
                <StackPanel Orientation="Horizontal" Grid.Column="2">
                    <Button ToolTipService.ToolTip="Zoom out">
                        <Viewbox Width="16" Height="16">
                            <SymbolIcon Symbol="ZoomOut"/>
                        </Viewbox>
                    </Button>
                    <TextBlock Text="50%" Width="40"
                               HorizontalTextAlignment="Center"/>
                    <Button ToolTipService.ToolTip="Zoom in">
                        <Viewbox Width="16" Height="16">
                            <SymbolIcon Symbol="ZoomIn"/>
                        </Viewbox>
                    </Button>
                </StackPanel>
            </Grid>
        </AppBarElementContainer>
        <!-- end Zoom controls -->
        <AppBarSeparator/>
        <AppBarButton Label="Undo" Icon="Undo"/>
        <AppBarButton Label="Redo" Icon="Redo"/>
        <AppBarButton Label="Select all" Icon="SelectAll"/>
    </CommandBarFlyout.SecondaryCommands>
</CommandBarFlyout>
```

Here's the collapsed command bar flyout with an open SplitButton.



Here's the expanded command bar flyout with custom zoom UI in the menu.



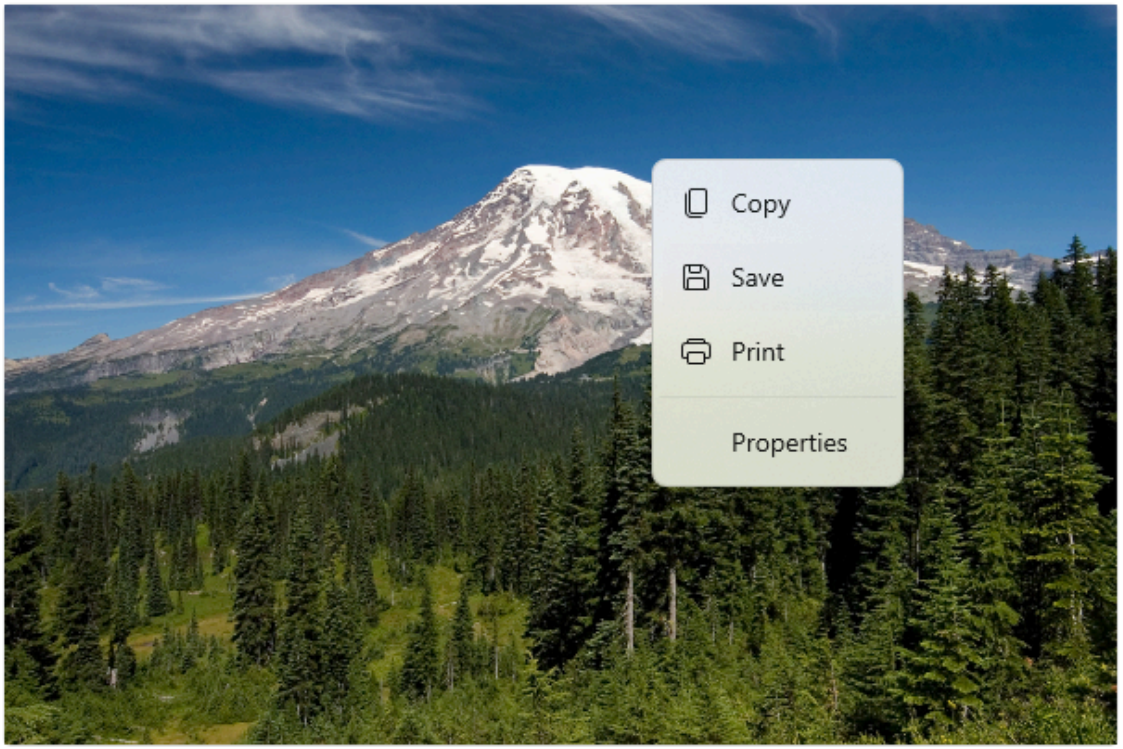
Create a context menu with secondary commands only

You can use a command bar flyout with only secondary commands to create a context menu that achieves the same look and behavior of menu flyout.

XAML

```
<Grid>
    <Grid.Resources>
        <!-- A command bar flyout with only secondary commands. -->
        <CommandBarFlyout x:Name="ContextMenu">
            <CommandBarFlyout.SecondaryCommands>
                <AppBarButton Label="Copy" Icon="Copy"/>
                <AppBarButton Label="Save" Icon="Save"/>
                <AppBarButton Label="Print" Icon="Print"/>
                <AppBarSeparator />
                <AppBarButton Label="Properties"/>
            </CommandBarFlyout.SecondaryCommands>
        </CommandBarFlyout>
    </Grid.Resources>

    <Image Source="Assets/image1.png" Width="300"
           ContextFlyout="{x:Bind ContextMenu}"/>
</Grid>
```

Here's the command bar flyout as a context menu.



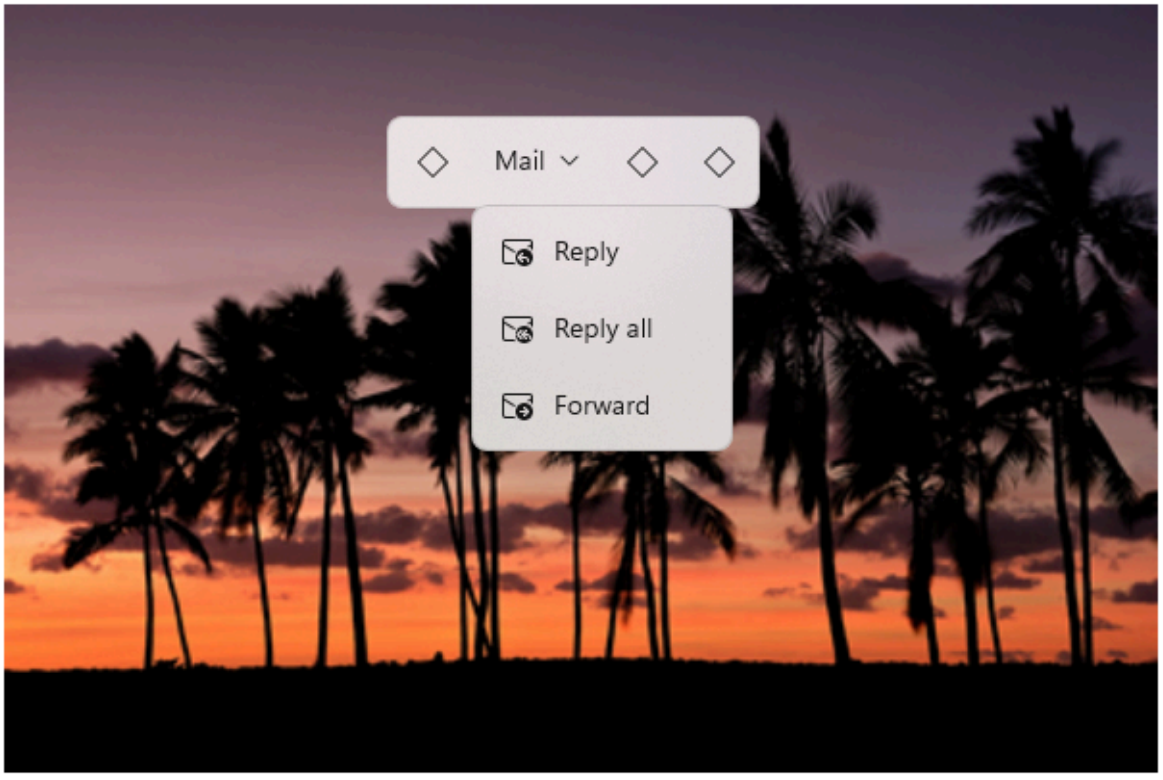
You can also use a CommandBarFlyout with a DropDownButton to create a standard menu.

XAML

```
<CommandBarFlyout>
    <AppBarButton Icon="Placeholder"/>
    <AppBarElementContainer>
        <DropDownButton Content="Mail">
            <DropDownButton.Resources>
                <!-- Override default brushes to make the DropDownButton
                     match other command bar elements. -->
                <Style TargetType="DropDownButton">
                    <Setter Property="Height" Value="38"/>
                </Style>
                <SolidColorBrush x:Key="ButtonBackground"
                                 Color="Transparent"/>
                <SolidColorBrush x:Key="ButtonBackgroundPressed"
                                 Color="{ThemeResource
SystemListMediumColor}"/>
                <SolidColorBrush x:Key="ButtonBackgroundPointerOver"
                                 Color="{ThemeResource
SystemListLowColor}"/>

                <SolidColorBrush x:Key="ButtonBorderBrush"
                                 Color="Transparent"/>
                <SolidColorBrush x:Key="ButtonBorderBrushPointerOver"
                                 Color="Transparent"/>
                <SolidColorBrush x:Key="ButtonBorderBrushChecked"
                                 Color="Transparent"/>
```

```
            </DropDownButton.Resources>
            <DropDownButton.Flyout>
                <CommandBarFlyout Placement="BottomEdgeAlignedLeft">
                    <CommandBarFlyout.SecondaryCommands>
                        <AppBarButton Icon="MailReply" Label="Reply"/>
                        <AppBarButton Icon="MailReplyAll" Label="Reply
all"/>
                        <AppBarButton Icon="MailForward" Label="Forward"/>
                    </CommandBarFlyout.SecondaryCommands>
                </CommandBarFlyout>
            </DropDownButton.Flyout>
        </DropDownButton>
    </AppBarElementContainer>
    <AppBarButton Icon="Placeholder"/>
    <AppBarButton Icon="Placeholder"/>
</CommandBarFlyout>
```

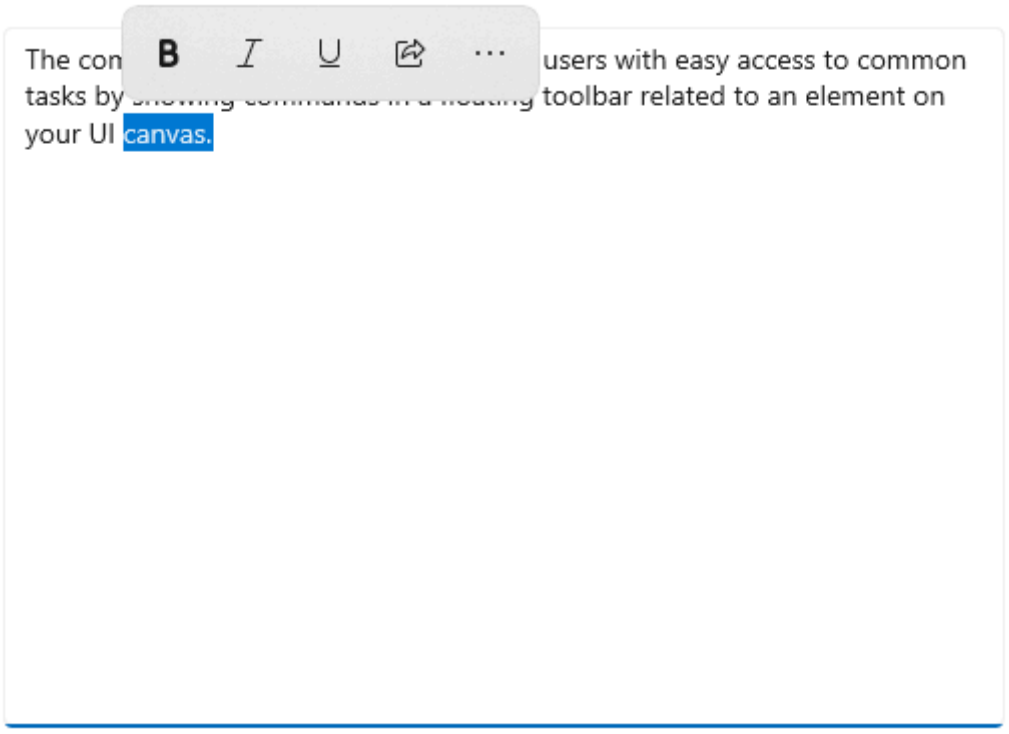
Here's a drop down button menu in a command bar flyout.



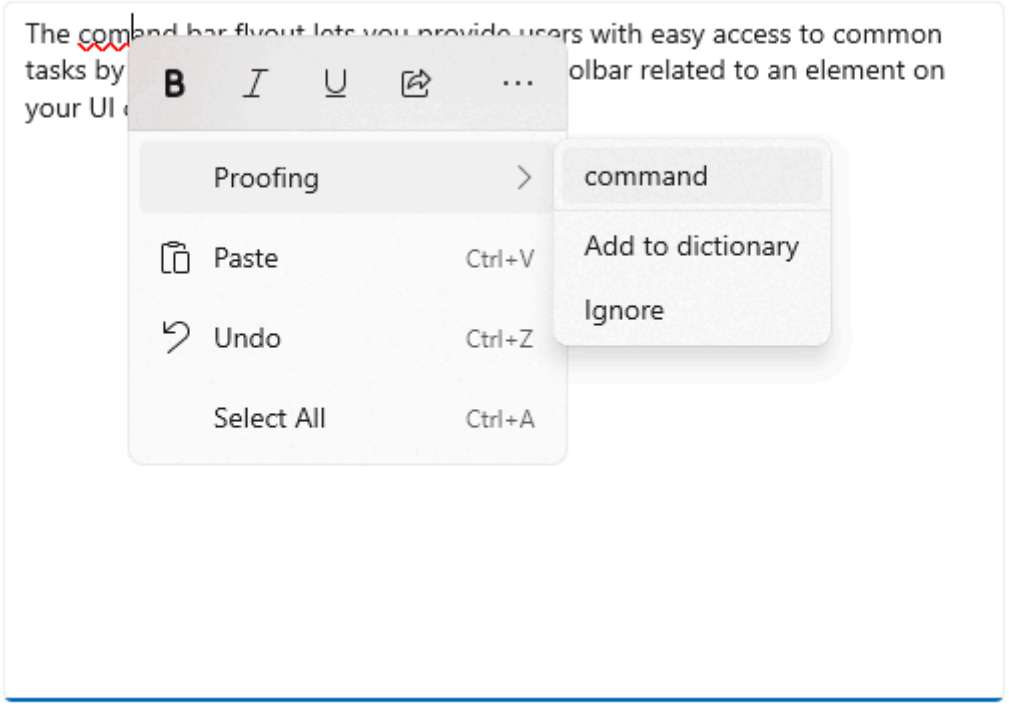
Command bar flyouts for text controls

The TextCommandBarFlyout is a specialized command bar flyout that contains commands for editing text. Each text control shows the TextCommandBarFlyout automatically as a context menu (right-click), or when text is selected. The text command bar flyout adapts to the text selection to only show relevant commands.

Here's a text command bar flyout on text selection.



Here's an expanded text command bar flyout that shows the secondary commands.



Available commands

This table shows the commands that are included in a TextCommandBarFlyout, and when they are shown.

Expand table

Command	Shown...
Bold	when the text control is not read-only (RichEditBox only).
Italic	when the text control is not read-only (RichEditBox only).
Underline	when the text control is not read-only (RichEditBox only).
Proofing	when IsSpellCheckEnabled is **true** and misspelled text is selected.
Cut	when the text control is not read-only and text is selected.
Copy	when text is selected.
Paste	when the text control is not read-only and the clipboard has content.
Undo	when there is an action that can be undone.
Select all	when text can be selected.

Custom text command bar flyouts

TextCommandBarFlyout can't be customized, and is managed automatically by each text control. However, you can replace the default TextCommandBarFlyout with custom commands.

- To replace the default TextCommandBarFlyout that's shown on text selection, you can create a custom CommandBarFlyout (or other flyout type) and assign it to the **SelectionFlyout** property. If you set SelectionFlyout to **null**, no commands are shown on selection.
- To replace the default TextCommandBarFlyout that's shown as the context menu, assign a custom CommandBarFlyout (or other flyout type) to the **ContextFlyout** property on a text control. If you set ContextFlyout to **null**, the menu flyout shown in previous versions of the text control is shown instead of the TextCommandBarFlyout.

Light dismiss

Light dismiss controls–such as menus, context menus, and other flyouts–trap keyboard and gamepad focus inside the transient UI until dismissed. To provide a visual cue for

this behavior, light dismiss controls on Xbox will draw an overlay that dims the visibility of out of scope UI. This behavior can be modified with the LightDismissOverlayMode property. By default, transient UIs will draw the light dismiss overlay on Xbox (**Auto**) but not other device families. You can choose to force the overlay to be always **On** or always **Off**.

XAML

```
<CommandBarFlyout LightDismissOverlayMode="Off" /> >
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The CommandBarFlyout control for UWP apps is included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in both the Windows.UI.Xaml.Controls (UWP) and Microsoft.UI.Xaml.Controls (WinUI) namespaces.

- ✓ **UWP APIs:** CommandBarFlyout class, TextCommandBarFlyout class, AppBarButton class, AppBarToggleButton class, AppBarSeparator class
- ✓ **WinUI 2 Apis:** CommandBarFlyout class, TextCommandBarFlyout class
- ✓ Open the WinUI 2 Gallery app and see the CommandBarFlyout in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started

with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:CommandBarFlyout />
```

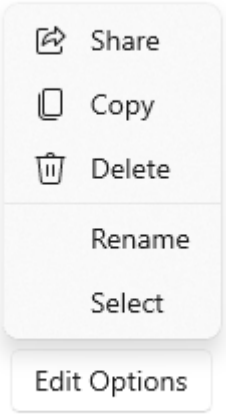
Related articles

- Command design basics for Windows apps
- Contextual commanding for collections and lists.
- Menus and context menus
- Command bar
- CommandBar class
- CommandBarFlyout class

Menu flyout and menu bar

Article • 02/26/2025

Menu flyouts are used in menu and context menu scenarios to display a list of commands or options when requested by the user. A menu flyout shows a single, inline, top-level menu that can have menu items and sub-menus. To show a set of multiple top-level menus in a horizontal row, use menu bar (which you typically position at the top of the app window).



Is this the right control?

See menus and context menus for help identifying menu vs. context menu scenarios and guidance on when to use menu flyout vs. command bar flyout.

Menu flyouts can be used as menus and context menus to organize commands. To display arbitrary content, such as a notification or confirmation request, use a dialog or a flyout.

If a particular command will be used frequently and you have the space available, see collection commanding for examples on placing a command directly in its own element so that users don't have to go through a menu to get to it.

Create a menu flyout

- ✓ **Important APIs:** MenuFlyout class, ContextFlyout property, FlyoutBase.AttachedFlyout property

Open the WinUI 3 Gallery app and see MenuFlyout in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

To create a menu flyout, you use the MenuFlyout class. You define the contents of the menu by adding MenuFlyoutItem, MenuFlyoutSubItem, ToggleMenuFlyoutItem, RadioMenuFlyoutItem and MenuFlyoutSeparator objects to the MenuFlyout.

These objects are for:

- MenuFlyoutItem—Performing an immediate action.
- MenuFlyoutSubItem—Containing a cascading list of menu items.
- ToggleMenuFlyoutItem—Switching an option on or off.
- RadioMenuFlyoutItem—Switching between mutually-exclusive menu items.
- MenuFlyoutSeparator—Visually separating menu items.

This example creates a MenuFlyout and uses the ContextFlyout property, a property available to most controls, to show the MenuFlyout as a context menu.

XAML

```
<Rectangle
  Height="100" Width="100">
  <Rectangle.ContextFlyout>
    <MenuFlyout>
      <MenuFlyoutItem Text="Change color" Click="ChangeColorItem_Click" />
    </MenuFlyout>
  </Rectangle.ContextFlyout>
  <Rectangle.Fill>
    <SolidColorBrush x:Name="rectangleFill" Color="Red" />
  </Rectangle.Fill>
</Rectangle>
```

C#

```
private void ChangeColorItem_Click(object sender, RoutedEventArgs e)
{
    // Change the color from red to blue or blue to red.
    if (rectangleFill.Color == Windows.UI.Colors.Red)
    {
        rectangleFill.Color = Windows.UI.Colors.Blue;
    }
    else
    {
        rectangleFill.Color = Windows.UI.Colors.Red;
```

```
    }
}
```

The next example is nearly identical, but instead of using the ContextFlyout property to show the MenuFlyout class as a context menu, the example uses the FlyoutBase.ShowAttachedFlyout property to show it as a menu.

XAML

```
<Rectangle
  Height="100" Width="100"
  Tapped="Rectangle_Tapped">
  <FlyoutBase.AttachedFlyout>
    <MenuFlyout>
      <MenuFlyoutItem Text="Change color" Click="ChangeColorItem_Click" />
    </MenuFlyout>
  </FlyoutBase.AttachedFlyout>
  <Rectangle.Fill>
    <SolidColorBrush x:Name="rectangleFill" Color="Red" />
  </Rectangle.Fill>
</Rectangle>
```

C#

```
private void Rectangle_Tapped(object sender, TappedRoutedEventArgs e)
{
    FlyoutBase.ShowAttachedFlyout((FrameworkElement)sender);
}

private void ChangeColorItem_Click(object sender, RoutedEventArgs e)
{
    // Change the color from red to blue or blue to red.
    if (rectangleFill.Color == Windows.UI.Colors.Red)
    {
        rectangleFill.Color = Windows.UI.Colors.Blue;
    }
    else
    {
        rectangleFill.Color = Windows.UI.Colors.Red;
    }
}
```

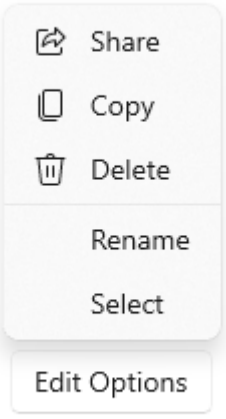
Icons

Consider providing menu item icons for:

- The most commonly used items.
- Menu items whose icon is standard or well known.

- Menu items whose icon well illustrates what the command does.

Don't feel obligated to provide icons for commands that don't have a standard visualization. Cryptic icons aren't helpful, create visual clutter, and prevent users from focusing on the important menu items.



XAML

```
<MenuFlyout>
  <MenuFlyoutItem Text="Share" >
    <MenuFlyoutItem.Icon>
      <FontIcon Glyph="&#xE72D;" />
    </MenuFlyoutItem.Icon>
  </MenuFlyoutItem>
  <MenuFlyoutItem Text="Copy" Icon="Copy" />
  <MenuFlyoutItem Text="Delete" Icon="Delete" />
  <MenuFlyoutSeparator />
  <MenuFlyoutItem Text="Rename" />
  <MenuFlyoutItem Text="Select" />
</MenuFlyout>
```

Tip

The size of the icon in a MenuFlyoutItem is 16x16px. If you use SymbolIcon, FontIcon, or PathIcon, the icon automatically scales to the correct size with no loss of fidelity. If you use BitmapIcon, ensure that your asset is 16x16px.

Light dismiss

Light dismiss controls such as menus, context menus, and other flyouts, trap keyboard and gamepad focus inside the transient UI until dismissed. To provide a visual cue for

this behavior, light dismiss controls on Xbox will draw an overlay that dims the visibility of out of scope UI. This behavior can be modified with the LightDismissOverlayMode property. By default, transient UIs will draw the light dismiss overlay on Xbox (**Auto**) but not other device families. You can choose to force the overlay to be always **On** or always **Off**.

XAML

```
<MenuFlyout LightDismissOverlayMode="Off" />
```

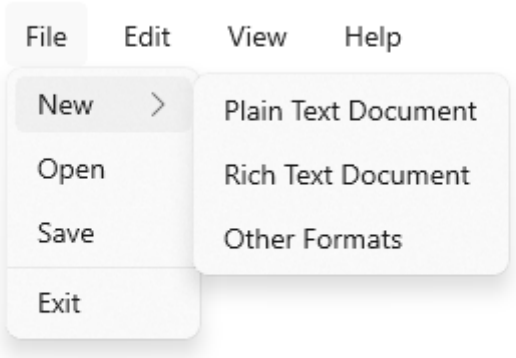
Create a menu bar

✔ **Important APIs:** MenuBar class. MenuBarItem class

Open the WinUI 3 Gallery app and see MenuBar in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

You use the same elements to create menus in a menu bar as in a menu flyout. However, instead of grouping MenuFlyoutItem objects in a MenuFlyout, you group them in a MenuBarItem element. Each MenuBarItem is added to the MenuBar as a top level menu.



> ⓘ **Note**
>
> This example shows only how to create the UI structure, but does not show implementation of any of the commands.

XAML

```
<muxc:MenuBar>
    <muxc:MenuBarItem Title="File">
        <MenuFlyoutSubItem Text="New">
            <MenuFlyoutItem Text="Plain Text Document"/>
            <MenuFlyoutItem Text="Rich Text Document"/>
            <MenuFlyoutItem Text="Other Formats..."/>
        </MenuFlyoutSubItem>
        <MenuFlyoutItem Text="Open..."/>
        <MenuFlyoutItem Text="Save"/>
        <MenuFlyoutSeparator />
        <MenuFlyoutItem Text="Exit"/>
    </muxc:MenuBarItem>

    <muxc:MenuBarItem Title="Edit">
        <MenuFlyoutItem Text="Undo"/>
        <MenuFlyoutItem Text="Cut"/>
        <MenuFlyoutItem Text="Copy"/>
        <MenuFlyoutItem Text="Paste"/>
    </muxc:MenuBarItem>

    <muxc:MenuBarItem Title="View">
        <MenuFlyoutItem Text="Output"/>
        <MenuFlyoutSeparator/>
        <muxc:RadioMenuFlyoutItem Text="Landscape"
GroupName="OrientationGroup"/>
        <muxc:RadioMenuFlyoutItem Text="Portrait"
GroupName="OrientationGroup" IsChecked="True"/>
        <MenuFlyoutSeparator/>
        <muxc:RadioMenuFlyoutItem Text="Small icons" GroupName="SizeGroup"/>
        <muxc:RadioMenuFlyoutItem Text="Medium icons" IsChecked="True"
GroupName="SizeGroup"/>
        <muxc:RadioMenuFlyoutItem Text="Large icons" GroupName="SizeGroup"/>
    </muxc:MenuBarItem>

    <muxc:MenuBarItem Title="Help">
        <MenuFlyoutItem Text="About"/>
    </muxc:MenuBarItem>
</muxc:MenuBar>
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

MenuBar requires Windows 10, version 1809 (SDK 17763) or later, or WinUI 2.

The MenuFlyout and MenuBar controls for UWP apps are included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for these controls exist in both the Windows.UI.Xaml.Controls and Microsoft.UI.Xaml.Controls namespaces.

- ✓ **UWP APIs:** MenuFlyout class, MenuBar class, ContextFlyout property, FlyoutBase.AttachedFlyout property
- ✓ **WinUI 2 Apis:** MenuBar class, ContextFlyout property, FlyoutBase.AttachedFlyout property
- ✓ Open the WinUI 2 Gallery app and see MenuBar in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for these controls that uses rounded corners. For more info, see Corner radius.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:MenuFlyout />
<muxc:MenuBar />
```

Related articles

- Command design basics for Windows apps
- Menus and context menus
- Contextual commanding for collections and lists
- MenuFlyout class
- MenuBar class

Breadcrumb Bar

Article • 02/26/2025

A BreadcrumbBar provides the direct path of pages or folders to the current location. It is often used for situations where the user's navigation trail (in a file system or menu system) needs to be persistently visible and the user may need to go back to a previous location.

Is this the right control?

A breadcrumb bar lets a user keep track of their location when navigating through an app or folders, and lets them quickly jump back to a previous location in the path.

Use a BreadcrumbBar when the path taken to the current position is relevant. This UI is commonly used in folder managers and when a user may navigate many levels deep into an app.

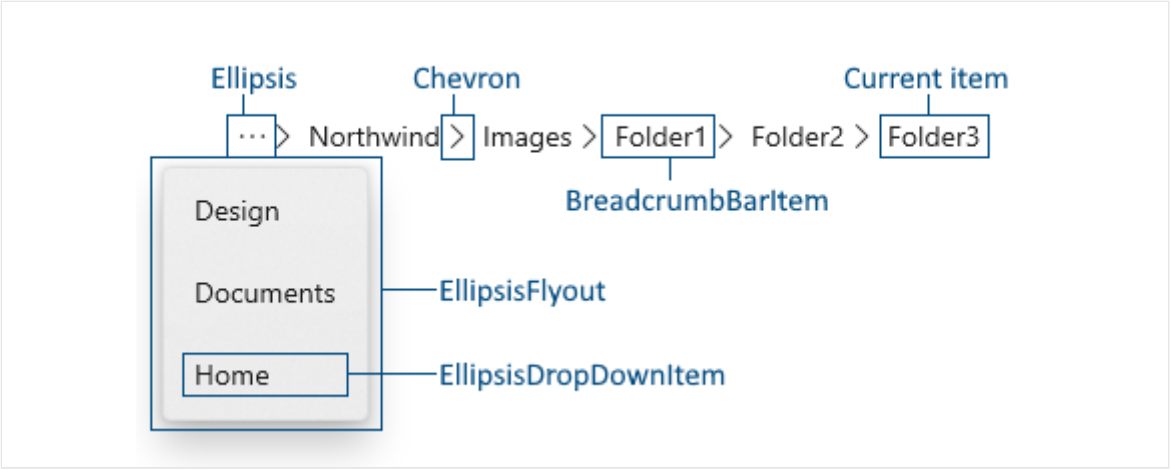
Breadcrumb bar UI

The breadcrumb bar displays each node in a horizontal line, separated by chevrons.

If the app is resized so that there is not enough space to show all the nodes, the breadcrumbs collapse and an ellipsis replaces the leftmost nodes. Clicking the ellipsis opens a flyout to show the collapsed nodes.

Anatomy

The image below shows the parts of the `BreadcrumbBar` control. You can modify the appearance of some parts by using lightweight styling.



Recommendations

- Use a breadcrumb bar when you have many levels of navigation and expect users to be able to return to any previous level.
- Don't use a breadcrumb bar if you only have 2 possible levels of navigation. Simple back navigation is sufficient.
- Show the current location as the last item in the breadcrumb bar. However, you typically don't want to perform any navigation if the user clicks the current item. (If you want to let the user reload the current page or data, consider providing a dedicated 'reload' option.)

Create a breadcrumb bar

✓ **Important APIs:** BreadcrumbBar class

Open the WinUI 3 Gallery app and see the BreadcrumbBar in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This example shows how to create a breadcrumb bar with the default styling. You can place the breadcrumb bar anywhere in your app UI. You populate the breadcrumbs by setting the ItemsSource property. Here, it's set to an array of strings that are shown in the breadcrumb bar.

XAML

```
<BreadcrumbBar x:Name="BreadcrumbBar1"/>
```

C#

```
BreadcrumbBar1.ItemsSource =
   new string[] { "Home", "Documents", "Design", "Northwind", "Images",
"Folder1", "Folder2", "Folder3" };
```

ItemsSource

The breadcrumb bar does not have an `Items` property, it only has an ItemsSource property. This means you can't populate the breadcrumbs in XAML or by adding them directly to an `Items` collection in code. Instead, you create a collection and connect the `ItemsSource` property to it in code or using data binding.

You can set the ItemsSource to a collection of any type of data to suit the needs of your app. The data items in the collection are used both to display the breadcrumb in the bar, and to navigate when an item in the breadcrumb bar is clicked. In the examples on this page, we create a simple `struct` (named `Crumb`) that contains a label to display in the breadcrumb bar and a data object that holds information used for navigation.

C#

```
public readonly struct Crumb
{
    public Crumb(String label, object data)
    {
        Label = label;
        Data = data;
    }
    public string Label { get; }
    public object Data { get; }
    public override string ToString() => Label;
}
```

ItemTemplate

By default, the breadcrumb bar displays the string representation of each item in the collection. If the data items in your collection don't have an appropriate `ToString` override, you can use the ItemTemplate property to specify a data template that defines how the items are shown in the breadcrumb bar.

For example, if your breadcrumb collection was a list of StorageFolder objects, you could provide a data template and bind to the DisplayName property like this.

C#

```
ObservableCollection<StorageFolder> Breadcrumbs =
    new ObservableCollection<StorageFolder>();
```

XAML

```
<BreadcrumbBar x:Name="FolderBreadcrumbBar"
               ItemsSource="{x:Bind Breadcrumbs}">
    <BreadcrumbBar.ItemTemplate>
        <DataTemplate x:DataType="StorageFolder">
            <TextBlock Text="{x:Bind DisplayName}"/>
        </DataTemplate>
    </BreadcrumbBar.ItemTemplate>
</BreadcrumbBar>
```

ItemClicked

Handle the ItemClicked event to navigate to the item the user has clicked in the breadcrumb bar. The current location is typically shown as the last item in the breadcrumb bar, so you should include a check in your event handler if you don't want to reload the current location.

This example checks the Index to see whether the clicked Item is the last item in the collection, which is the current location. If it is, no navigation occurs.

C#

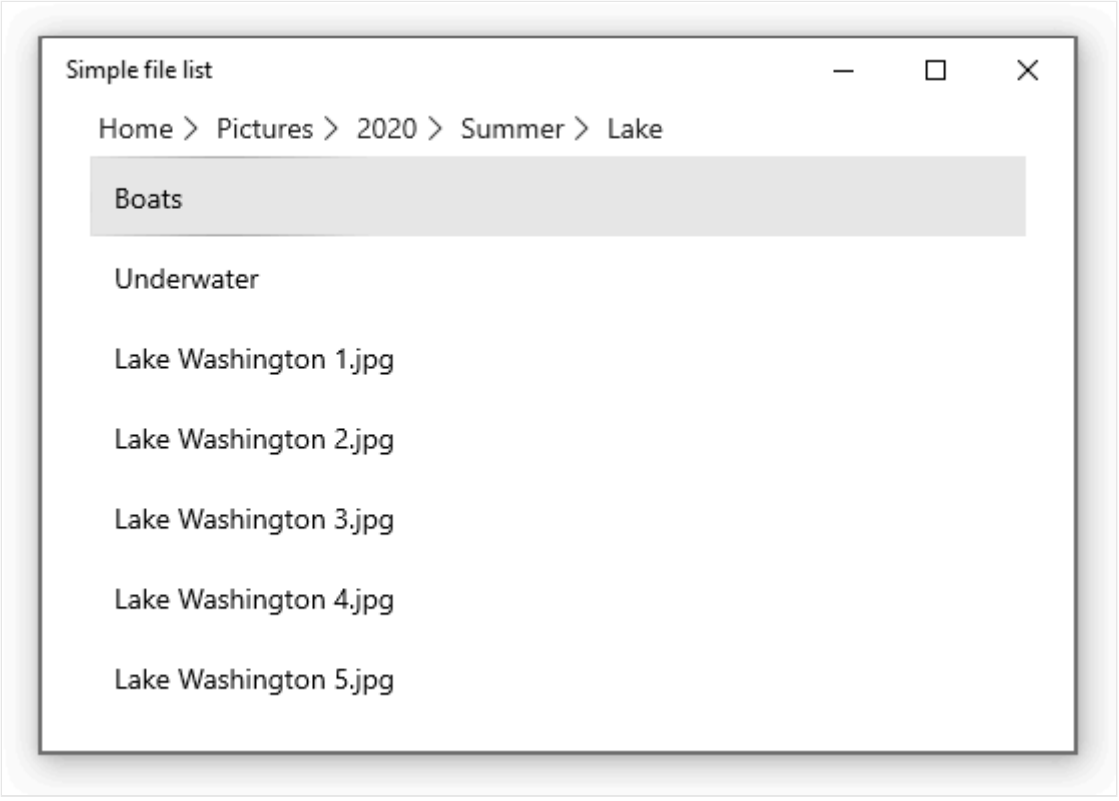
```
// Breadcrumbs is set as BreadcrumbBar1.ItemsSource.
List<Crumb> Breadcrumbs = new List<Crumb>();

...

private void BreadcrumbBar1_ItemClicked(muxc.BreadcrumbBar sender,
muxc.BreadcrumbBarItemClickedEventArgs args)
{
    if (args.Index < Breadcrumbs.Count - 1)
    {
        var crumb = (Crumb)args.Item;
        Frame.Navigate((Type)crumb.Data);
    }
}
```

Lightweight styling

You can modify the default Style and ControlTemplate to give the control a unique appearance. See the Control Style and Template section of the BreadcrumbBar API docs for a list of the available theme resources. For more info, see the Light-weight styling section of the Styling controls article.

Code examples

This example shows how you could use a breadcrumb bar in a simple file explorer scenario. The list view shows the contents of the selected Pictures or Music library, and lets the user drill down into sub-folders. The breadcrumb bar is placed in the header of the list view, and shows the path to the current folder.



XAML

```
<Grid>
   <ListView x:Name="FolderListView" Margin="24,0"
             IsItemClickEnabled="True"
             ItemClick="FolderListView_ItemClick">
      <ListView.Header>
         <BreadcrumbBar x:Name="FolderBreadcrumbBar"
                        ItemsSource="{x:Bind Breadcrumbs}"
                        ItemClicked="FolderBreadcrumbBar_ItemClicked">
         </BreadcrumbBar>
      </ListView.Header>
      <ListView.ItemTemplate>
```

```
          <DataTemplate>
             <TextBlock Text="{Binding Name}"/>
             </DataTemplate>
       </ListView.ItemTemplate>
    </ListView>
</Grid>
```

C#

```
public sealed partial class MainPage : Page
{
    List<IStorageItem> Items;
    ObservableCollection<object> Breadcrumbs =
        new ObservableCollection<object>();

    public MainPage()
    {
        this.InitializeComponent();
        InitializeView();
    }

    private void InitializeView()
    {
        // Start with Pictures and Music libraries.
        Items = new List<IStorageItem>();
        Items.Add(KnownFolders.PicturesLibrary);
        Items.Add(KnownFolders.MusicLibrary);
        FolderListView.ItemsSource = Items;

        Breadcrumbs.Clear();
        Breadcrumbs.Add(new Crumb("Home", null));
    }

    private async void FolderBreadcrumbBar_ItemClicked(muxc.BreadcrumbBar
sender, muxc.BreadcrumbBarItemClickedEventArgs args)
    {
        // Don't process last index (current location)
        if (args.Index < Breadcrumbs.Count - 1)
        {
            // Home is special case.
            if (args.Index == 0)
            {
                InitializeView();
            }
            // Go back to the clicked item.
            else
            {
                var crumb = (Crumb)args.Item;
                await GetFolderItems((StorageFolder)crumb.Data);

                // Remove breadcrumbs at the end until
                // you get to the one that was clicked.
                while (Breadcrumbs.Count > args.Index + 1)
```

```
                {
                    Breadcrumbs.RemoveAt(Breadcrumbs.Count - 1);
                }
            }
        }
    }

    private async void FolderListView_ItemClick(object sender,
ItemClickEventArgs e)
    {
        // Ignore if a file is clicked.
        // If a folder is clicked, drill down into it.
        if (e.ClickedItem is StorageFolder)
        {
            StorageFolder folder = e.ClickedItem as StorageFolder;
            await GetFolderItems(folder);
            Breadcrumbs.Add(new Crumb(folder.DisplayName, folder));
        }
    }

    private async Task GetFolderItems(StorageFolder folder)
    {
        IReadOnlyList<IStorageItem> itemsList = await
folder.GetItemsAsync();
        FolderListView.ItemsSource = itemsList;
    }
}

public readonly struct Crumb
{
    public Crumb(String label, object data)
    {
        Label = label;
        Data = data;
    }
    public string Label { get; }
    public object Data { get; }
    public override string ToString() => Label;
}
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

The BreadcrumbBar for UWP apps requires WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in the Microsoft.UI.Xaml.Controls namespace.

- ✓ **WinUI 2 Apis:** BreadcrumbBar class
- ✓ Open the WinUI 2 Gallery app and see the BreadcrumbBar in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:BreadcrumbBar />
```

Related articles

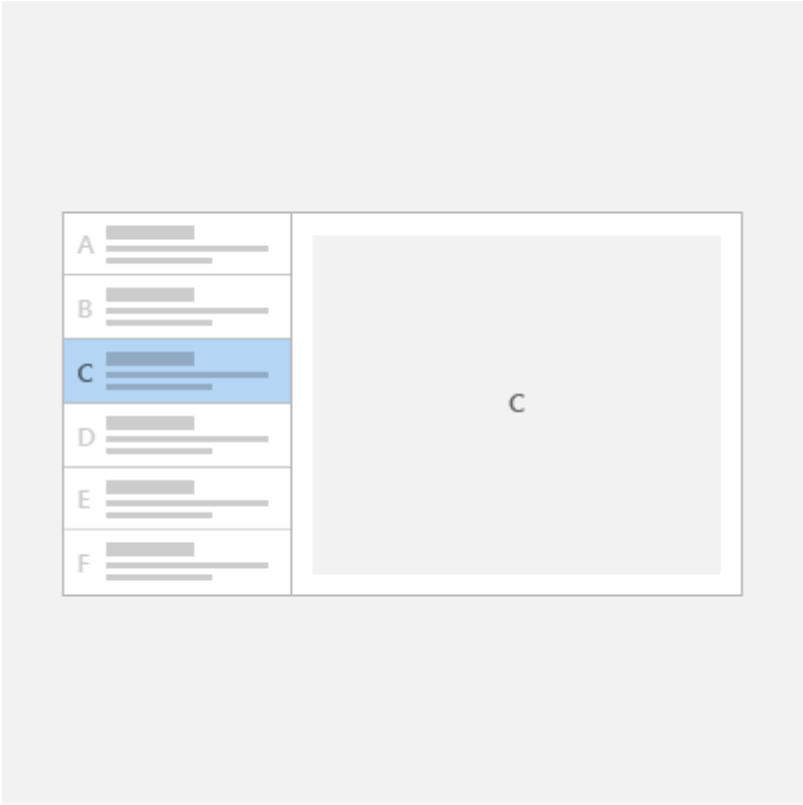
- Navigation design basics
- NavigationView
- BreadcrumbBar class

List/details pattern

Article • 06/09/2022

The list/details pattern has a list pane (usually with a list view) and a details pane for content. When an item in the list is selected, the details pane is updated. This pattern is frequently used for email and address books.

> **Important APIs**: ListView class, SplitView class



Tip

If you'd like to use a XAML control that implements this pattern for you, we recommend the **ListDetailsView XAML Control** from the Windows Community Toolkit.

Is this the right pattern?

The list/details pattern works well if you want to:

- Build an email app, address book, or any app that is based on a list-details layout.
- Locate and prioritize a large collection of content.

- Allow the quick addition and removal of items from a list while working back-and-forth between contexts.

Choose the right style

When implementing the list/details pattern, we recommend that you use either the stacked style or the side-by-side style, based on the amount of available screen space.

Available window width	Recommended style
320 epx-640 epx	Stacked
641 epx or wider	Side-by-side

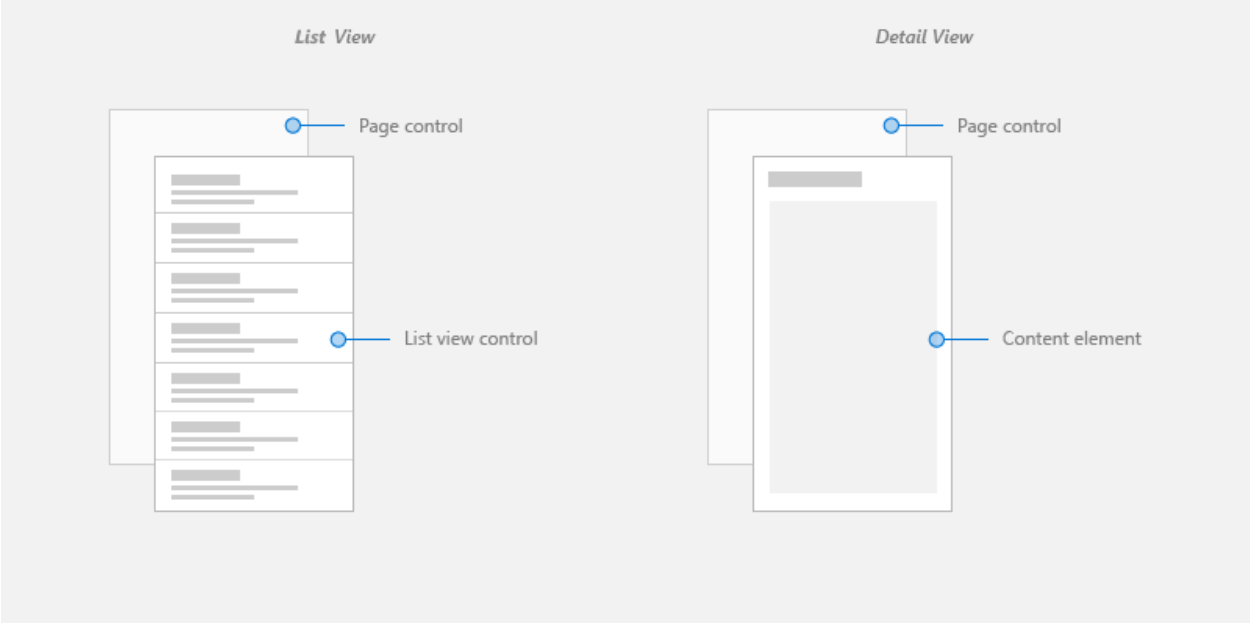
Stacked style

In the stacked style, only one pane is visible at a time: the list or the details.

The user starts at the list pane and "drills down" to the details pane by selecting an item in the list. To the user, it appears as though the list and details views exist on two separate pages.

Create a stacked list/details pattern

One way to create the stacked list/details pattern is to use separate pages for the list pane and the details pane. Place the list view on one page, and the details pane on a separate page.



For the list view page, a list view control works well for presenting lists that can contain images and text.

For the details view page, use the content element that makes the most sense. If you have a lot of separate fields, consider using a **Grid** layout to arrange elements into a form.

For navigation between pages, see navigation history and backwards navigation for Windows apps.

Side-by-side style

In the side-by-side style, the list pane and details pane are visible at the same time.

The list in the list pane has a selection visual to indicate the currently selected item. Selecting a new item in the list updates the details pane.

Create a side-by-side list/details pattern

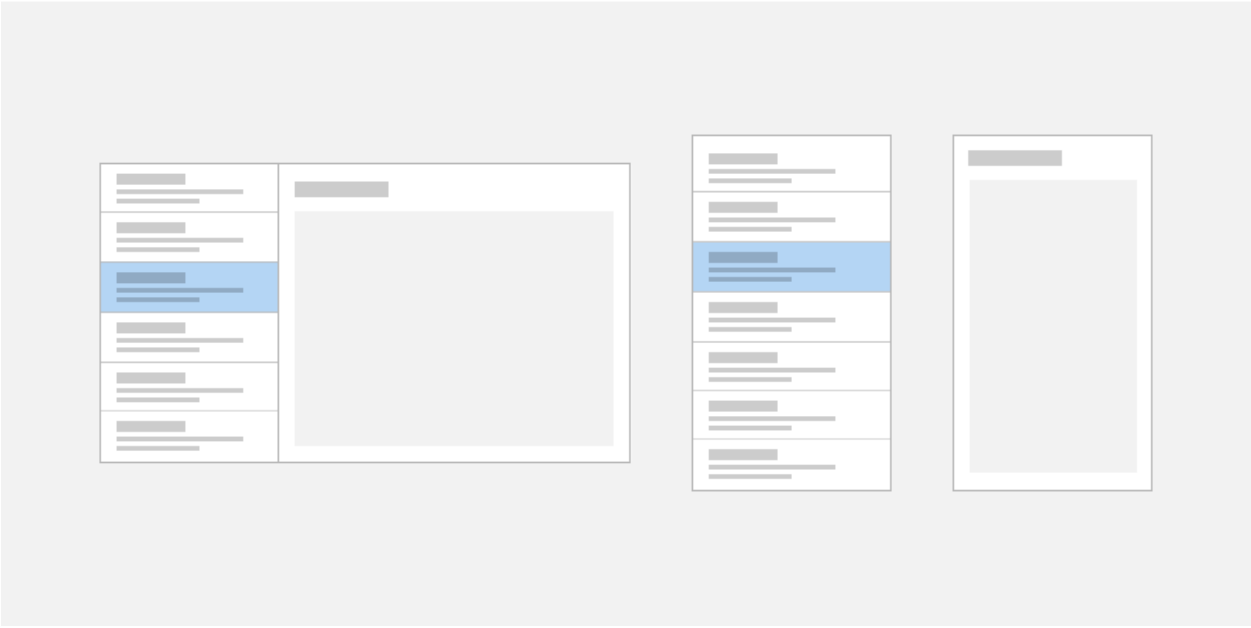
One way to create a side-by-side list/details pattern is to use the split view control. Place the list view in the split view pane, and the details view in the split view content.

For the list pane, a list view control works well for presenting lists that can contain images and text.

For the details content, use the content element that makes the most sense. If you have a lot of separate fields, consider using a **Grid** layout to arrange elements into a form.

Adaptive layout

To implement a list/details pattern for any screen size, create a responsive UI with an adaptive layout.



Create an adaptive list/details pattern

To create an adaptive layout, define different **VisualStates** for your UI, and declare breakpoints for the different states with **AdaptiveTriggers**.

Get the sample code

The following samples implement the list/details pattern with adaptive layouts and demonstrate data binding to static, database, and online resources:

- Master/detail sample
- ListView and GridView sample
- Customer orders database sample
- RSS Reader sample

> **Tip**
>
> If you'd like to use a XAML control that implements this pattern for you, we recommend the **ListDetailsView XAML Control** from the Windows Community

Toolkit.

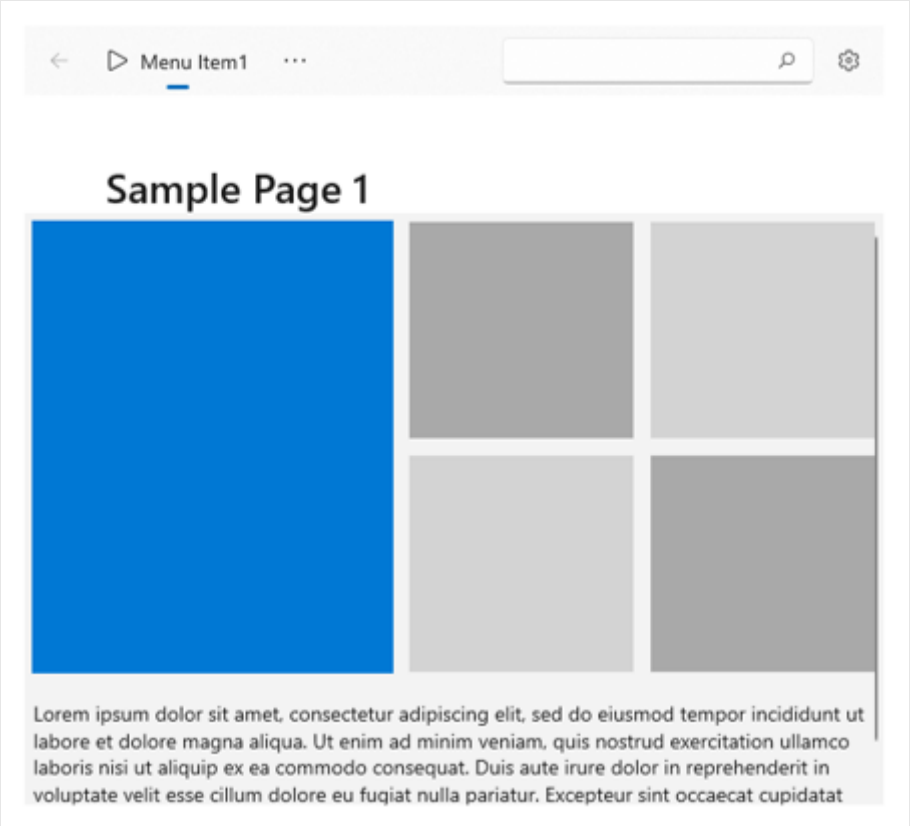
Related articles

- Lists
- App and command bars
- ListView class
- SplitView class

NavigationView

Article • 02/26/2025

The NavigationView control provides top-level navigation for your app. It adapts to a variety of screen sizes and supports both *top* and *left* navigation styles.



NavigationView supports both top and left navigation pane or menu

Is this the right control?

NavigationView is an adaptive navigation control that works well for:

- Providing a consistent navigational experience throughout your app.
- Preserving screen real estate on smaller windows.
- Organizing access to many navigation categories.

For other navigation patterns, see Navigation design basics.

Create a navigation view

✔ **Important APIs:** NavigationView class

Open the WinUI 3 Gallery app and see the NavigationView in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This example shows how to create a simple navigation view in XAML.

XAML

```
<NavigationView>
    <NavigationView.MenuItems>
        <NavigationViewItem Content="Nav Item A"/>
        <NavigationViewItem Content="Nav Item B"/>
        <NavigationViewItem Content="Nav Item C"/>
    </NavigationView.MenuItems>

    <Frame x:Name="ContentFrame"/>
</NavigationView>
```

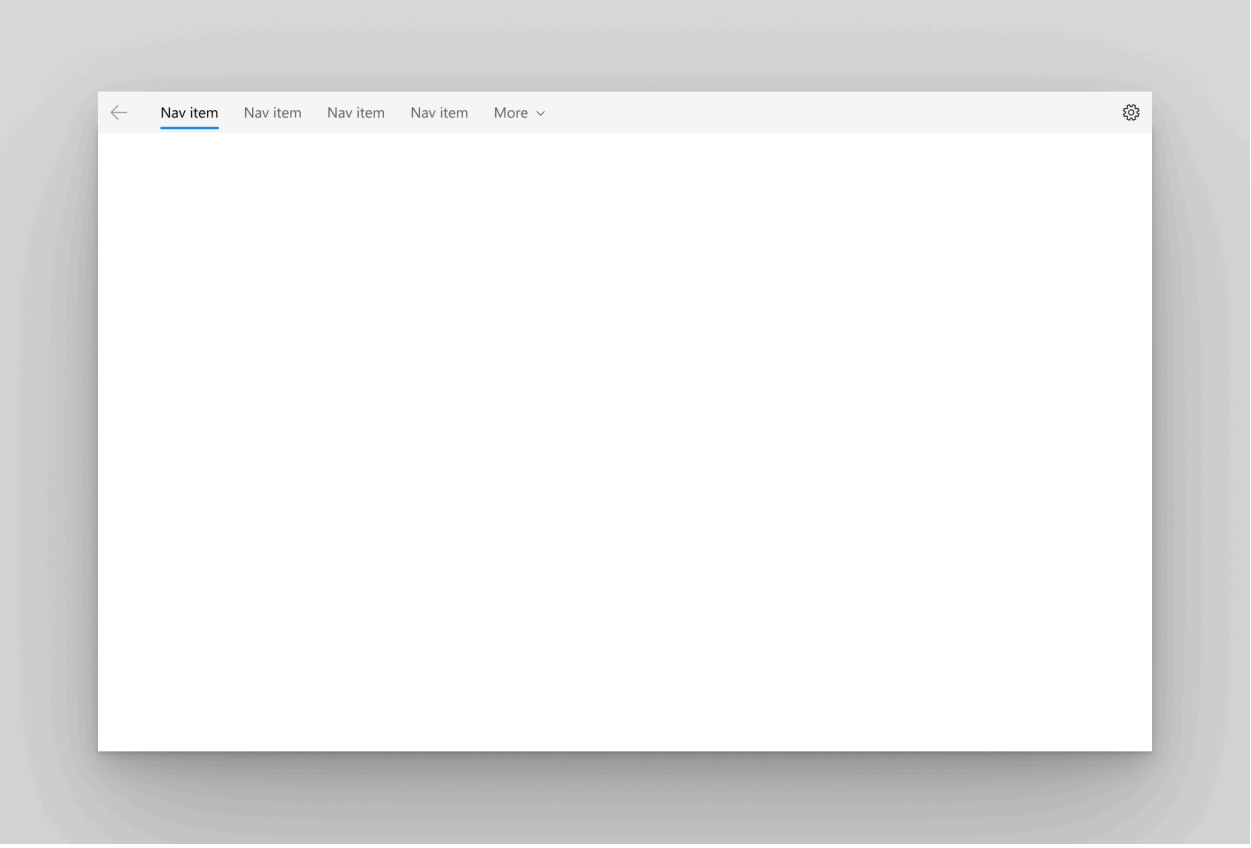
Display modes

You can use the PaneDisplayMode property to configure different navigation styles, or display modes, for the NavigationView.

Top

The pane is positioned above the content.

`PaneDisplayMode="Top"`



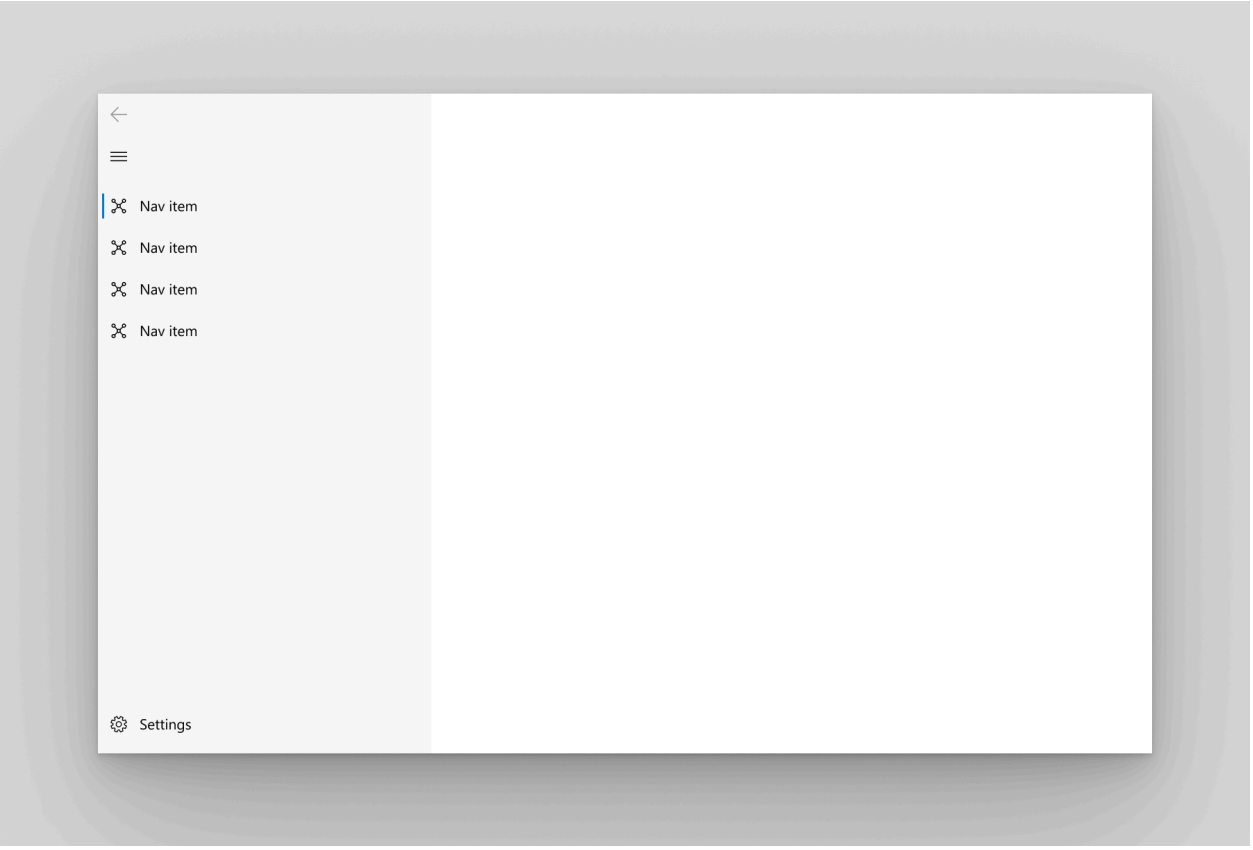
We recommend *top* navigation when:

- You have 5 or fewer top-level navigation categories that are equally important, and any additional top-level navigation categories that end up in the dropdown overflow menu are considered less important.
- You need to show all navigation options on screen.
- You want more space for your app content.
- Icons cannot clearly describe your app's navigation categories.

Left

The pane is expanded and positioned to the left of the content.

`PaneDisplayMode="Left"`



We recommend *left* navigation when:

- You have 5-10 equally important top-level navigation categories.
- You want navigation categories to be very prominent, with less space for other app content.

LeftCompact

The pane shows only icons until opened and is positioned to the left of the content. When opened, the pane overlays the content.
`PaneDisplayMode="LeftCompact"`

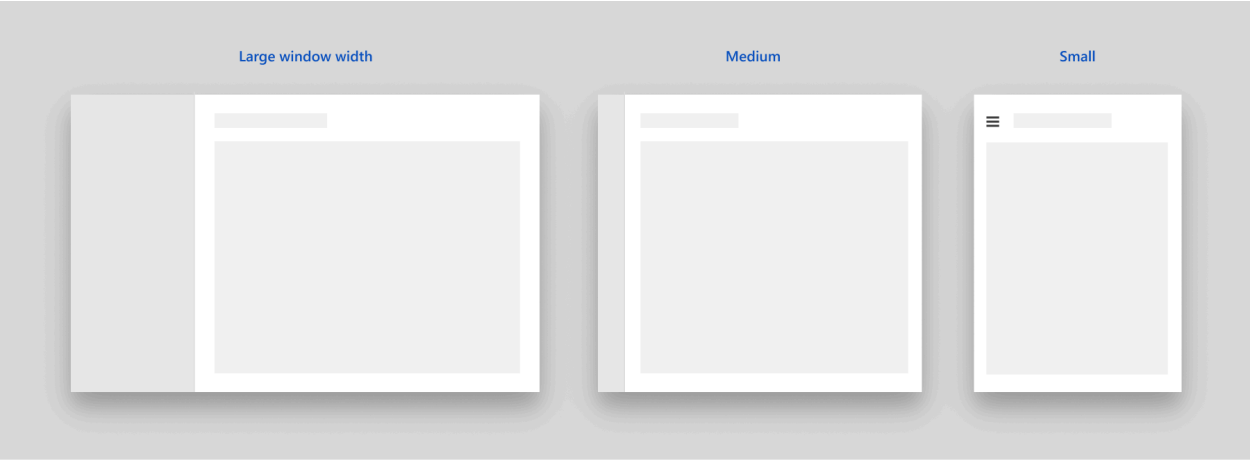
LeftMinimal

Only the menu button is shown until the pane is opened. When opened, the pane overlays the left side of the content.

`PaneDisplayMode="LeftMinimal"`

Auto

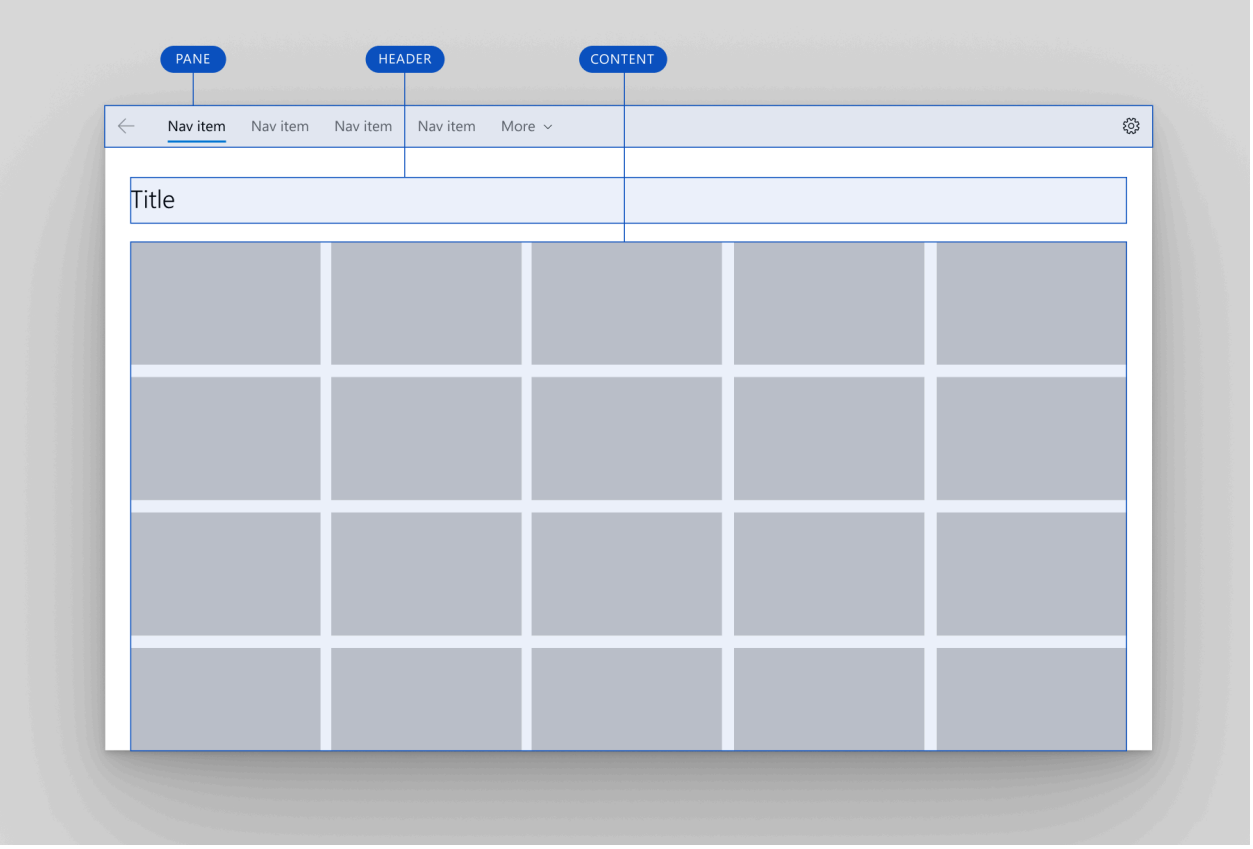
By default, PaneDisplayMode is set to `Auto`. In `Auto` mode, the NavigationView adapts between `LeftMinimal` when the window is narrow, to `LeftCompact`, and then `Left` as the window gets wider. For more info, see the adaptive behavior section.



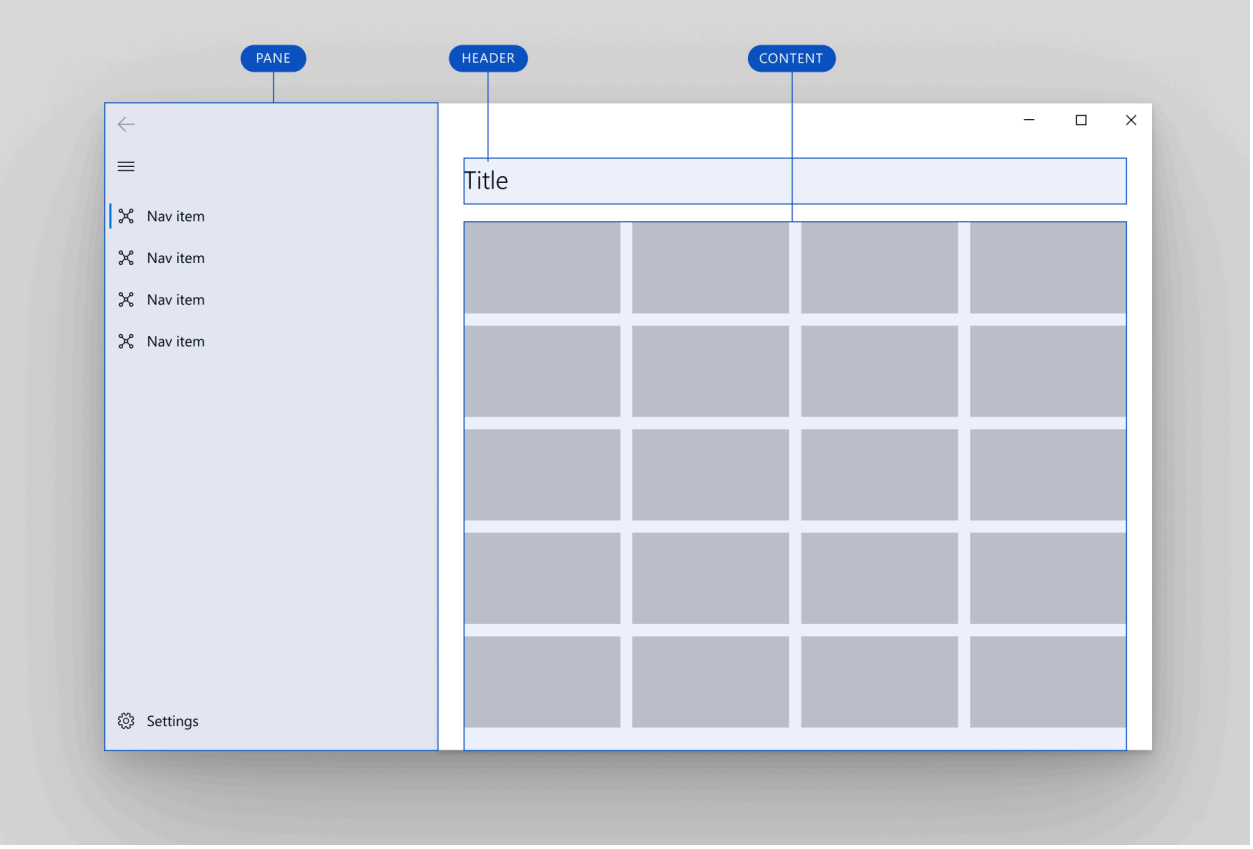
NavigationView default adaptive behavior

Anatomy

These images show the layout of the pane, header, and content areas of the control when configured for *top* or *left* navigation.



Top navigation layout



Left navigation layout

Pane

You can use the PaneDisplayMode property to position the pane above the content or to the left of the content.

The NavigationView pane can contain:

- NavigationViewItem objects. Navigation items for navigating to specific pages.
- NavigationViewItemSeparator objects. Separators for grouping navigation items. Set the Opacity property to 0 to render the separator as space.
- NavigationViewItemHeader objects. Headers for labeling groups of items.
- An optional AutoSuggestBox control to allow for app-level search. Assign the control to the NavigationView.AutoSuggestBox property.
- An optional entry point for app settings. To hide the settings item, set the IsSettingsVisible property to `false`.

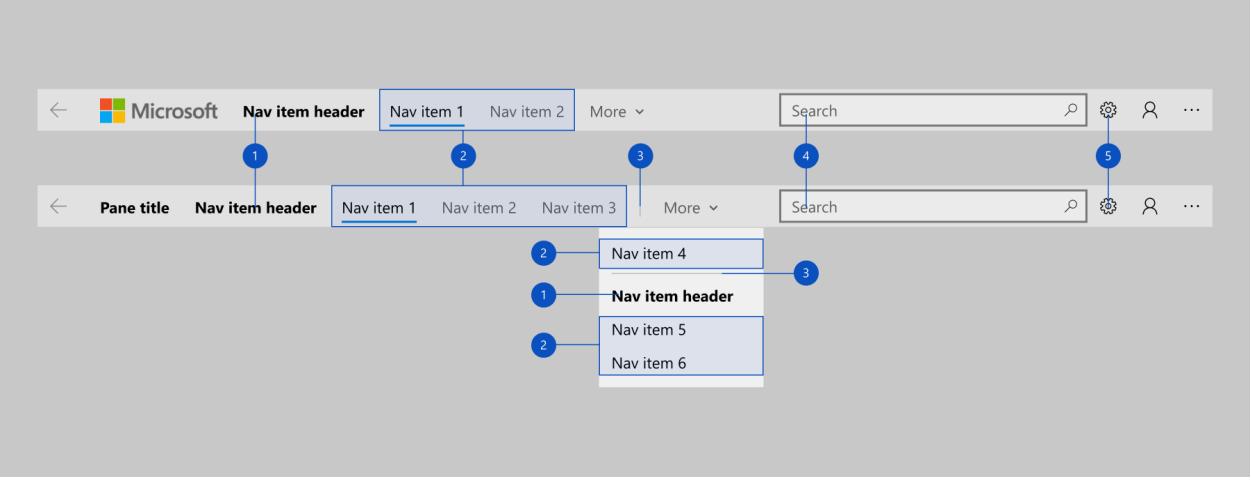
The left pane also contains:

- A menu button to toggle the pane opened and closed. On larger app windows when the pane is open, you may choose to hide this button using the IsPaneToggleButtonVisible property.

The NavigationView has a back button that is placed in the top left-hand corner of the pane. However, it does not automatically handle backwards navigation and add content to the back stack. To enable backwards navigation, see the backwards navigation section.

Here is the detailed pane anatomy for the top and left pane positions.

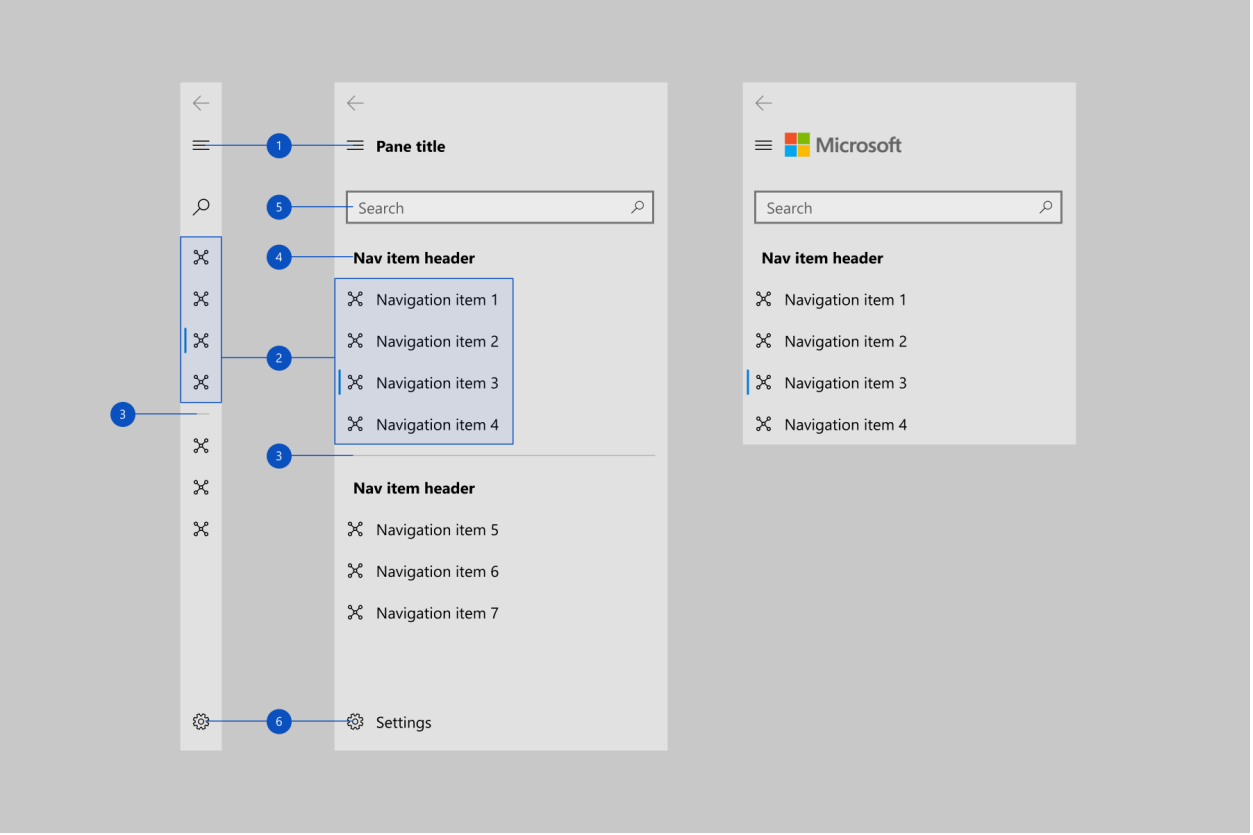
Top navigation pane



1. Headers
2. Navigation items
3. Separators

4. AutoSuggestBox (optional)
5. Settings button (optional)

Left navigation pane



1. Menu button
2. Navigation items
3. Separators
4. Headers
5. AutoSuggestBox (optional)
6. Settings button (optional)

Footer menu items

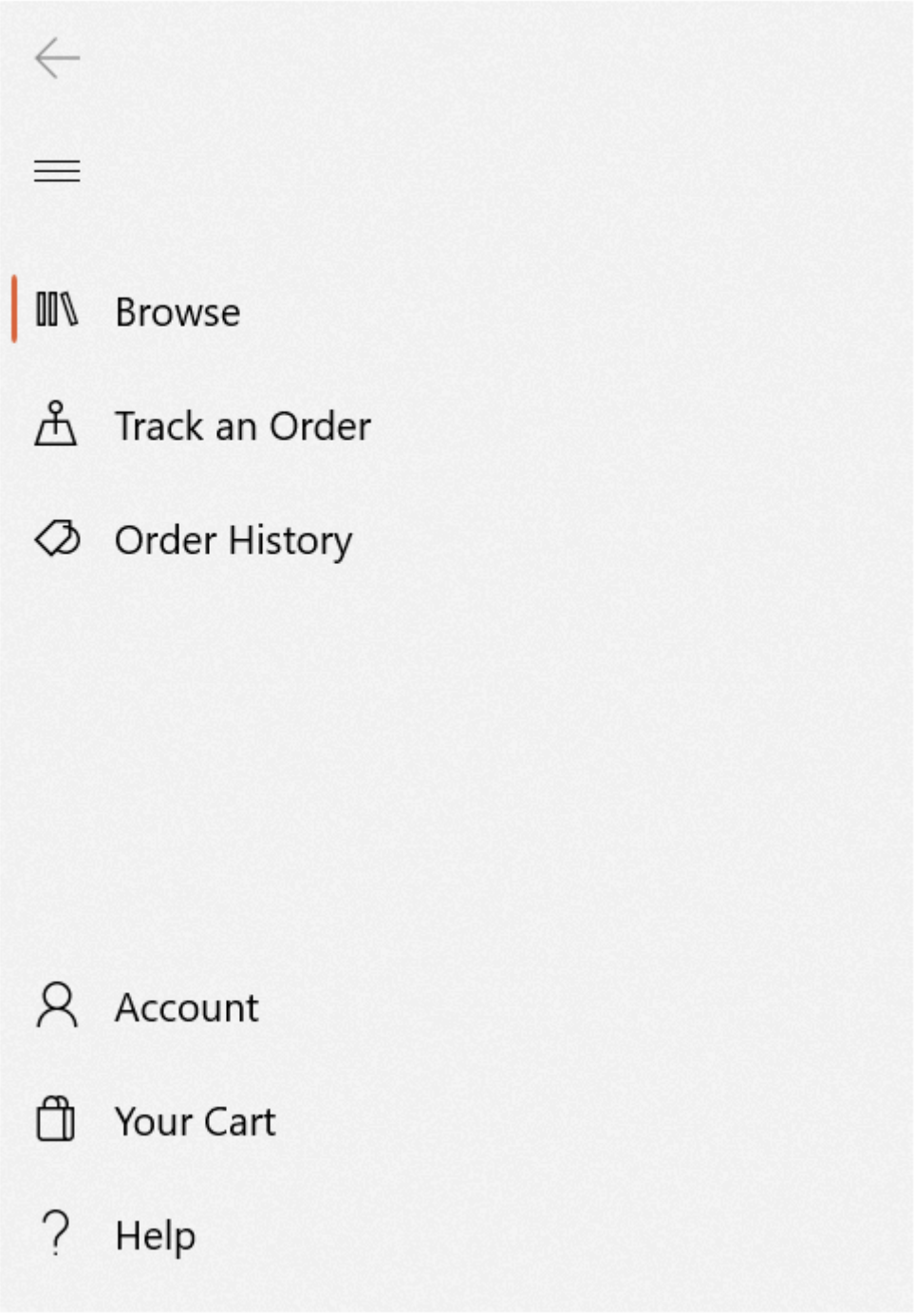
You can use FooterMenuItems to place navigation items at the end of the navigation pane, contrasted with the MenuItems property which places items at the start of the pane.

FooterMenuItems will be displayed before the `Settings` item by default. The `Settings` item can still be toggled using the IsSettingsVisible property.

Only Navigation items should be placed in FooterMenuItems - any other content that needs to align to the footer of the pane should be placed in PaneFooter.

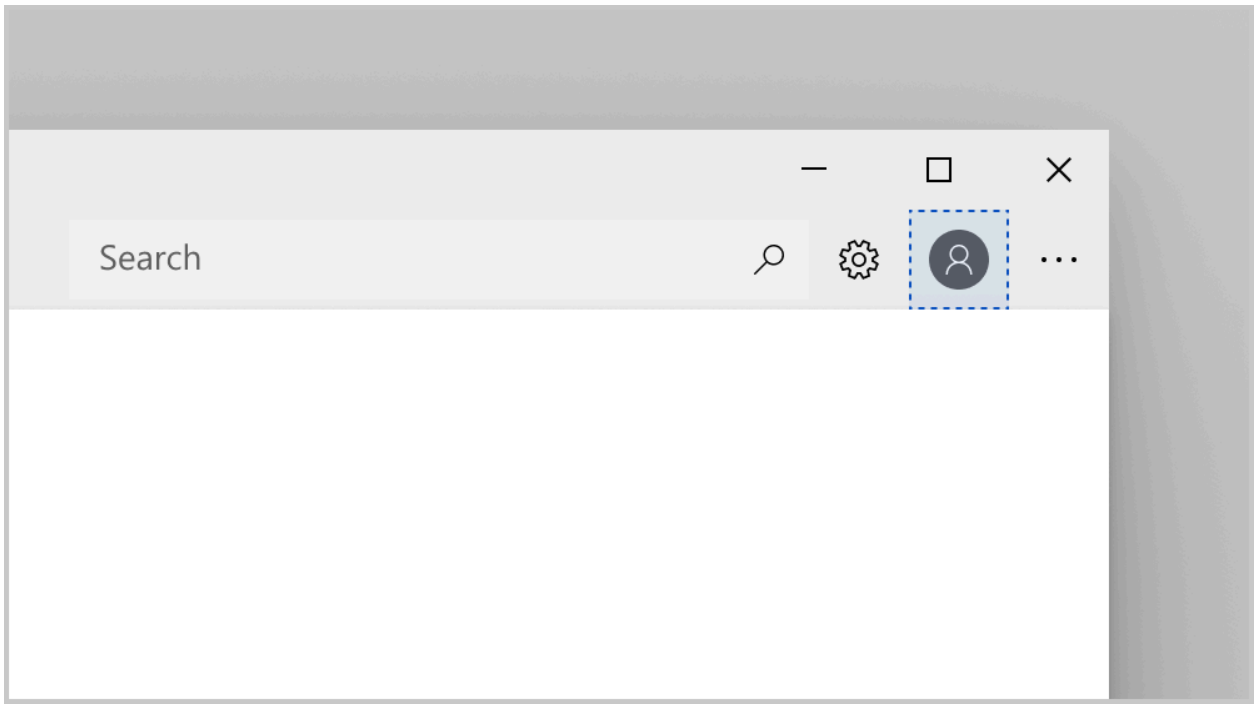
For an example of how to add FooterMenuItems to your NavigationView, see the FooterMenuItems class.

The image below shows a NavigationView with **Account**, **Your Cart**, and **Help** navigation items in the footer menu.

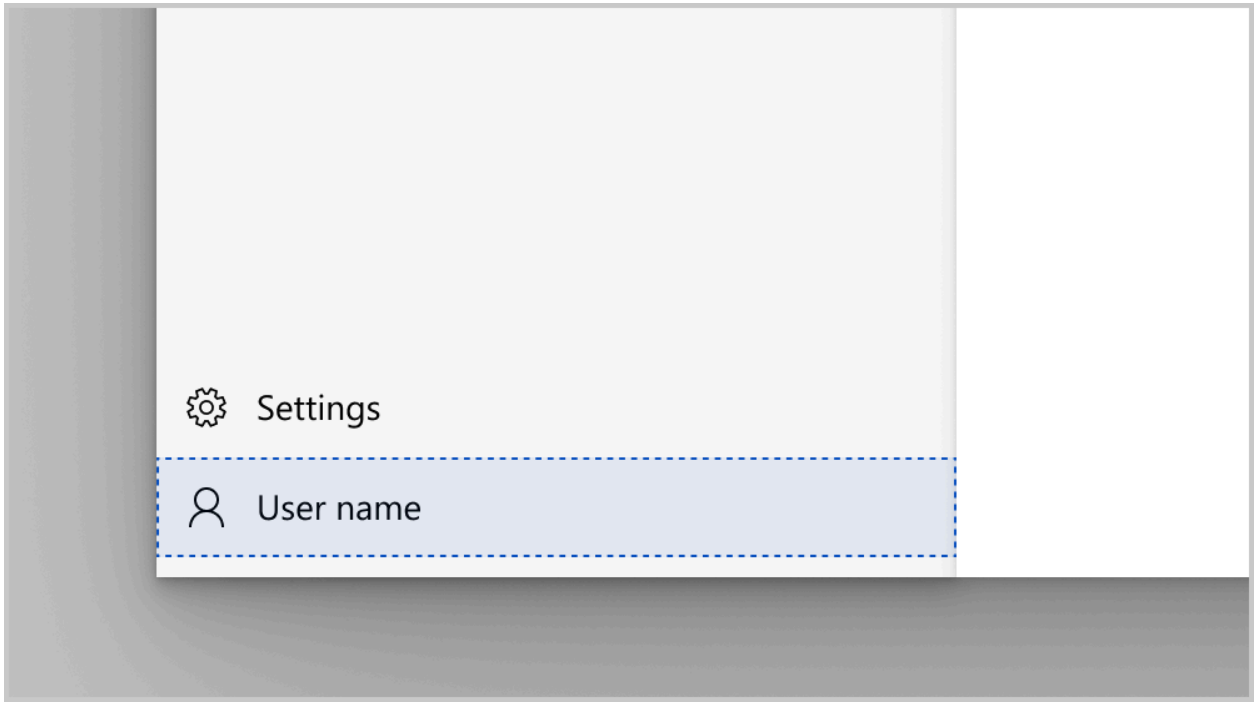


Pane footer

You can place free-form content in the pane's footer by adding it to the PaneFooter property.



Top pane footer



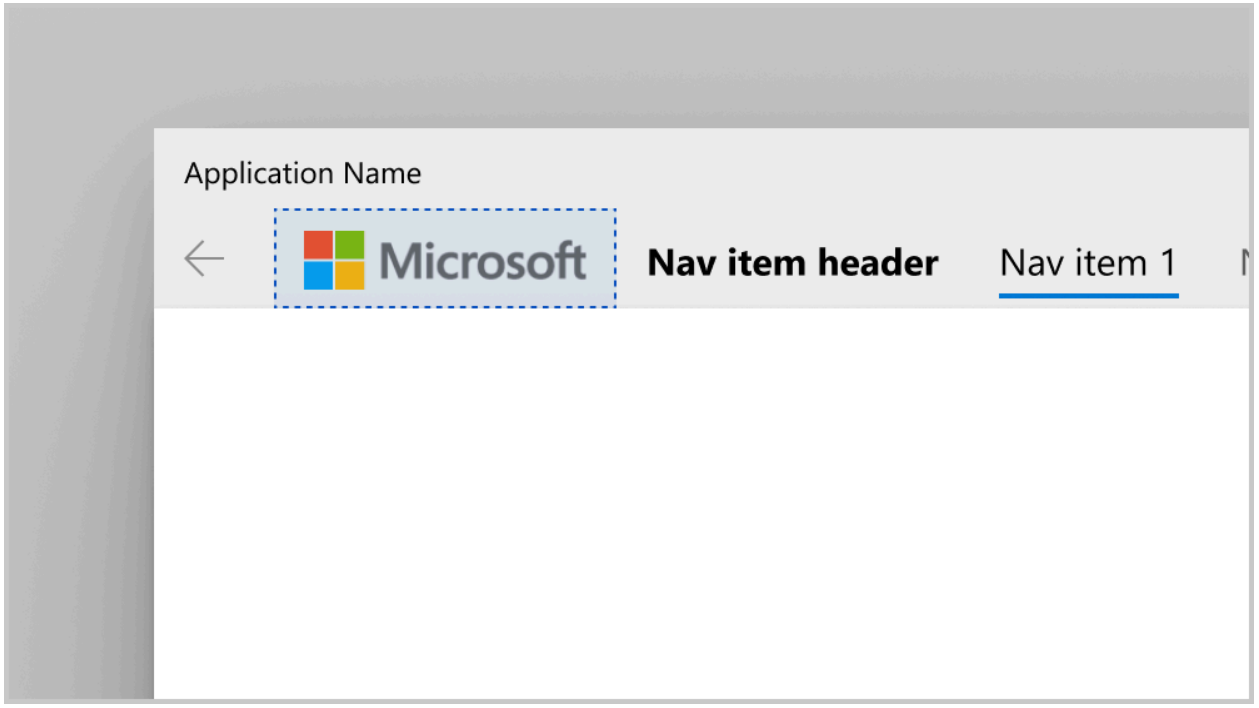
Left pane footer

Pane title and header

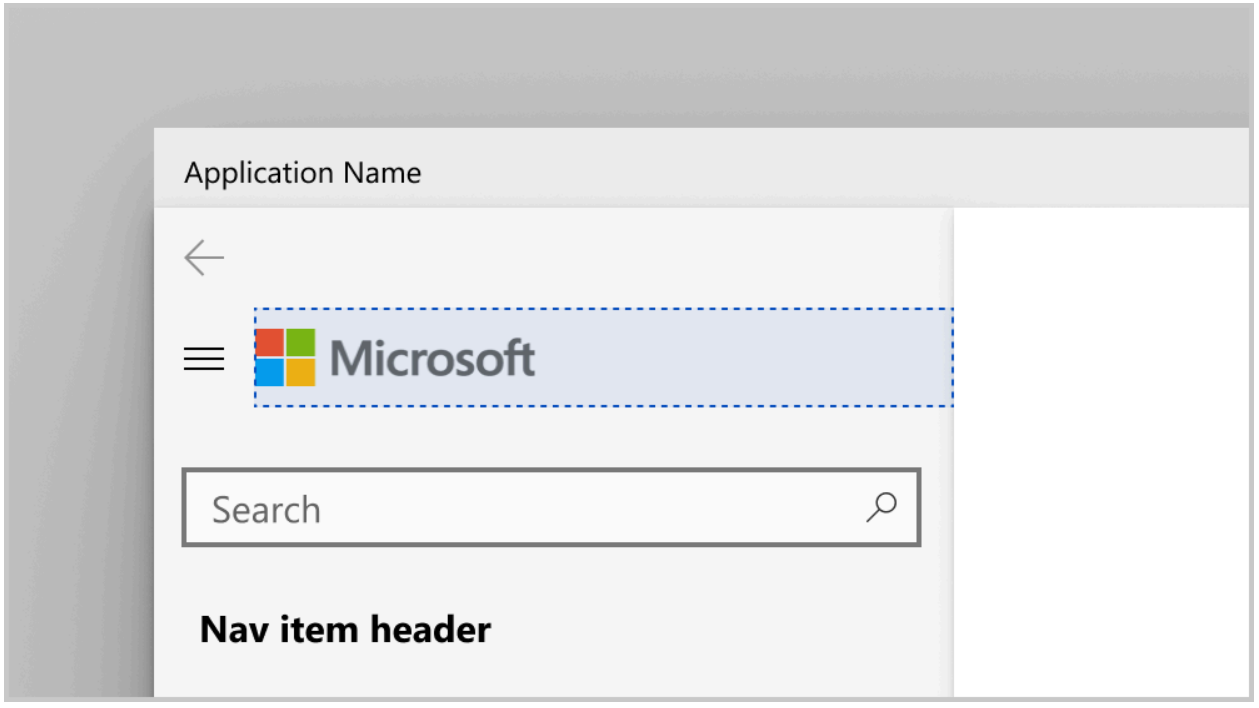
You can place text content in the pane header area by setting the PaneTitle property. It takes a string and shows the text next to the menu button.

To add non-text content, such as an image or logo, you can place any element in the pane's header by adding it to the PaneHeader property.

If both PaneTitle and PaneHeader are set, the content is stacked horizontally next to the menu button, with the PaneTitle closest to the menu button.



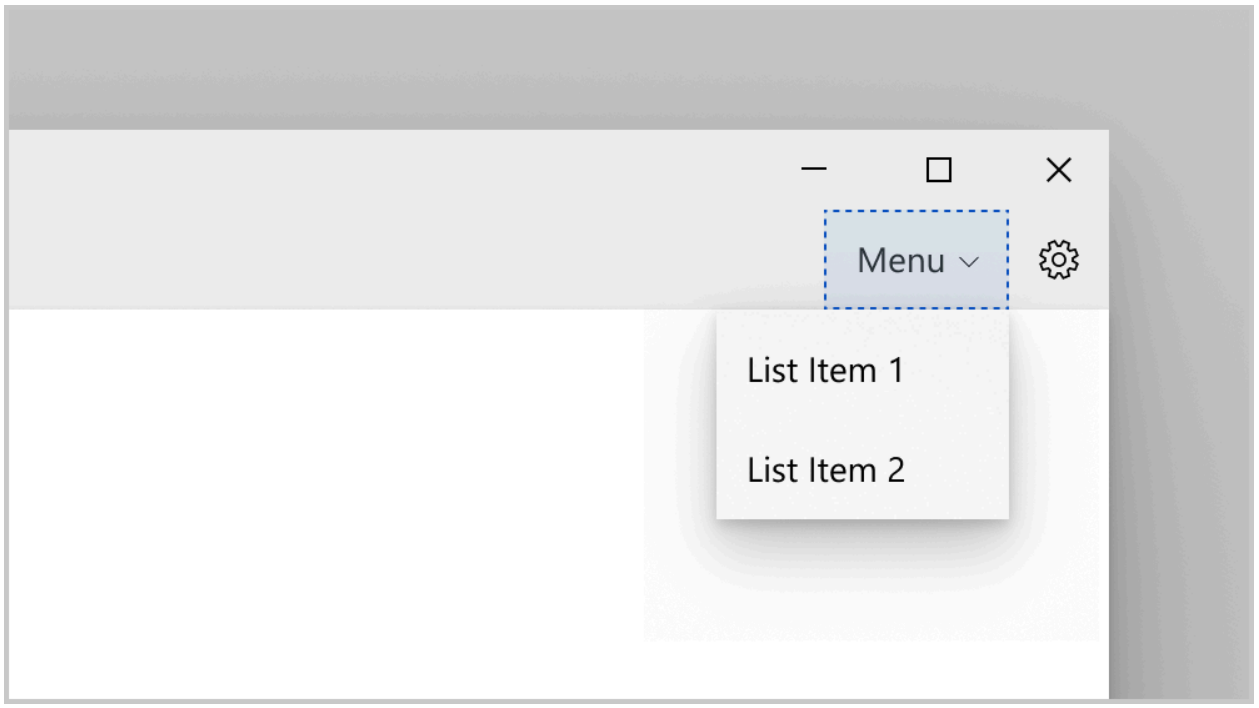
Top pane header



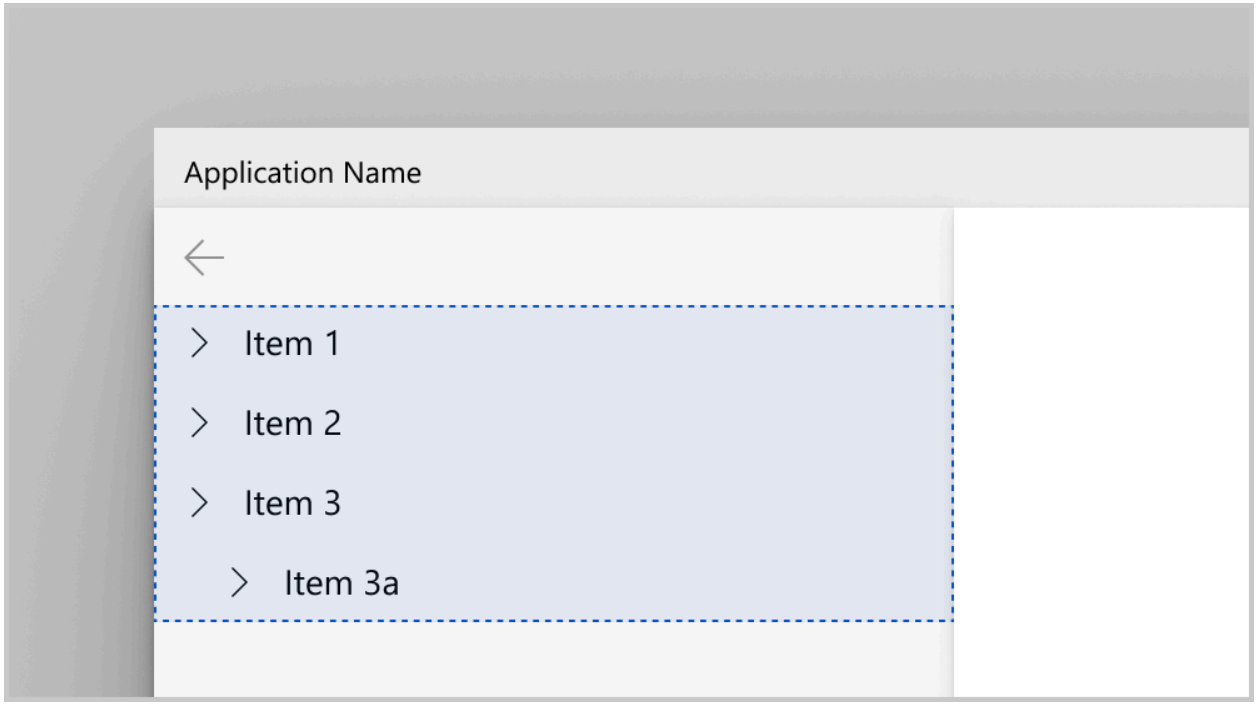
Left pane header

Pane content

You can place free-form content in the pane by adding it to the PaneCustomContent property.



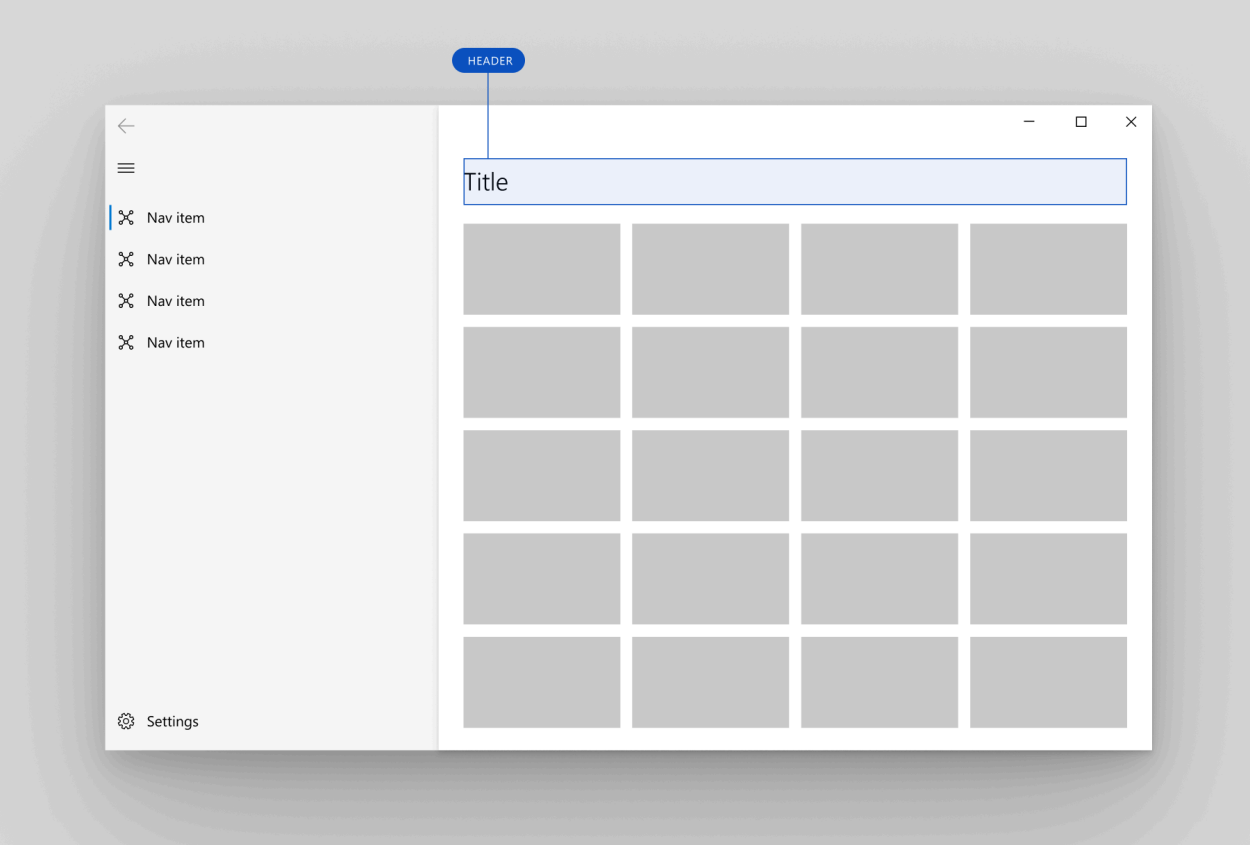
Top pane custom content



Left pane custom content

Header

You can add a page title by setting the Header property.

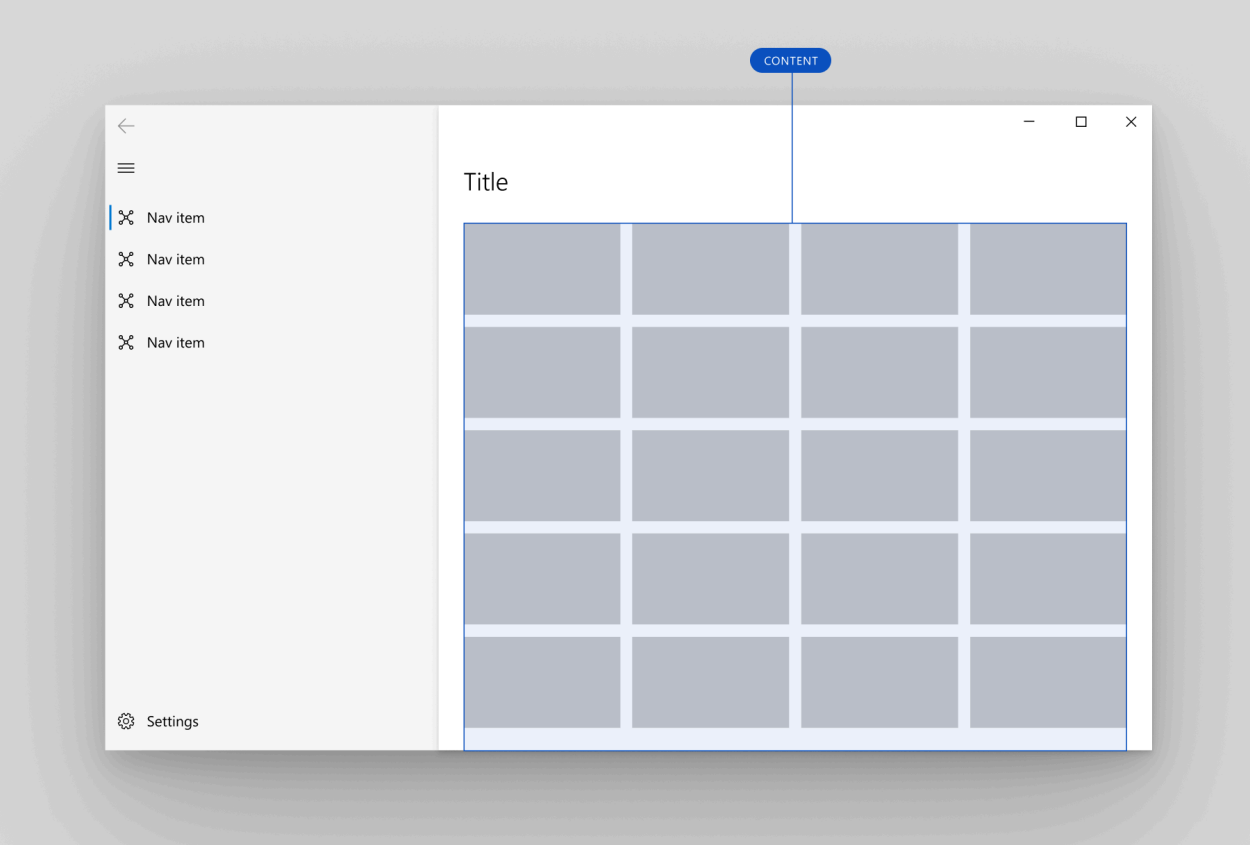


NavigationView header

The header area is vertically aligned with the navigation button in the left pane position, and lies below the pane in the top pane position. It has a fixed height of 52 px. Its purpose is to hold the page title of the selected navigation category. The header is docked to the top of the page and acts as a scroll clipping point for the content area.

The header is visible any time the NavigationView is in `Minimal` display mode. You may choose to hide the header in other modes, which are used on larger window widths. To hide the header, set the AlwaysShowHeader property to `false`.

Content



NavigationView content

The content area is where most of the information for the selected navigation category is displayed.

We recommend 12px margins for your content area when NavigationView is in `Minimal` mode and 24px margins otherwise.

Adaptive behavior

By default, the NavigationView automatically changes its display mode based on the amount of screen space available to it. The CompactModeThresholdWidth and ExpandedModeThresholdWidth properties specify the breakpoints at which the display mode changes. You can modify these values to customize the adaptive display mode behavior.

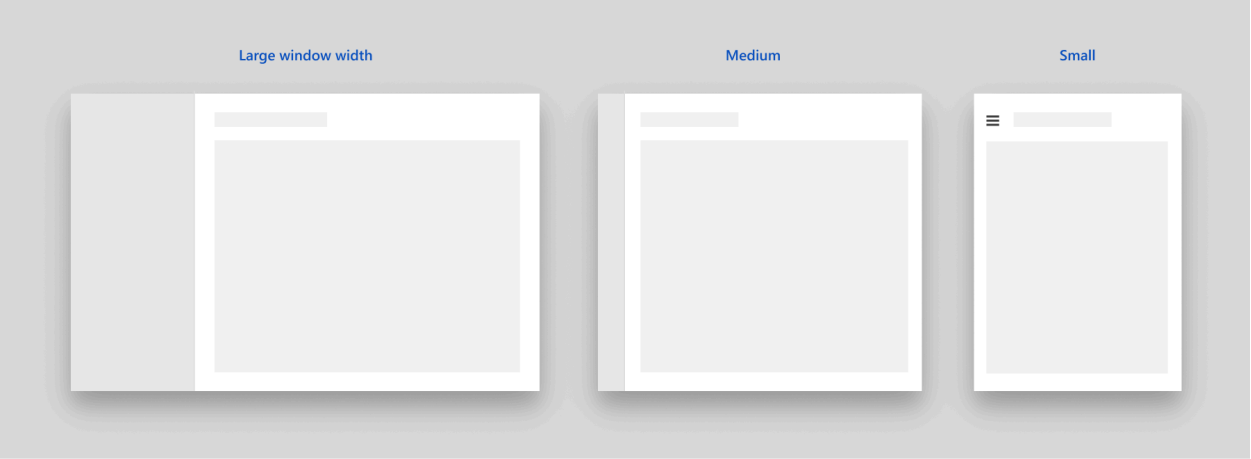
Default

When PaneDisplayMode is set to its default value of `Auto`, the adaptive behavior is to show:

- An expanded left pane on large window widths (1008px or greater).
- A left, icon-only, nav pane (`LeftCompact`) on medium window widths (641px to 1007px).

- Only a menu button (`LeftMinimal`) on small window widths (640px or less).

For more information about window sizes for adaptive behavior, see Screen sizes and breakpoints.



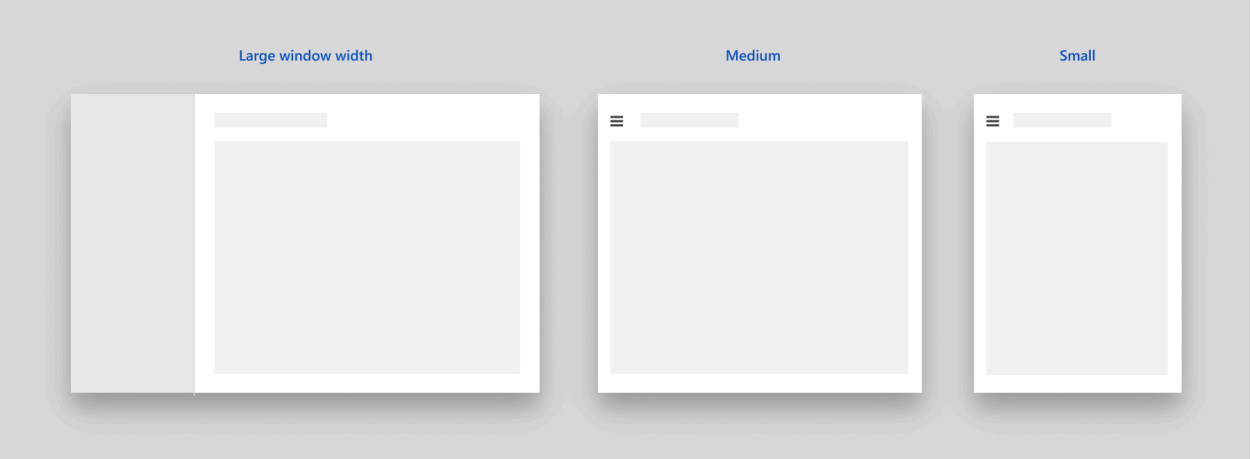
NavigationView default adaptive behavior

Minimal

A second common adaptive pattern is to use an expanded left pane on large window widths, and only a menu button on both medium and small window widths.

We recommend this when:

- You want more space for app content on smaller window widths.
- Your navigation categories cannot be clearly represented with icons.



NavigationView "minimal" adaptive behavior

To configure this behavior, set CompactModeThresholdWidth to the width at which you want the pane to collapse. Here, it's changed from the default of 640 to 1007. You should also set ExpandedModeThresholdWidth to ensure the values don't conflict.

XAML

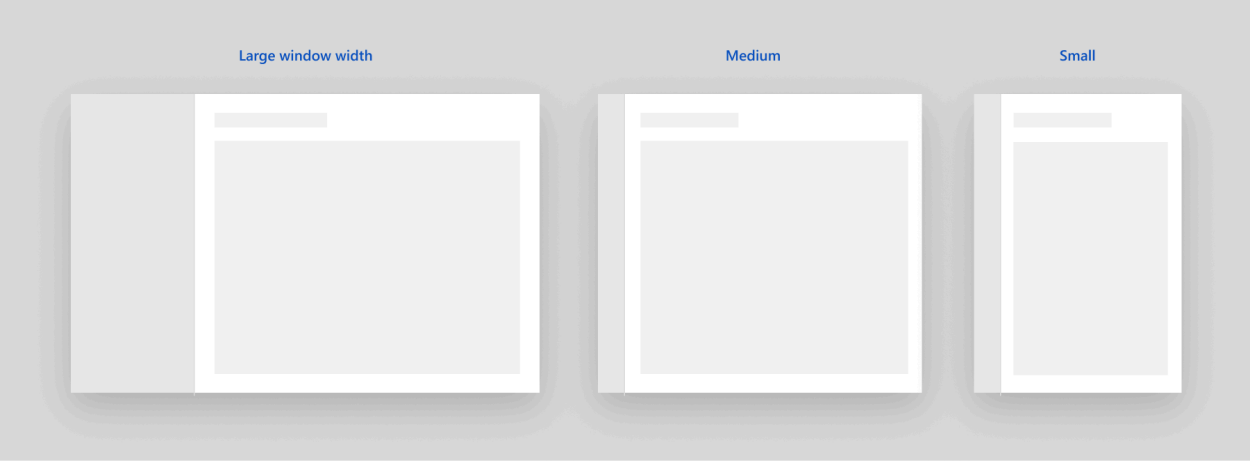
```
<NavigationView CompactModeThresholdWidth="1007"
ExpandedModeThresholdWidth="1007"/>
```

Compact

A third common adaptive pattern is to use an expanded left pane on large window widths, and a LeftCompact, icon-only, nav pane on both medium and small window widths.

We recommend this when:

- It is important to always show all navigation options on screen.
- Your navigation categories can be clearly represented with icons.



NavigationView "compact" adaptive behavior

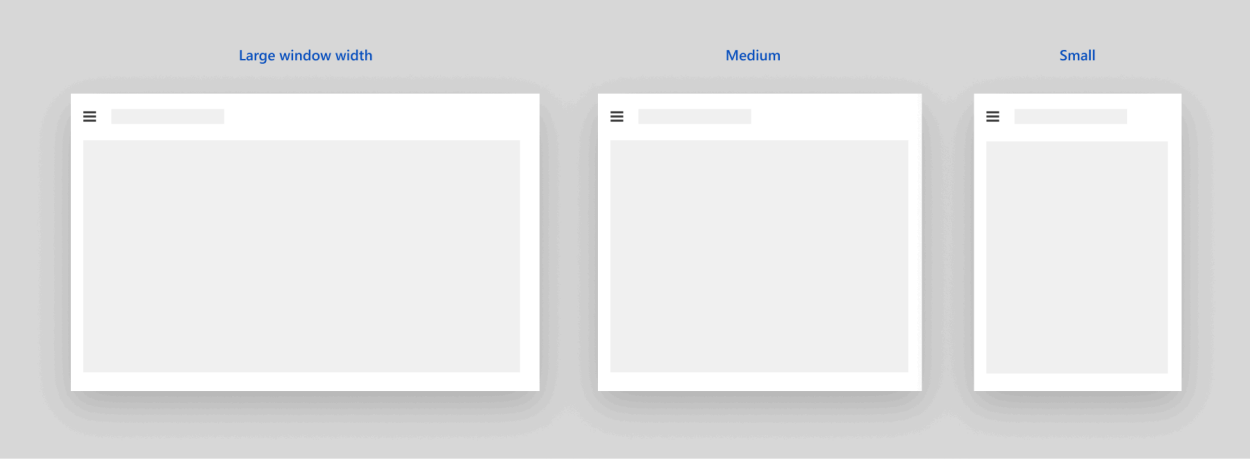
To configure this behavior, set CompactModeThresholdWidth to 0.

XAML

```
<NavigationView CompactModeThresholdWidth="0"/>
```

No adaptive behavior

To disable the automatic adaptive behavior, set PaneDisplayMode to a value other than `Auto`. Here, it's set to LeftMinimal, so only the menu button is shown regardless of the window width.



NavigationView with PaneDisplayMode set to LeftMinimal

XAML

```
<NavigationView PaneDisplayMode="LeftMinimal" />
```

As described previously in the *Display modes* section, you can set the pane to be always on top, always expanded, always compact, or always minimal. You can also manage the display modes yourself in your app code. An example of this is shown in the next section.

Top to left navigation

When you use top navigation in your app, navigation items collapse into an overflow menu as the window width decreases. When your app window is narrow, it can provide a better user experience to switch the PaneDisplayMode from `Top` to `LeftMinimal` navigation, rather than letting all the items collapse into the overflow menu.

We recommend using top navigation on large window sizes and left navigation on small window sizes when:

- You have a set of equally important top-level navigation categories to be displayed together, such that if one category in this set doesn't fit on screen, you collapse to left navigation to give them equal importance.
- You wish to preserve as much content space as possible in small window sizes.

This example shows how to use a VisualStateManager and AdaptiveTrigger.MinWindowWidth property to switch between `Top` and `LeftMinimal` navigation.

XAML

```
<Grid>
    <NavigationView x:Name="NavigationViewControl" >
        <NavigationView.MenuItems>
            <NavigationViewItem Content="A" x:Name="A" />
            <NavigationViewItem Content="B" x:Name="B" />
            <NavigationViewItem Content="C" x:Name="C" />
        </NavigationView.MenuItems>
    </NavigationView>

    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup>
            <VisualState>
                <VisualState.StateTriggers>
                    <AdaptiveTrigger
                        MinWindowWidth="{x:Bind
NavigationViewControl.CompactModeThresholdWidth}" />
                </VisualState.StateTriggers>

                <VisualState.Setters>
                    <Setter Target="NavigationViewControl.PaneDisplayMode"
Value="Top"/>
                </VisualState.Setters>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
</Grid>
```

Tip

When you use AdaptiveTrigger.MinWindowWidth, the visual state is triggered when the window is wider than the specified minimum width. This means the default XAML defines the narrow window, and the VisualState defines the modifications that are applied when the window gets wider. The default PaneDisplayMode for the NavigationView is Auto, so when the window width is less than or equal to

`CompactModeThresholdWidth`, `LeftMinimal` navigation is used. When the window gets wider, the VisualState overrides the default, and `Top` navigation is used.

Navigation

The NavigationView doesn't perform any navigation tasks automatically. When the user taps on a navigation item, the NavigationView shows that item as selected and raises an ItemInvoked event. If the tap results in a new item being selected, a SelectionChanged event is also raised.

You can handle either event to perform tasks related to the requested navigation. Which one you should handle depends on the behavior you want for your app. Typically, you navigate to the requested page and update the NavigationView header in response to these events.

- ItemInvoked is raised any time the user taps a navigation item, even if it's already selected. (The item can also be invoked with an equivalent action using mouse, keyboard, or other input. For more info, see Input and interactions.) If you navigate in the ItemInvoked handler, by default, the page will be reloaded, and a duplicate entry is added to the navigation stack. If you navigate when an item is invoked, you should disallow reloading the page, or ensure that a duplicate entry is not created in the navigation backstack when the page is reloaded. (See code examples.)
- SelectionChanged can be raised by a user invoking an item that isn't currently selected, or by programmatically changing the selected item. If the selection change occurs because a user invoked an item, the ItemInvoked event occurs first. If the selection change is programmatic, ItemInvoked is not raised.

All navigation items are part of the same selection model, whether they are a part of MenuItems or FooterMenuItems. Only one navigation item can be selected at a time.

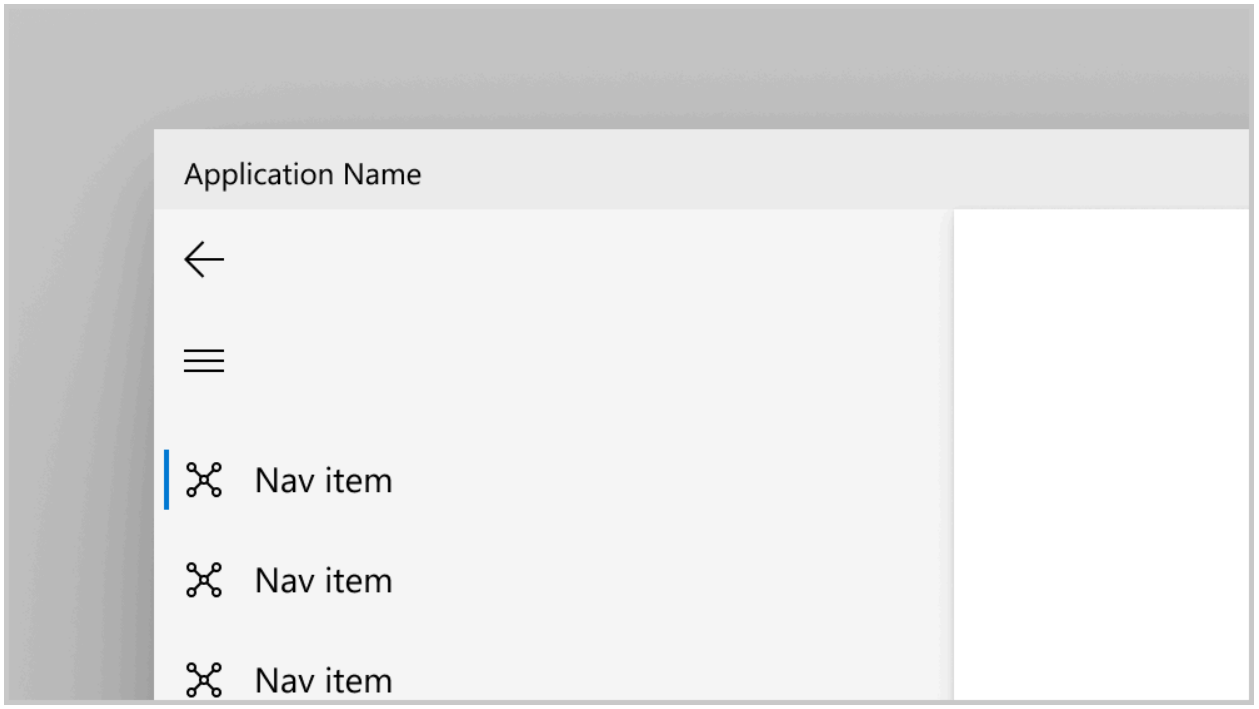
Backwards navigation

NavigationView has a built-in back button; but, as with forward navigation, it doesn't perform backwards navigation automatically. When the user taps the back button, the BackRequested event is raised. You handle this event to perform backwards navigation. For more info and code examples, see Navigation history and backwards navigation.

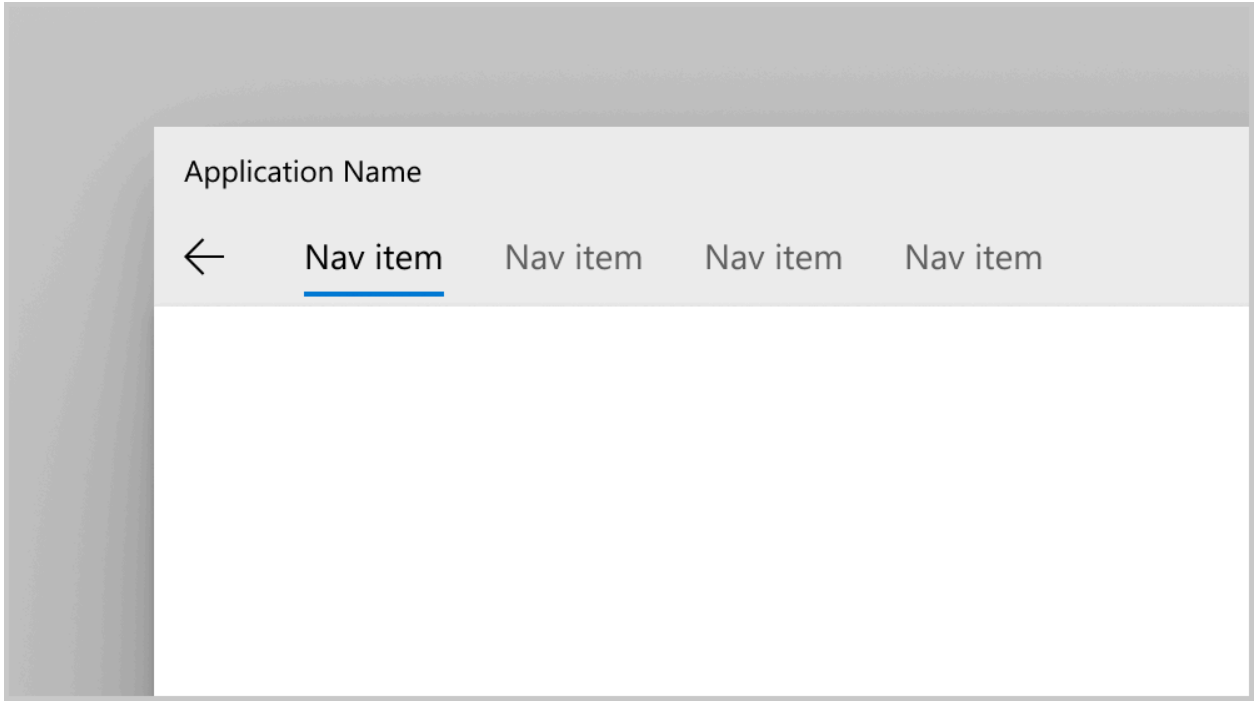
In `Minimal` or `Compact` mode, the NavigationView `Pane` is open as a flyout. In this case, clicking the back button will close the `Pane` and raise the PaneClosing event instead.

You can hide or disable the back button by setting these properties:

- IsBackButtonVisible: use to show and hide the back button. This property takes a value of the NavigationViewBackButtonVisible enumeration, and is set to `Auto` by default. When the button is collapsed, no space is reserved for it in the layout.
- IsBackEnabled: use to enable or disable the back button. You can data bind this property to the CanGoBack property of your navigation frame. BackRequested is not raised if `IsBackEnabled` is `false`.



The back button in the left navigation pane



The back button in the top navigation pane

Code example

This example shows how you can use NavigationView with both a top navigation pane on large window sizes and a left navigation pane on small window sizes. It can be adapted to left-only navigation by removing the *top* navigation settings in the VisualStateManager.

The example demonstrates a common way to set up navigation data that will work for many scenarios. In this example, you first store (in the tag of the NavigationViewItem) the full type name of the page to which you want to navigate. In the event handler, you unbox that value, turn it into a Type(C#) or **Windows::UI::Xaml::Interop::TypeName**(C++/WinRT) object, and use that to navigate to the destination page. This lets you create unit tests to confirm that the values inside your tags are of a valid type. (Also see Boxing and unboxing values to IInspectable with C++/WinRT). It also demonstrates how to implement backwards navigation with NavigationView's back button.

This code assumes that your app contains pages with the following names to navigate to: *HomePage*, *AppsPage*, *GamesPage*, *MusicPage*, *MyContentPage*, and *SettingsPage*. Code for these pages is not shown.

XAML

```
<Page ... >
 <Grid>
     <NavigationView x:Name="NavView"
                     Loaded="NavView_Loaded"
                     ItemInvoked="NavView_ItemInvoked"
                     BackRequested="NavView_BackRequested">
         <NavigationView.MenuItems>
             <NavigationViewItem Tag="NavigationViewDemo.HomePage"
Icon="Home" Content="Home"/>
             <NavigationViewItemSeparator/>
             <NavigationViewItemHeader x:Name="MainPagesHeader"
                                       Content="Main pages"/>
             <NavigationViewItem Tag="NavigationViewDemo.AppsPage"
Content="Apps">
                 <NavigationViewItem.Icon>
                     <FontIcon Glyph="&#xEB3C;"/>
                 </NavigationViewItem.Icon>
             </NavigationViewItem>
             <NavigationViewItem Tag="NavigationViewDemo.GamesPage"
Content="Games">
                 <NavigationViewItem.Icon>
                     <FontIcon Glyph="&#xE7FC;"/>
                 </NavigationViewItem.Icon>
             </NavigationViewItem>
             <NavigationViewItem Tag="NavigationViewDemo.MusicPage"
```

```
Icon="Audio" Content="Music"/>
        </NavigationView.MenuItems>

        <NavigationView.AutoSuggestBox>
            <!-- See AutoSuggestBox documentation for
             more info about how to implement search. -->
            <AutoSuggestBox x:Name="NavViewSearchBox" QueryIcon="Find"/>
        </NavigationView.AutoSuggestBox>

        <ScrollViewer>
            <Frame x:Name="ContentFrame" IsTabStop="True"
               NavigationFailed="ContentFrame_NavigationFailed"/>
        </ScrollViewer>
    </NavigationView>

    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup>
            <VisualState>
                <VisualState.StateTriggers>
                    <AdaptiveTrigger
                    MinWindowWidth="{x:Bind
NavViewCompactModeThresholdWidth}"/>
                </VisualState.StateTriggers>
                <VisualState.Setters>
                    <!-- Remove the next 3 lines for left-only navigation.
-->
                    <Setter Target="NavView.PaneDisplayMode" Value="Top"/>
                    <Setter Target="NavViewSearchBox.Width" Value="200"/>
                    <Setter Target="MainPagesHeader.Visibility"
Value="Collapsed"/>
                    <!-- Leave the next line for left-only navigation. -->
                    <Setter Target="ContentFrame.Padding"
Value="24,0,24,24"/>
                </VisualState.Setters>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
 </Grid>
</Page>
```

C#

```
private double NavViewCompactModeThresholdWidth { get { return
NavView.CompactModeThresholdWidth; } }

private void ContentFrame_NavigationFailed(object sender,
NavigationFailedEventArgs e)
{
    throw new Exception("Failed to load Page " + e.SourcePageType.FullName);
}

private void NavView_Loaded(object sender, RoutedEventArgs e)
{
```

```
    // You can also add items in code.
    NavView.MenuItems.Add(new NavigationViewItemSeparator());
    NavView.MenuItems.Add(new NavigationViewItem
    {
        Content = "My content",
        Icon = new SymbolIcon((Symbol)0xF1AD),
        Tag = "NavigationViewDemo.MyContentPage"
    });

    // Add handler for ContentFrame navigation.
    ContentFrame.Navigated += On_Navigated;

    // NavView doesn't load any page by default, so load home page.
    NavView.SelectedItem = NavView.MenuItems[0];
    // If navigation occurs on SelectionChanged, this isn't needed.
    // Because we use ItemInvoked to navigate, we need to call Navigate
    // here to load the home page.
    NavView_Navigate(typeof(HomePage), new
EntranceNavigationTransitionInfo());
}

private void NavView_ItemInvoked(NavigationView sender,
                                 NavigationViewItemInvokedEventArgs args)
{
    if (args.IsSettingsInvoked == true)
    {
        NavView_Navigate(typeof(SettingsPage),
args.RecommendedNavigationTransitionInfo);
    }
    else if (args.InvokedItemContainer != null)
    {
        Type navPageType =
Type.GetType(args.InvokedItemContainer.Tag.ToString());
        NavView_Navigate(navPageType,
args.RecommendedNavigationTransitionInfo);
    }
}

// NavView_SelectionChanged is not used in this example, but is shown for
completeness.
// You will typically handle either ItemInvoked or SelectionChanged to
perform navigation,
// but not both.
private void NavView_SelectionChanged(NavigationView sender,

NavigationViewSelectionChangedEventArgs args)
{
    if (args.IsSettingsSelected == true)
    {
        NavView_Navigate(typeof(SettingsPage),
args.RecommendedNavigationTransitionInfo);
    }
    else if (args.SelectedItemContainer != null)
    {
        Type navPageType =
```

```
Type.GetType(args.SelectedItemContainer.Tag.ToString());
        NavView_Navigate(navPageType,
args.RecommendedNavigationTransitionInfo);
    }
}

private void NavView_Navigate(
    Type navPageType,
    NavigationTransitionInfo transitionInfo)
{
    // Get the page type before navigation so you can prevent duplicate
    // entries in the backstack.
    Type preNavPageType = ContentFrame.CurrentSourcePageType;

    // Only navigate if the selected page isn't currently loaded.
    if (navPageType is not null && !Type.Equals(preNavPageType,
navPageType))
    {
        ContentFrame.Navigate(navPageType, null, transitionInfo);
    }
}

private void NavView_BackRequested(NavigationView sender,
                                   NavigationViewBackRequestedEventArgs
args)
{
    TryGoBack();
}

private bool TryGoBack()
{
    if (!ContentFrame.CanGoBack)
        return false;

    // Don't go back if the nav pane is overlayed.
    if (NavView.IsPaneOpen &&
        (NavView.DisplayMode == NavigationViewDisplayMode.Compact ||
         NavView.DisplayMode == NavigationViewDisplayMode.Minimal))
        return false;

    ContentFrame.GoBack();
    return true;
}

private void On_Navigated(object sender, NavigationEventArgs e)
{
    NavView.IsBackEnabled = ContentFrame.CanGoBack;

    if (ContentFrame.SourcePageType == typeof(SettingsPage))
    {
        // SettingsItem is not part of NavView.MenuItems, and doesn't have a
Tag.
        NavView.SelectedItem = (NavigationViewItem)NavView.SettingsItem;
        NavView.Header = "Settings";
    }
```

```
    else if (ContentFrame.SourcePageType != null)
    {
        // Select the nav view item that corresponds to the page being
navigated to.
        NavView.SelectedItem = NavView.MenuItems
                    .OfType<NavigationViewItem>()
                    .First(i =>
i.Tag.Equals(ContentFrame.SourcePageType.FullName.ToString()));

        NavView.Header =
            ((NavigationViewItem)NavView.SelectedItem)?.Content?.ToString();

    }
}
```

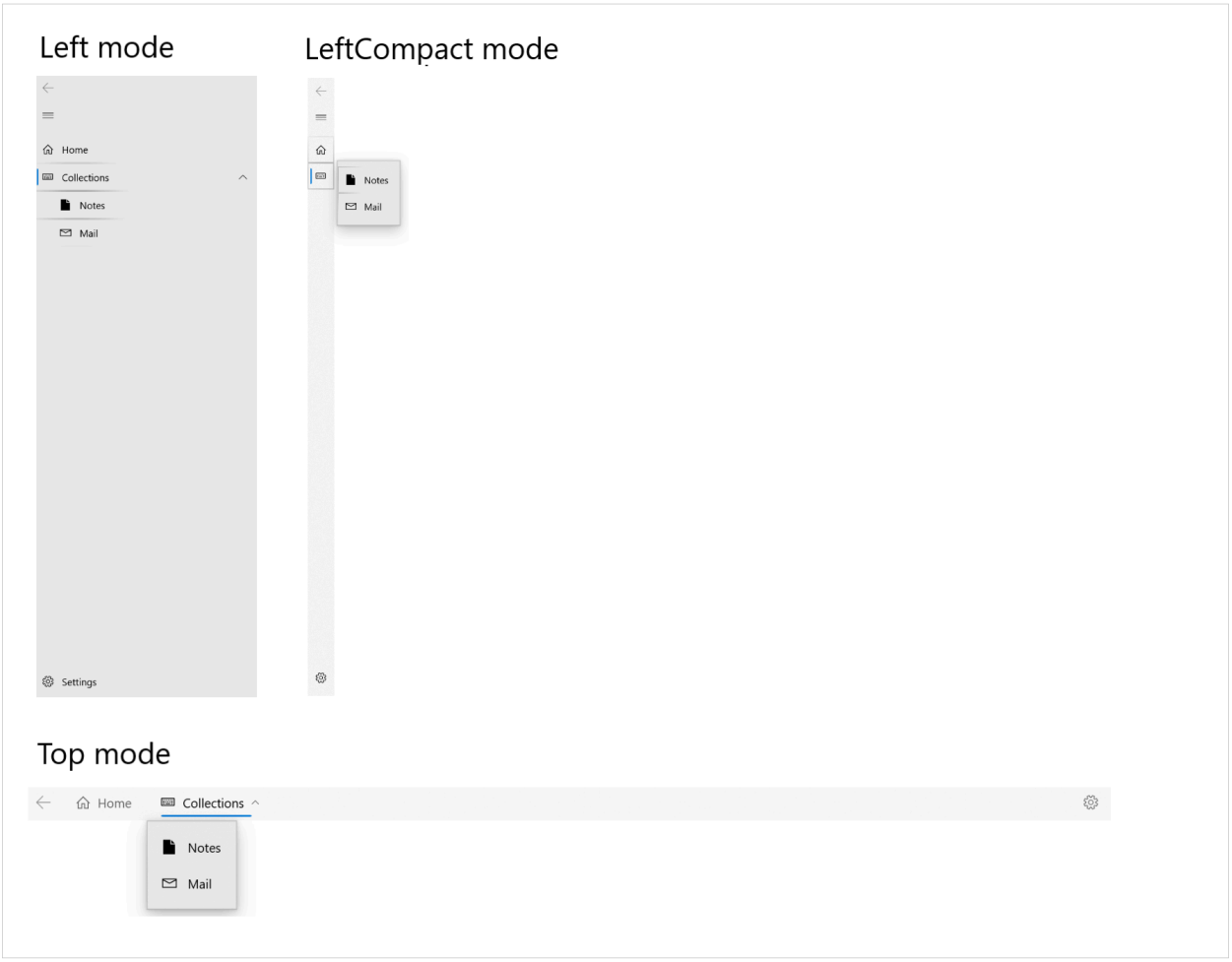
Hierarchical navigation

Some apps may have a more complex hierarchical structure that requires more than just a flat list of navigation items. You may want to use top-level navigation items to display categories of pages, with children items displaying specific pages. It is also useful if you have hub-style pages that only link to other pages. For these kinds of cases, you should create a hierarchical NavigationView.

To show a hierarchical list of nested navigation items in the pane, use either the MenuItems property or the MenuItemsSource property of **NavigationViewItem**. Each NavigationViewItem can contain other NavigationViewItems and organizing elements like item headers and separators. To show a hierarchical list when using `MenuItemsSource`, set the `ItemTemplate` to be a NavigationViewItem, and bind its `MenuItemsSource` property to the next level of the hierarchy.

Although NavigationViewItem can contain any number of nested levels, we recommend keeping your app's navigation hierarchy shallow. We believe two levels is ideal for usability and comprehension.

NavigationView shows hierarchy in `Top`, `Left`, and `LeftCompact` pane display modes. Here is what an expanded subtree looks like in each of the pane display modes:



Adding a hierarchy of items in markup

This example shows how to declare hierarchical app navigation in XAML markup.

XAML

```
<NavigationView>
    <NavigationView.MenuItems>
        <NavigationViewItem Content="Home" Icon="Home"
ToolTipService.ToolTip="Home"/>
        <NavigationViewItem Content="Collections" Icon="Keyboard"
ToolTipService.ToolTip="Collections">
            <NavigationViewItem.MenuItems>
                <NavigationViewItem Content="Notes" Icon="Page"
ToolTipService.ToolTip="Notes"/>
                <NavigationViewItem Content="Mail" Icon="Mail"
ToolTipService.ToolTip="Mail"/>
            </NavigationViewItem.MenuItems>
        </NavigationViewItem>
    </NavigationView.MenuItems>
</NavigationView>
```

Adding a hierarchy of items using data binding

Add a hierarchy of menu items to the NavigationView by

- binding the MenuItemsSource property to the hierarchical data
- defining the item template to be a NavigationViewMenuItem, with its Content set to be the label of the menu item, and its MenuItemsSource property bound to the next level of the hierarchy

This example also demonstrates the Expanding and Collapsed events. These events are raised for a menu item with children.

XAML

```
<Page ... >
    <Page.Resources>
        <DataTemplate x:Key="NavigationViewMenuItem"
x:DataType="local:Category">
            <NavigationViewItem Content="{x:Bind Name}" MenuItemsSource="
{x:Bind Children}"/>
        </DataTemplate>
    </Page.Resources>

    <Grid>
        <NavigationView x:Name="navview"
    MenuItemsSource="{x:Bind Categories, Mode=OneWay}"
    MenuItemTemplate="{StaticResource NavigationViewMenuItem}"
    ItemInvoked="{x:Bind OnItemInvoked}"
    Expanding="OnItemExpanding"
    Collapsed="OnItemCollapsed"
    PaneDisplayMode="Left">
            <StackPanel Margin="10,10,0,0">
                <TextBlock Margin="0,10,0,0" x:Name="ExpandingItemLabel"
Text="Last Expanding: N/A"/>
                <TextBlock x:Name="CollapsedItemLabel" Text="Last Collapsed:
N/A"/>
            </StackPanel>
        </NavigationView>
    </Grid>
</Page>
```

C#

```
public class Category
{
    public String Name { get; set; }
    public String CategoryIcon { get; set; }
    public ObservableCollection<Category> Children { get; set; }
}

public sealed partial class HierarchicalNavigationViewDataBinding : Page
{
    public HierarchicalNavigationViewDataBinding()
```

```
    {
        this.InitializeComponent();
    }

    public ObservableCollection<Category> Categories = new
ObservableCollection<Category>()
    {
        new Category(){
            Name = "Menu item 1",
            CategoryIcon = "Icon",
            Children = new ObservableCollection<Category>() {
                new Category(){
                    Name = "Menu item 2",
                    CategoryIcon = "Icon",
                    Children = new ObservableCollection<Category>() {
                        new Category() {
                            Name  = "Menu item 3",
                            CategoryIcon = "Icon",
                            Children = new ObservableCollection<Category>()
{
                                new Category() { Name  = "Menu item 4",
CategoryIcon = "Icon" },
                                new Category() { Name  = "Menu item 5",
CategoryIcon = "Icon" }
                            }
                        }
                    }
                }
            }
        },
        new Category(){
            Name = "Menu item 6",
            CategoryIcon = "Icon",
            Children = new ObservableCollection<Category>() {
                new Category(){
                    Name = "Menu item 7",
                    CategoryIcon = "Icon",
                    Children = new ObservableCollection<Category>() {
                        new Category() { Name  = "Menu item 8", CategoryIcon
= "Icon" },
                        new Category() { Name  = "Menu item 9", CategoryIcon
= "Icon" }
                    }
                }
            }
        },
        new Category(){ Name = "Menu item 10", CategoryIcon = "Icon" }
    };

    private void OnItemInvoked(object sender,
NavigationViewItemInvokedEventArgs e)
    {
        var clickedItem = e.InvokedItem;
        var clickedItemContainer = e.InvokedItemContainer;
    }
```

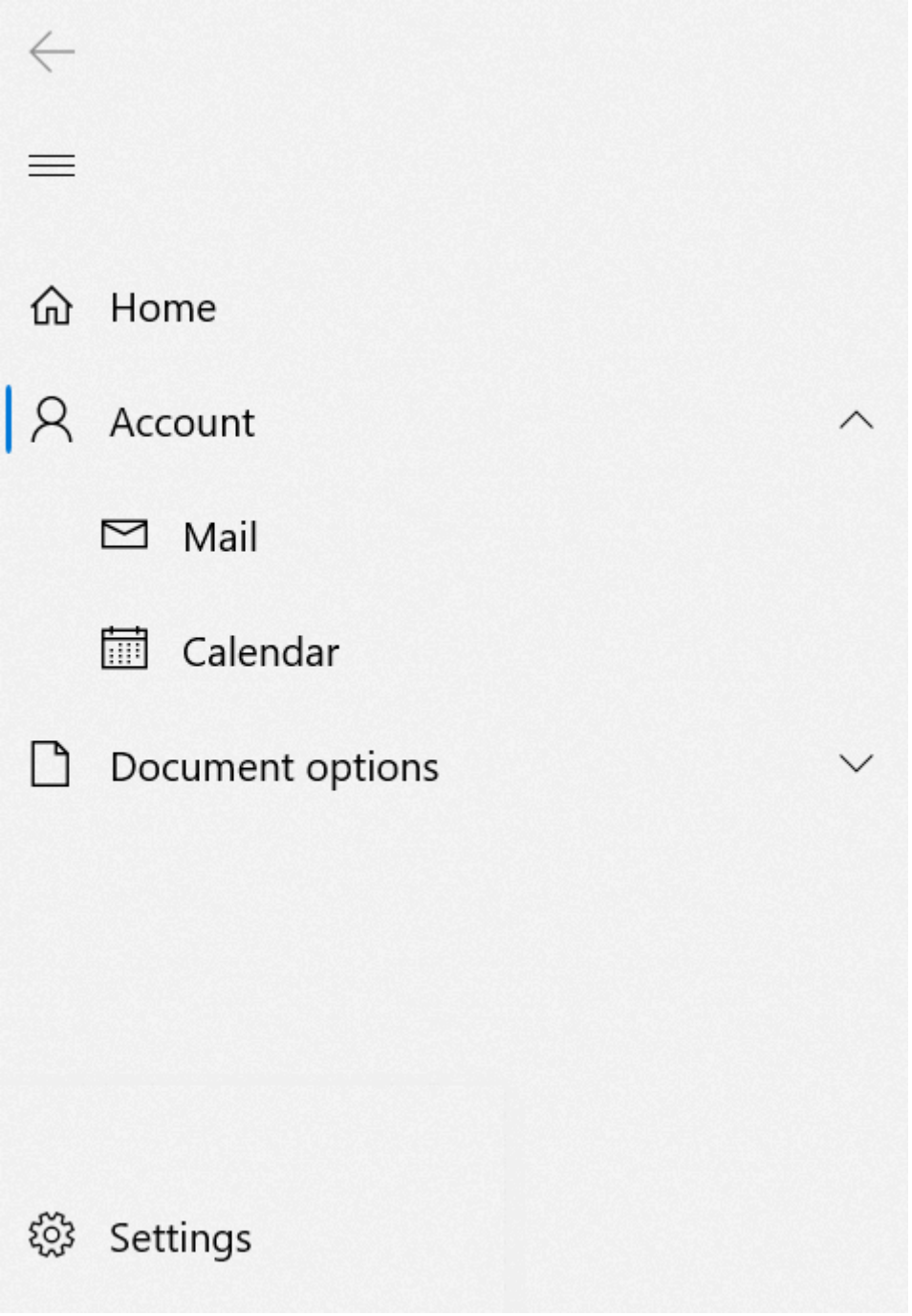
```
    private void OnItemExpanding(object sender,
NavigationViewItemExpandingEventArgs e)
    {
        var nvib = e.ExpandingItemContainer;
        var name = "Last expanding: " + nvib.Content.ToString();
        ExpandingItemLabel.Text = name;
    }
    private void OnItemCollapsed(object sender,
NavigationViewItemCollapsedEventArgs e)
    {
        var nvib = e.CollapsedItemContainer;
        var name = "Last collapsed: " + nvib.Content;
        CollapsedItemLabel.Text = name;
    }
}
```

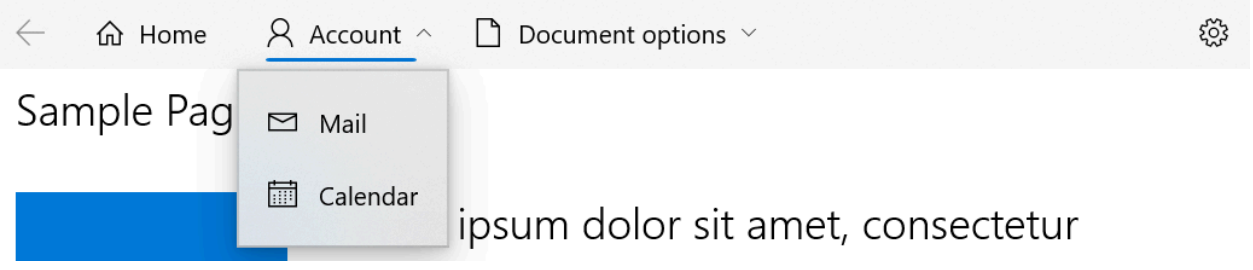
Selection

By default, any item can contain children, be invoked, or be selected.

When providing users with a hierarchical tree of navigation options, you may choose to make parent items non-selectable, for example when your app doesn't have a destination page associated with that parent item. If your parent items *are* selectable, it's recommend you use the Left-Expanded or Top pane display modes. LeftCompact mode will cause the user to navigate to the parent item in order to open the child subtree every time it's invoked.

Selected items will draw their selection indicators along their left edge when in left mode or their bottom edge when in top mode. Shown below are NavigationViews in left and top mode where a parent item is selected.





The selected item may not always remain visible. If a child in a collapsed/non-expanded subtree is selected, their first visible ancestor will show as selected. The selection indicator will move back to the selected item if/when the sub-tree is expanded.

For example - in the above image, the Calendar item may be selected by the user, and then the user may collapse its subtree. In this case, the selection indicator would show up underneath the Account item as Account is Calendar's first visible ancestor. The

selection indicator will move back to the Calendar item as the user expands the subtree again.

The entire NavigationView will show no more than one selection indicator.

In both Top and Left modes, clicking the arrows on NavigationViewItems will expand or collapse the subtree. Clicking or tapping *elsewhere* on the NavigationViewItem will trigger the `ItemInvoked` event, and it will also collapse or expand the subtree.

To prevent an item from showing the selection indicator when invoked, set its SelectsOnInvoked property to False, as shown below:

XAML

```
<Page ...>
    <Page.Resources>
        <DataTemplate x:Key="NavigationViewMenuItem"
x:DataType="local:Category">
            <NavigationViewItem Content="{x:Bind Name}"
            MenuItemsSource="{x:Bind Children}"
            SelectsOnInvoked="{x:Bind IsLeaf}"/>
        </DataTemplate>
    </Page.Resources>

    <Grid>
        <NavigationView x:Name="navview"
    MenuItemsSource="{x:Bind Categories, Mode=OneWay}"
    MenuItemTemplate="{StaticResource NavigationViewMenuItem}">
        </NavigationView>
    </Grid>
</Page>
```

C#

```
public class Category
{
    public String Name { get; set; }
    public String CategoryIcon { get; set; }
    public ObservableCollection<Category> Children { get; set; }
    public bool IsLeaf { get; set; }
}

public sealed partial class HierarchicalNavigationViewDataBinding : Page
{
    public HierarchicalNavigationViewDataBinding()
    {
        this.InitializeComponent();
    }

    public ObservableCollection<Category> Categories = new
ObservableCollection<Category>()
```

```
    {
        new Category(){
            Name = "Menu item 1",
            CategoryIcon = "Icon",
            Children = new ObservableCollection<Category>() {
                new Category(){
                    Name = "Menu item 2",
                    CategoryIcon = "Icon",
                    Children = new ObservableCollection<Category>() {
                        new Category() {
                            Name  = "Menu item 3",
                            CategoryIcon = "Icon",
                            Children = new ObservableCollection<Category>()
{
                                new Category() { Name  = "Menu item 4",
CategoryIcon = "Icon", IsLeaf = true },
                                new Category() { Name  = "Menu item 5",
CategoryIcon = "Icon", IsLeaf = true }
                            }
                        }
                    }
                }
            }
        },
        new Category(){
            Name = "Menu item 6",
            CategoryIcon = "Icon",
            Children = new ObservableCollection<Category>() {
                new Category(){
                    Name = "Menu item 7",
                    CategoryIcon = "Icon",
                    Children = new ObservableCollection<Category>() {
                        new Category() { Name  = "Menu item 8", CategoryIcon
= "Icon", IsLeaf = true },
                        new Category() { Name  = "Menu item 9", CategoryIcon
= "Icon", IsLeaf = true }
                    }
                }
            }
        },
        new Category(){ Name = "Menu item 10", CategoryIcon = "Icon", IsLeaf
= true }
    };
}
```

Keyboarding within hierarchical NavigationView

Users can move focus around the NavigationView using their keyboard. The arrow keys expose "inner navigation" within the pane and follow the interactions provided in tree view. The key actions change when navigating through the NavigationView or its flyout

menu, which is displayed in Top and Left-compact modes of HierarchicalNavigationView. Below are the specific actions that each key can take in a hierarchical NavigationView:

Expand table

Key	In Left Mode	In Top Mode	In Flyout
Up	Moves focus to the item directly above the item currently in focus.	Does nothing.	Moves focus to the item directly above the item currently in focus.
Down	Moves focus directly below the item currently in focus.*	Does nothing.	Moves focus directly below the item currently in focus.*
Right	Does nothing.	Moves focus to the item directly to the right of the item currently in focus.	Does nothing.
Left	Does nothing.	Moves focus to the item directly to the left the item currently in focus.	Does nothing.
Space/Enter	If item has children, expands/collapses item and does not change focus.	If item has children, expands children into a flyout and places focus on first item in flyout.	Invokes/selects item and closes flyout.
Esc	Does nothing.	Does nothing.	Closes flyout.

The space or enter key always invokes/selects an item.

*Note that the items do not need to be visually adjacent, focus will move from the last item in the pane's list to the settings item.

NavigationView customization

Pane Backgrounds

By default, the NavigationView pane uses a different background depending on the display mode:

- the pane is a solid grey color when expanded on the left, side-by-side with the content (in Left mode).

- the pane uses in-app acrylic when open as an overlay on top of content (in Top, Minimal, or Compact mode).

To modify the pane background, you can override the XAML theme resources used to render the background in each mode. (This technique is used rather than a single PaneBackground property in order to support different backgrounds for different display modes.)

This table shows which theme resource is used in each display mode.

Expand table

Display mode	Theme resource
Left	NavigationViewExpandedPaneBackground
LeftCompact LeftMinimal	NavigationViewDefaultPaneBackground
Top	NavigationViewTopPaneBackground

This example shows how to override the theme resources in App.xaml. When you override theme resources, you should always provide "Default" and "HighContrast" resource dictionaries at a minimum, and dictionaries for "Light" or "Dark" resources as needed. For more info, see ResourceDictionary.ThemeDictionaries.

> **Important**
>
> This code shows how to use the WinUI 2 version of AcrylicBrush. If you use the platform version of AcrylicBrush instead, the minimum version for your app project must be SDK 16299 or greater. To use the platform version, remove all references to `muxm:`.

XAML

```
<Application ... xmlns:muxm="using:Microsoft.UI.Xaml.Media" ...>
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <XamlControlsResources
xmlns="using:Microsoft.UI.Xaml.Controls"/>
                <ResourceDictionary>
                    <ResourceDictionary.ThemeDictionaries>
                        <ResourceDictionary x:Key="Default">
                            <!-- The "Default" theme dictionary is used
unless a specific
```

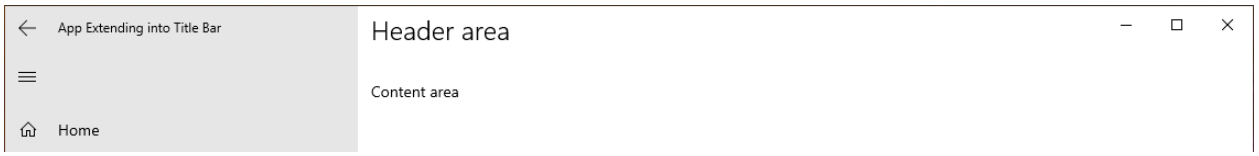
```
                                light, dark, or high contrast dictionary is
provided. These
                                resources should be tested with both the
light and dark themes,
                                and specific light or dark resources
provided as needed. -->
                            <muxm:AcrylicBrush
x:Key="NavigationViewDefaultPaneBackground"
                                BackgroundSource="Backdrop"
                                TintColor="LightSlateGray"
                                TintOpacity=".6"/>
                            <muxm:AcrylicBrush
x:Key="NavigationViewTopPaneBackground"
                                BackgroundSource="Backdrop"
                                TintColor="{ThemeResource
SystemAccentColor}"
                                TintOpacity=".6"/>
                            <LinearGradientBrush
x:Key="NavigationViewExpandedPaneBackground"
                                  StartPoint="0.5,0" EndPoint="0.5,1">
                                <GradientStop Color="LightSlateGray"
Offset="0.0" />
                                <GradientStop Color="White" Offset="1.0" />
                            </LinearGradientBrush>
                        </ResourceDictionary>
                        <ResourceDictionary x:Key="HighContrast">
                            <!-- Always include a "HighContrast" dictionary
when you override
                                theme resources. This empty dictionary
ensures that the
                                default high contrast resources are used
when the user
                                turns on high contrast mode. -->
                        </ResourceDictionary>
                    </ResourceDictionary.ThemeDictionaries>
                </ResourceDictionary>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

Top whitespace

> The `IsTitleBarAutoPaddingEnabled` property requires WinUI 2.2 or later.

Some apps choose to customize their window's title bar, potentially extending their app content into the title bar area. When NavigationView is the root element in apps that extend into the title bar **using the ExtendViewIntoTitleBar API**, the control automatically adjusts the position of its interactive elements to prevent overlap with the draggable region.

If your app specifies the draggable region by calling the Window.SetTitleBar method and you would prefer to have the back and menu buttons draw closer to the top of your app window, set IsTitleBarAutoPaddingEnabled to `false`.



XAML

```
<muxc:NavigationView x:Name="NavView" IsTitleBarAutoPaddingEnabled="False">
```

Remarks

To further adjust the position of NavigationView's header area, override the *NavigationViewHeaderMargin* XAML theme resource, for example in your Page resources.

XAML

```
<Page.Resources>
    <Thickness x:Key="NavigationViewHeaderMargin">12,0</Thickness>
</Page.Resources>
```

This theme resource modifies the margin around NavigationView.Header.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

The NavigationView control for UWP apps is included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in both the Windows.UI.Xaml.Controls and Microsoft.UI.Xaml.Controls namespaces.

- ✓ **UWP APIs:** Windows.UI.Xaml.Controls.NavigationView class
- ✓ **WinUI 2 Apis:** Microsoft.UI.Xaml.Controls.NavigationView class
- ✓ Open the WinUI 2 Gallery app and see the NavigationView in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles, templates, and features for all controls. Some features of NavigationView, such as *top* and *hierarchical* navigation, require Windows 10, version 1809 (SDK 17763) or later, or WinUI 2.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:NavigationView />
```

Related topics

- NavigationView class
- List/details
- Navigation basics
- Fluent Design overview

Pivot

Article • 02/26/2025

The Pivot control enables touch-swiping between a small set of content sections.

Is this the right control?

> ⚠ **Warning**
>
> The Pivot control is not recommended for **Windows 11 design patterns**. We strongly recommend using one of these alternatives instead:
>
> - **WinUI 3** - Use the **SelectorBar** control.
> - **WinUI 2/UWP** - Use a **NavigationView** or **TabView** control instead of a Pivot control. See the **Use NavigationView instead of Pivot** section for an example.

To create a Pivot-like UI when using WinUI 3 and the Windows App SDK, use the SelectorBar control.

To create a tabbed UI, use a TabView control.

To achieve common top navigation patterns, we recommend using NavigationView, which automatically adapts to different screen sizes and allows for greater customization.

Some key differences between the NavigationView and Pivot are listed here:

- Pivot supports touch-swiping to switch between items.

- Overflow items in a Pivot carousel, while NavigationView uses a menu dropdown overflow so that users can see all items.
- Pivot handles navigation between content sections, while NavigationView allows for more control over navigation behavior.

Use NavigationView instead of Pivot

If your app's UI uses the Pivot control, you can convert Pivot to NavigationView following this example.

This XAML creates a NavigationView with 3 sections of content, like the example Pivot in Create a pivot control.

XAML

```
<NavigationView x:Name="rootNavigationView" Header="Category Title"
 ItemInvoked="NavView_ItemInvoked">
    <NavigationView.MenuItems>
        <NavigationViewItem Content="Section 1" x:Name="Section1Content" />
        <NavigationViewItem Content="Section 2" x:Name="Section2Content" />
        <NavigationViewItem Content="Section 3" x:Name="Section3Content" />
    </NavigationView.MenuItems>

    <Frame x:Name="ContentFrame" />
</NavigationView>

<Page x:Class="AppName.Section1Page">
    <TextBlock Text="Content of section 1."/>
</Page>

<Page x:Class="AppName.Section2Page">
    <TextBlock Text="Content of section 2."/>
</Page>

<Page x:Class="AppName.Section3Page">
    <TextBlock Text="Content of section 3."/>
</Page>
```

NavigationView provides more control over navigation customization and requires corresponding code-behind. To accompany the above XAML, use the following code-behind:

C#

```
private void NavView_ItemInvoked(NavigationView sender,
NavigationViewItemInvokedEventArgs args)
{
   var navOptions = new FrameNavigationOptions
```

```
    {
        TransitionInfoOverride = args.RecommendedNavigationTransitionInfo,
        IsNavigationStackEnabled = false,
    };

    switch (args.InvokedItemContainer.Name)
    {
        case nameof(Section1Content):
            ContentFrame.NavigateToType(typeof(Section1Page), null,
navOptions);
            break;

        case nameof(Section2Content):
            ContentFrame.NavigateToType(typeof(Section2Page), null,
navOptions);
            break;

        case nameof(Section3Content):
            ContentFrame.NavigateToType(typeof(Section3Page), null,
navOptions);
            break;
    }
}
```

This code mimics the Pivot control's built-in navigation experience, minus the touch-swiping experience between content sections. However, as you can see, you could also customize several points, including the animated transition, navigation parameters, and stack capabilities.

Create a pivot control

> ⚠ **Warning**
>
> The Pivot control is not recommended for Windows 11 design patterns. We strongly recommend using one of these alternatives instead:
>
> - **WinUI 3** - Use the SelectorBar control.
> - **WinUI 2/UWP** - Use a NavigationView or TabView control instead of a Pivot control. See the Use NavigationView instead of Pivot section for an example.

This XAML creates a basic Pivot control with 3 sections of content.

XAML

```
<Pivot x:Name="rootPivot" Title="Category Title">
    <PivotItem Header="Section 1">
```

```
        <!--Pivot content goes here-->
        <TextBlock Text="Content of section 1."/>
    </PivotItem>
    <PivotItem Header="Section 2">
        <!--Pivot content goes here-->
        <TextBlock Text="Content of section 2."/>
    </PivotItem>
    <PivotItem Header="Section 3">
        <!--Pivot content goes here-->
        <TextBlock Text="Content of section 3."/>
    </PivotItem>
</Pivot>
```

Pivot items

Pivot is an ItemsControl, so it can contain a collection of items of any type. Any item you add to the Pivot that is not explicitly a PivotItem is implicitly wrapped in a PivotItem. Because a Pivot is often used to navigate between pages of content, it's common to populate the Items collection directly with XAML UI elements. Or, you can set the ItemsSource property to a data source. Items bound in the ItemsSource can be of any type, but if they aren't explicitly PivotItems, you must define an ItemTemplate and HeaderTemplate to specify how the items are displayed.

You can use the SelectedItem property to get or set the Pivot's active item. Use the SelectedIndex property to get or set the index of the active item.

Pivot headers

You can use the LeftHeader and RightHeader properties to add other controls to the Pivot header.

For example, you can add a CommandBar in the Pivot's RightHeader.

XAML

```
<Pivot>
    <Pivot.RightHeader>
        <CommandBar>
                <AppBarButton Icon="Add"/>
                <AppBarSeparator/>
                <AppBarButton Icon="Edit"/>
                <AppBarButton Icon="Delete"/>
                <AppBarSeparator/>
                <AppBarButton Icon="Save"/>
        </CommandBar>
```

```
    </Pivot.RightHeader>
</Pivot>
```

Pivot interaction

The control features these touch gesture interactions:

- Tapping on a pivot item header navigates to that header's section content.
- Swiping left or right on a pivot item header navigates to the adjacent section.
- Swiping left or right on section content navigates to the adjacent section.

The control comes in two modes:

Stationary

- Pivots are stationary when all pivot headers fit within the allowed space.
- Tapping on a pivot label navigates to the corresponding page, though the pivot itself will not move. The active pivot is highlighted.

Carousel

- Pivots carousel when all pivot headers don't fit within the allowed space.
- Tapping a pivot label navigates to the corresponding page, and the active pivot label will carousel into the first position.
- Pivot items in a carousel loop from last to first pivot section.

> **Tip**
>
> - Avoid using more than 5 headers when using carousel mode, as looping more than 5 can become confusing.
> - Pivot headers should not carousel in a 10ft environment. Set the IsHeaderItemsCarouselEnabled property to `false` if your app will run on Xbox.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that

use **WinUI 2**. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** Pivot class
- ✓ Open the WinUI 2 Gallery app and see the Pivot in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls.

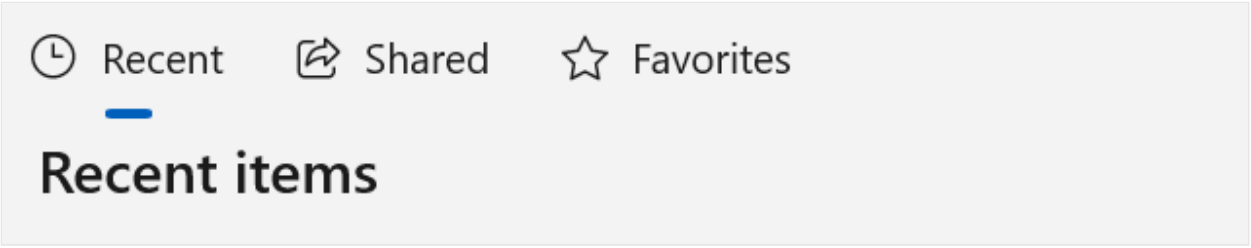
Related topics

- Pivot class
- SelectorBar
- NavigationView
- TabView
- Navigation design basics

Selector bar

Article • 02/26/2025

A selector bar lets a user switch between a small number of different sets or views of data. One item at a time can be selected.



When a user selects an item in the selector bar, you typically change the view by either:

- navigating between different pages in your app.
- changing the data shown in a collection control.

The selector bar is a light-weight control that supports an icon and text. It's intended to present a limited number of options so it does not rearrange items to adapt to different window sizes.

Is this the right control?

Use a SelectorBar when you want to let a user navigate between a limited number of views or pages and only one option can be selected at once.

Some examples include:

- Switching between "Recent," "Shared," and "Favorites" pages, where each page displays a unique list of content.
- Switching between "All," "Unread," "Flagged," and "Urgent" views, where each view displays a uniquely filtered list of email items.

When should a different control be used?

There are some scenarios where another control may be more appropriate to use.

- Use NavigationView when you require consistent, top-level app navigation that adapts to different window sizes.
- Use TabView when the user should be able to open, close, rearrange, or tear off new views of the content.
- Use PipsPager when you need regular pagination of a single data view.

- Use RadioButtons when an option is not selected by default, and context is unrelated to page navigation.

Create a SelectorBar control

✔ **Important APIs**: SelectorBar class, Items property, SelectionChanged event, SelectorBarItem class

Open the WinUI 3 Gallery app and see the SelectorBar in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This XAML creates a basic SelectorBar control with 3 sections of content.

XAML

```
<SelectorBar x:Name="SelectorBar">
    <SelectorBarItem x:Name="SelectorBarItemRecent"
                     Text="Recent" Icon="Clock"/>
    <SelectorBarItem x:Name="SelectorBarItemShared"
                     Text="Shared" Icon="Share"/>
    <SelectorBarItem x:Name="SelectorBarItemFavorites"
                     Text="Favorites" Icon="Favorite"/>
</SelectorBar>
```

This shows how to add a SelectorBarItem in code.

C#

```
SelectorBarItem newItem = new SelectorBarItem()
{
    Text = "New Item",
    Icon = new SymbolIcon(Symbol.Add)
};
selectorBar.Items.Add(newItem);
```

SelectorBar items

You populate the SelectorBar Items collection with SelectorBarItem objects. You can do this directly in XAML or in code. Because it's intended to display a limited number of

options, SelectorBar does not have an `ItemsSource` property for binding to an external collection of items.

Item content

The SelectorBarItem class provides Text and Icon properties that you use to set the content of your selector bar. You can set one or both properties; however, we recommend that you set the `Text` property to make the item more meaningful.

The `Icon` property takes an IconElement, so you can use any of these derived icon types:

- AnimatedIcon
- BitmapIcon
- FontIcon
- IconSourceElement
- ImageIcon
- PathIcon
- SymbolIcon

> **Note**
>
> SelectorBarItem inherits the **Child** property from ItemContainer. You can use this property to set the content, but we don't recommend this. Content set this way will not get the styling and visual states provided by the SelectorBarItem control template.

Item selection

You can use the SelectedItem property to get or set the SelectorBar's active item. This is synchronized with the SelectorBarItem's IsSelected property. If you set either property, the other is updated automatically.

Whenever the SelectorBar gets focus and `SelectedItem` is `null`, `SelectedItem` is automatically set to the first focusable instance in the Items collection, if any exists.

Whenever the selected item is removed from the `Items` collection, the `SelectedItem` property is set to `null`. If `SelectedItem` is set to `null` while the SelectorBar has focus, SelectorBar will have no item selected but keeps focus.

Setting `SelectedItem` to an element that is not currently in the `Items` collection throws an exception.

There is no `SelectedIndex` property, but you can get the index of the `SelectedItem` like this:

C#

```
int currentSelectedIndex =
    selectorBar.Items.IndexOf(selectorBar.SelectedItem);
```

Selection changed

Handle the SelectionChanged event to respond to the users selection and change what is shown to the user. The `SelectionChanged` event is raised when an item is selected in any of these ways:

- UI Automation
- Tab focus (and a new item is selected)
- Left and right navigation within the SelectorBar
- Tapped event through mouse or touch
- Programmatic selection (through either the SelectorBar.SelectedItem property or SelectorBarItem's IsSelected property).

When a user selects an item, you typically change the view by either navigating between different pages in your app or changing the data shown in a collection control. Examples of both are shown here.

Navigate with transition animations

> **Tip**
>
> You can find these examples in the **SelectorBar page of the WinUI Gallery app**. Use the WinUI Gallery app to run and view the full code.

This example demonstrates handling the SelectionChanged event to navigate between different pages. The navigation uses the SlideNavigationTransitionEffect to slide the pages in from the left or right, as appropriate.

XAML

```
<SelectorBar x:Name="SelectorBar2"
             SelectionChanged="SelectorBar2_SelectionChanged">
    <SelectorBarItem x:Name="SelectorBarItemPage1" Text="Page1"
                     IsSelected="True" />
    <SelectorBarItem x:Name="SelectorBarItemPage2" Text="Page2" />
    <SelectorBarItem x:Name="SelectorBarItemPage3" Text="Page3" />
    <SelectorBarItem x:Name="SelectorBarItemPage4" Text="Page4" />
    <SelectorBarItem x:Name="SelectorBarItemPage5" Text="Page5" />
</SelectorBar>

<Frame x:Name="ContentFrame" IsNavigationStackEnabled="False" />
```

C#

```
int previousSelectedIndex = 0;

private void SelectorBar2_SelectionChanged
             (SelectorBar sender, SelectorBarSelectionChangedEventArgs args)
{
    SelectorBarItem selectedItem = sender.SelectedItem;
    int currentSelectedIndex = sender.Items.IndexOf(selectedItem);
    System.Type pageType;

    switch (currentSelectedIndex)
    {
        case 0:
            pageType = typeof(SamplePage1);
            break;
        case 1:
            pageType = typeof(SamplePage2);
            break;
        case 2:
            pageType = typeof(SamplePage3);
            break;
        case 3:
            pageType = typeof(SamplePage4);
            break;
        default:
            pageType = typeof(SamplePage5);
            break;
    }

    var slideNavigationTransitionEffect =
            currentSelectedIndex - previousSelectedIndex > 0 ?
                SlideNavigationTransitionEffect.FromRight :
                SlideNavigationTransitionEffect.FromLeft;

    ContentFrame.Navigate(pageType, null, new
SlideNavigationTransitionInfo()
                            { Effect = slideNavigationTransitionEffect });
```

```
    previousSelectedIndex = currentSelectedIndex;
}
```

Display different collections in an ItemsView

This example shows how to change the data source of an ItemsView when the user selects an option in the SelectorBar.

XAML

```
<SelectorBar x:Name="SelectorBar3"
             SelectionChanged="SelectorBar3_SelectionChanged">
    <SelectorBarItem x:Name="SelectorBarItemPink" Text="Pink"
                     IsSelected="True"/>
    <SelectorBarItem x:Name="SelectorBarItemPlum" Text="Plum"/>
    <SelectorBarItem x:Name="SelectorBarItemPowderBlue" Text="PowderBlue"/>
</SelectorBar>

<ItemsView x:Name="ItemsView3"
           ItemTemplate="{StaticResource ColorsTemplate}"/>
    <ItemsView.Layout>
        <UniformGridLayout/>
    </ItemsView.Layout>
</ItemsView/>
```

C#

```
private void SelectorBar3_SelectionChanged
             (SelectorBar sender, SelectorBarSelectionChangedEventArgs args)
{
    if (sender.SelectedItem == SelectorBarItemPink)
    {
        ItemsView3.ItemsSource = PinkColorCollection;
    }
    else if (sender.SelectedItem == SelectorBarItemPlum)
    {
        ItemsView3.ItemsSource = PlumColorCollection;
    }
    else
    {
        ItemsView3.ItemsSource = PowderBlueColorCollection;
    }
}
```

UWP and WinUI 2

ⓘ Important

The SelectorBar control is not available for UWP and WinUI 2. For alternatives, see NavigationView or TabView.

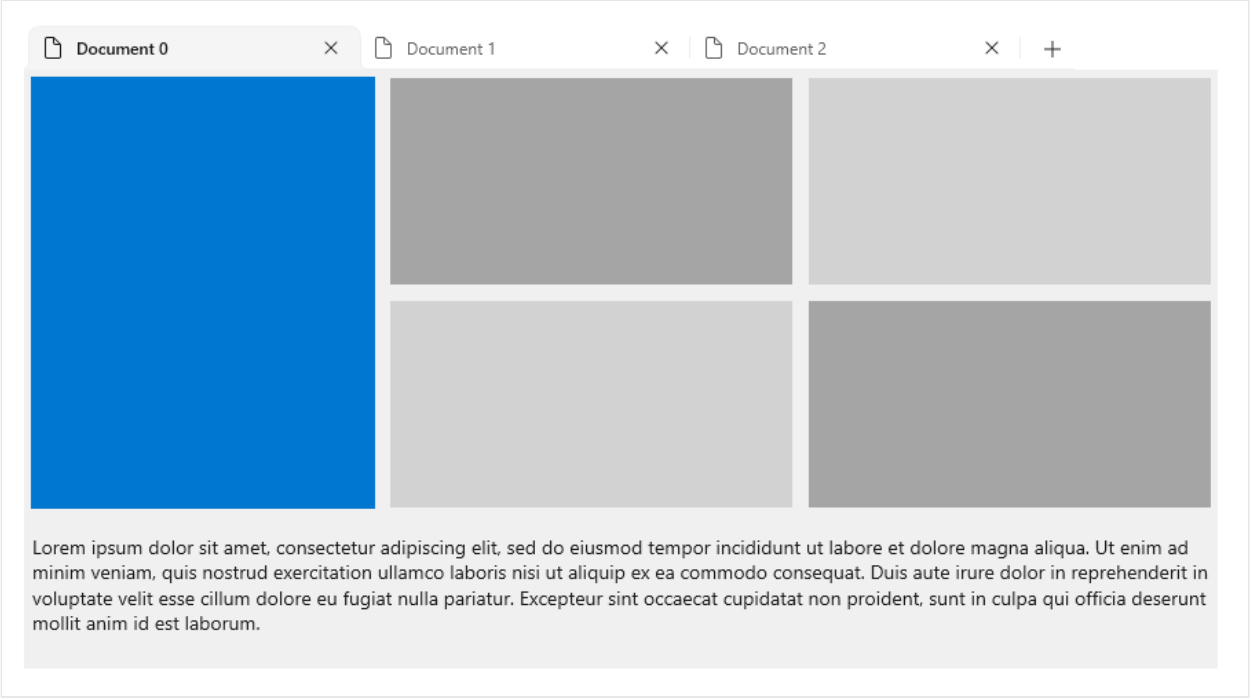
Related topics

- SelectorBar class
- Navigation design basics

Tab view

Article • 01/16/2025

The TabView control is a way to display a set of tabs and their respective content. TabView controls are useful for displaying several pages (or documents) of content while letting a user rearrange, close, or open new tabs.



Is this the right control?

In general, tabbed UIs come in one of two distinct styles that differ in function and appearance:

- **Static tabs** are the sort of tabs often found in settings windows. They contain a set number of pages in a fixed order that usually contain predefined content.
- **Document tabs** are the sort of tabs found in a browser, such as Microsoft Edge. Users can create, remove, and rearrange tabs; move tabs between windows; and change the content of tabs.

By default, TabView is configured to provide document tabs. We recommend TabView when users will be able to:

- Dynamically open, close, or rearrange tabs.
- Open documents or web pages directly into tabs.
- Drag and drop tabs between windows.

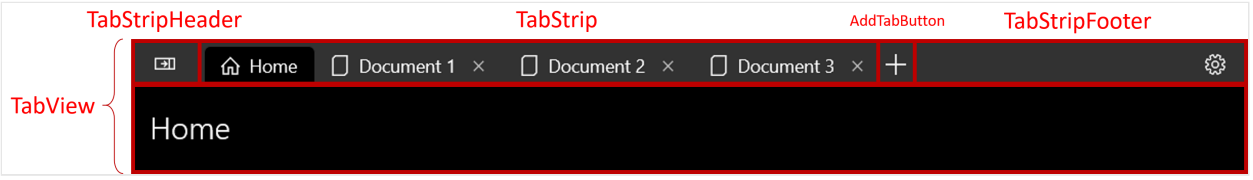
The TabView API does allow configuring the control for static tabs. However, to follow Windows design guidance, and if there are more than a few static navigation items, consider using a NavigationView control.

Anatomy

Tabbed UI is created with a TabView control and one or more TabViewItem controls. The TabView hosts instances of TabViewItem, which represents a single tab and it's content.

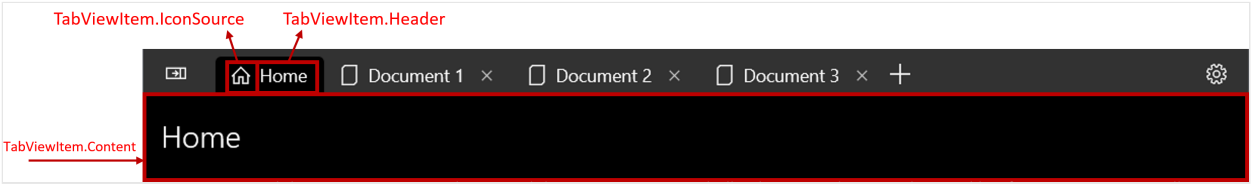
TabView parts

This image shows the parts of the TabView control. The *tab strip* has a header and footer, but unlike a document, the tab strip's header and footer are on the far left and far right of the strip, respectively.



TabViewItem parts

This image shows the parts of the TabViewItem control. Although the content is displayed inside of the TabView control, the content is actually a part of the TabViewItem.



Recommendations

Tab selection

Most users have experience using document tabs simply by using a web browser. When they use document tabs in your app, their experience informs their expectations for how your tabs should behave.

No matter how the user interacts with a set of document tabs, there should always be an active tab. If the user closes the selected tab or breaks the selected tab out into

another window, another tab should become the active tab. TabView attempts to do this automatically selecting the next tab. If you have a good reason that your app should allow a TabView with an unselected tab, the TabView's content area will simply be blank.

Keyboard navigation

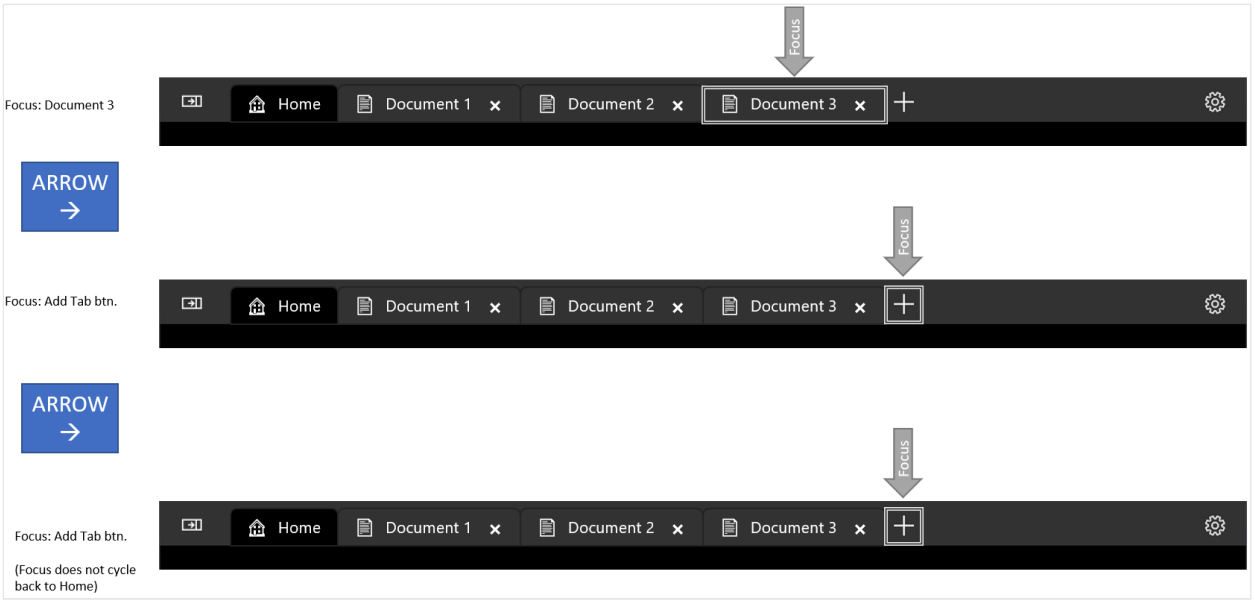
TabView supports many common keyboard navigation scenarios by default. This section explains the built-in functionality, and provides recommendations on additional functionality that might be helpful for some apps.

Tab and cursor key behavior

When focus moves into the *TabStrip* area, the selected TabViewItem gains focus. The user can then use the Left and Right arrow keys to move focus (not selection) to other tabs in the tab strip. Arrow focus is trapped inside the tab strip and the add tab (+) button, if one is present. To move focus out of the tab strip area, the user can press the `Tab` key, which will move focus to the next focusable element.

Move focus via Tab

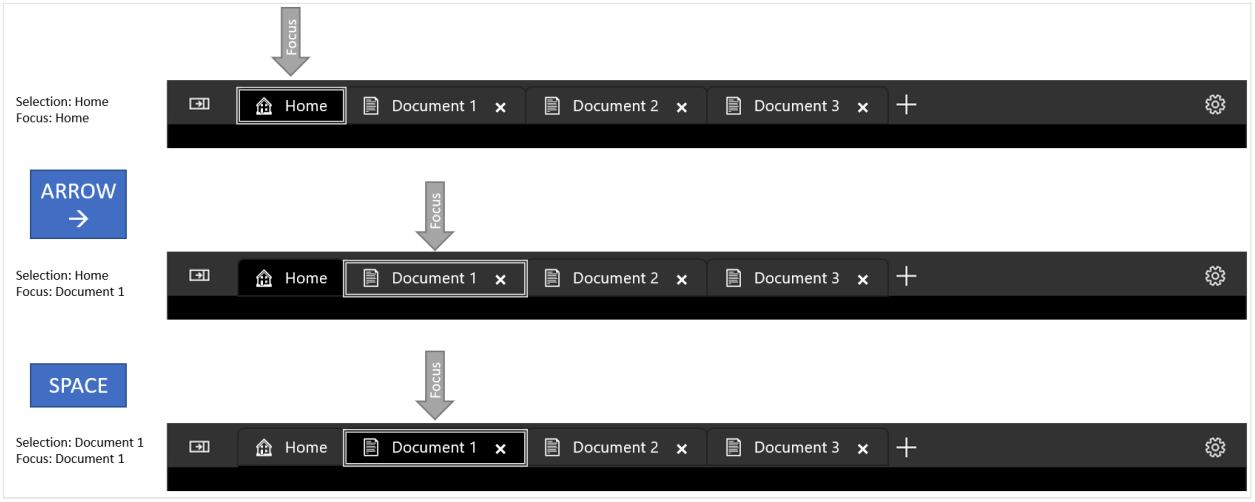
Arrow keys do not cycle focus



Selecting a tab

When a TabViewItem has focus, press `Space` or `Enter` to select that TabViewItem.

Use arrow keys to move focus, then press `Space` to select tab.



Shortcuts for selecting adjacent tabs

Press `Ctrl+Tab` to select the next TabViewItem. Press `Ctrl+Shift+Tab` to select the previous TabViewItem. For these purposes, the tab list is "looped," so selecting the next tab while the last tab is selected will cause the first tab to become selected.

Closing a tab

Press `Ctrl + F4` to raise the TabCloseRequested event. Handle the event and close the tab if appropriate.

> **Tip**
>
> For more information, see Keyboard guidance for developers later in this article.

Create a tab view

✓ **Important APIs**: TabView class, TabViewItem class

Open the WinUI 3 Gallery app and see the TabView in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

The examples in this section show a variety of ways to configure a TabView control.

Tab view items

Each tab in a TabView is represented by a TabViewItem control, which includes both the tab that's shown in the tab strip and the content shown below the tab strip.

Configure a tab

For each TabViewItem, you can set a header and an icon, and specify whether the user can close the tab.

- The Header property is typically set to a string value that provides a descriptive label for the tab. However, the `Header` property can be any object. You can also use the HeaderTemplate property to specify a DataTemplate that defines how bound header data should be displayed.
- Set the IconSource property to specify an icon for the tab.
- By default, the tab shows a *close button* (X). You can set the IsClosable property to `false` to hide the close button and ensure that a user can't close the tab. (If you close tabs in your app code outside of a *close requested* event, you should first check that `IsClosable` is `true`.)

For the TabView, you can configure several options that apply to all tabs.

- By default, the close button is always shown for closable tabs. You can set the CloseButtonOverlayMode property to `OnPointerOver` to change this behavior. In

this case, the selected tab always shows the close button if it is closable, but unselected tabs show the close button only when the tab is closable and the user has their pointer over it.

- You can set the TabWidthMode property to change how tabs are sized. (The `Width` property is ignored on `TabViewItem`.) These are the options in the TabViewWidthMode enumeration:
  - `Equal` - Each tab has the same width. This is the default.
  - `SizeToContent` - Each tab adjusts its width to the content within the tab.
  - `Compact` - Unselected tabs collapse to show only their icon. The selected tab adjusts to display the content within the tab.

Content

The elements displayed in the selected tab are added to the Content property of the TabViewItem. TabViewItem is a ContentControl, so you can add any type of object as content. You can also apply a DataTemplate to the ContentTemplate property. See the ContentControl class for more information.

The examples in this article show a simple case of adding text directly to the `Content` element in XAML. However, real UI is typically more complex. A common way to add complex UI as the content of a tab is to encapsulate it in a UserControl or a Page, and add that as the content of the TabViewItem. This example assumes your app has a XAML UserControl called `PictureSettingsControl`.

XAML

```
<TabViewItem>
    <TabViewItem.Content>
        <local:PictureSettingsControl/>
    </TabViewItem.Content>
</TabViewItem>
```

Static tabs

This example shows a simple TabView with two *static tabs*. Both tab items are added in XAML as content of the TabView.

To make a TabView static, use these settings:

- Set the IsAddTabButtonVisible property to `false` to hide the *add tab* button and prevent the AddTabButtonClick event from being raised.

- Set the CanReorderTabs property to `false` to prevent the user from dragging tabs into a different order.
- On each TabViewItem, set the IsClosable property to false to hide the *tab close* button prevent the user from raising the TabCloseRequested event.

XAML

```
<TabView VerticalAlignment="Stretch"
         IsAddTabButtonVisible="False"
         CanReorderTabs="False">
    <TabViewItem Header="Picture" IsClosable="False">
        <TabViewItem.IconSource>
            <SymbolIconSource Symbol="Pictures"/>
        </TabViewItem.IconSource>
        <TabViewItem.Content>
            <StackPanel Padding="12">
                <TextBlock Text="Picture settings"
                    Style="{ThemeResource TitleTextBlockStyle}"/>
            </StackPanel>
        </TabViewItem.Content>
    </TabViewItem>
    <TabViewItem Header="Sound" IsClosable="False">
        <TabViewItem.IconSource>
            <SymbolIconSource Symbol="Audio"/>
        </TabViewItem.IconSource>
        <TabViewItem.Content>
            <StackPanel Padding="12">
                <TextBlock Text="Sound settings"
                    Style="{ThemeResource TitleTextBlockStyle}"/>
            </StackPanel>
        </TabViewItem.Content>
    </TabViewItem>
</TabView>
```

Document tabs

By default, the TabView is configured for *document tabs*. The user can add new tabs, rearrange tabs, and close tabs. In this configuration, you need to handle the AddTabButtonClick and TabCloseRequested events to enable the functionality.

When tabs are added to a TabView, there might eventually be too many tabs to display in your tab strip. In this case, scroll bumpers will appear that let the user scroll the tab strip left and right to access hidden tabs.

This example creates a simple TabView along with event handlers to support opening and closing tabs. The `TabView_AddTabButtonClick` event handler shows how to add a TabViewItem in code.

XAML

```
<TabView VerticalAlignment="Stretch"
         AddTabButtonClick="TabView_AddTabButtonClick"
         TabCloseRequested="TabView_TabCloseRequested">
    <TabViewItem Header="Home" IsClosable="False">
        <TabViewItem.IconSource>
            <SymbolIconSource Symbol="Home" />
        </TabViewItem.IconSource>
        <TabViewItem.Content>
            <StackPanel Padding="12">
                <TextBlock Text="TabView content"
                           Style="{ThemeResource TitleTextBlockStyle}"/>
            </StackPanel>
        </TabViewItem.Content>
    </TabViewItem>
</TabView>
```

C#

```
// Add a new tab to the TabView.
private void TabView_AddTabButtonClick(TabView sender, object args)
{
    var newTab = new TabViewItem();
    newTab.Header = $"New Document {sender.TabItems.Count}";
    newTab.IconSource = new SymbolIconSource() { Symbol = Symbol.Document };
    newTab.Content = new TextBlock() { Text = $"Content for new tab
{sender.TabItems.Count}.",
                                       Padding = new Thickness(12) };
    sender.TabItems.Add(newTab);
    sender.SelectedItem = newTab;
}

// Remove the requested tab from the TabView.
private void TabView_TabCloseRequested(TabView sender,
                                       TabViewTabCloseRequestedEventArgs
args)
{
    sender.TabItems.Remove(args.Tab);
}
```

Close the window when the last tab is closed

If all tabs in your app are closeable and your app window should close when the last tab is closed, you should also close the window in the TabCloseRequested event handler.

First, in the `App.xaml.cs` file, add a `public static` property that will let you access the `Window` instance from the `Page` that hosts the TabView. (See User interface migration.)

C#

```
public partial class App : Application
{
    // ... code removed.

    // Add this.
    public static Window Window { get { return m_window; } }
    // Update this to make it static.
    private static Window m_window;
}
```

Then, modify the TabCloseRequested event handler to call Window.Close if all tabs have been removed from the TabView.

C#

```
// Remove the requested tab from the TabView.
// If all tabs have been removed, close the Window.
private void TabView_TabCloseRequested(TabView sender,
                                       TabViewTabCloseRequestedEventArgs
args)
{
    sender.TabItems.Remove(args.Tab);

    if (sender.TabItems.Count == 0)
    {
        App.Window.Close();
    }
}
```

> **Note**
>
> This example works for an app with a single window (`MainWindow`). If your app has multiple windows, or you have enabled tab tear-out, you need to track the windows and then find the correct one to close. See the next section for an example of this.

Tab tear-out

Tab tear-out describes what happens when a user drags a tab out of the TabView's tab strip and moves it to another TabView control, typically in a new window.

Starting in Windows App SDK 1.6, TabView has a CanTearOutTabs property that you can set to provide an enhanced experience for dragging tabs out to a new window. When a

user drags a tab out of the tab strip with this option is enabled, a new window is immediately created during the drag, allowing the user to drag it to the edge of the screen to maximize or snap the window in one smooth motion. This implementation also doesn't use drag-and-drop APIs, so it isn't impacted by any limitations in those APIs.

When you set the CanTearOutTabs property to `true`, it causes the *tab tear-out* events to be raised instead of *drag-and-drop* events. To implement tab tear-out, you must handle these events:

- TabTearOutWindowRequested

  This event occurs when a tab is first dragged out of the tab strip. Handle it to create a new Window and TabView that the tab will be moved to.

- TabTearOutRequested

  This event occurs after a new Window has been provided. Handle it to move the torn-out tab from the originating TabView to a TabView in the new window.

- ExternalTornOutTabsDropping

  This event occurs when a torn-out tab is dragged over an existing TabView. Handle it in the TabView that is receiving the torn-out tab to indicate whether or not the tab should be accepted.

- ExternalTornOutTabsDropped

  This event occurs when a torn-out tab is dragged over an existing TabView and the `ExternalTornOutTabsDropping` event has indicated that the drop is allowed. Handle it in the TabView that is receiving the torn-out tab to remove the tab from the originating TabView and insert it into the receiving TabView at the specified index.

These events are not raised when tab tear-out is enabled: TabDragStarting, TabStripDragOver, TabStripDrop, TabDragCompleted, TabDroppedOutside.

> **⊗ Caution**
>
> Tab tear-out is supported in processes running elevated as Administrator.

The following examples show how to implement the event handlers to support tab tear-out.

Set up the TabView

This XAML sets the CanTearOutTabs property to `true` and sets up the tab tear-out event handlers.

XAML

```
<TabView x:Name="tabView"
     CanTearOutTabs="True"
     TabTearOutWindowRequested="TabView_TabTearOutWindowRequested"
     TabTearOutRequested="TabView_TabTearOutRequested"
     ExternalTornOutTabsDropping="TabView_ExternalTornOutTabsDropping"
     ExternalTornOutTabsDropped="TabView_ExternalTornOutTabsDropped">
    <!-- TabView content -->
</TabView>
```

Create and track a new window

Tab tear-out requires that you create and manage new windows in your app.

> **Tip**
>
> The WinUI Gallery app includes a `WindowHelper` class that makes it easier to manage windows in your app. You can copy it from GitHub in the WinUI Gallery repo: WindowHelper.cs. We recommend this helper class to implement tab tear-out. See the TabViewWindowingSamplePage on GitHub to see how it's used.
>
> In this article, helper methods are copied from `WindowHelper.cs`, but are modified and shown inline for readability.

Here, a list for tracking all active windows is created in `App.xaml.cs`. The `OnLaunched` method is updated to track the window after it's created. (This is not needed if you use the `WindowHelper` class.)

C#

```
static public List<Window> ActiveWindows = new List<Window>();

protected override void
OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
{
    m_window = new MainWindow();
    // Track this window.
    ActiveWindows.Add(m_window);
    m_window.Activate();
}
```

When tab tear-out begins, a new window is requested. Here, the variable `tabTearOutWindow` provides access to the new window after it's created. The `CreateWindow` and `TrackWindow` helper methods create a new window and add it to the active window tracking list.

After you create the new window, you need to create a new Page and set it as the content of the window. The new Page must contain a TabView control that you will move the torn-out tab into in the `TabTearOutRequested` event handler.

> **Tip**
>
> In this example, we create a new `MainPage` class, since it contains only an empty TabView (no tabs are added directly in XAML). If `MainPage` includes other UI elements that should not appear in the torn-out window, you can create a separate page that includes only elements you require (including at least a TabView), and create an instance of that page.

Finally, assign the AppWindow.Id of the new window to the `args.`NewWindowId property. This will be used in the TabViewTabTearOutRequestedEventArgs.NewWindowId property so you can access the window from that event handler.

C#

```
private Window? tabTearOutWindow = null;

private void TabView_TabTearOutWindowRequested(TabView sender,
TabViewTabTearOutWindowRequestedEventArgs args)
{
    tabTearOutWindow = CreateWindow();
    tabTearOutWindow.Content = new MainPage();
    // Optional window setup, such as setting the icon or
    // extending content into the title bar happens here.
    args.NewWindowId = tabTearOutWindow.AppWindow.Id;
}

private Window CreateWindow()
{
    Window newWindow = new Window
    {
        SystemBackdrop = new MicaBackdrop()
    };
    newWindow.Title = "Torn Out Window";
    TrackWindow(newWindow);
    return newWindow;
}

private void TrackWindow(Window window)
```

```
{
    window.Closed += (sender, args) => {
        App.ActiveWindows.Remove(window);
    };
    App.ActiveWindows.Add(window);
}
```

Close a window when the last tab is closed

As mentioned earlier, you, might want to close the window when the last tab in a TabView is closed. If your app has multiple windows, you need to find the correct window to close in your list of tracked windows. This example shows how to do that.

C#

```
// Remove the requested tab from the TabView.
// If all tabs have been removed, close the Window.
private void TabView_TabCloseRequested(TabView sender,
TabViewTabCloseRequestedEventArgs args)
{
    sender.TabItems.Remove(args.Tab);

    if (sender.TabItems.Count == 0)
    {
        GetWindowForElement(this)?.Close();
    }
}

public Window? GetWindowForElement(UIElement element)
{
    if (element.XamlRoot != null)
    {
        foreach (Window window in App.ActiveWindows)
        {
            if (element.XamlRoot == window.Content.XamlRoot)
            {
                return window;
            }
        }
    }
    return null;
}
```

Move the tab to the new window

After the new window has been provided, you need to remove the torn-out tab from the `sender` TabView and add it to the TabView in the new window. In this example, the

`public AddTabToTabs` helper method let's you access the TabView in the new `MainPage` instance from the original page instance in order to add the torn-out tab to it.

```
C#

private void TabView_TabTearOutRequested(TabView sender,
TabViewTabTearOutRequestedEventArgs args)
{
    if (tabTearOutWindow?.Content is MainPage newPage
        && args.Tabs.FirstOrDefault() is TabViewItem tab)
    {
        sender.TabItems.Remove(tab);
        newPage.AddTabToTabs(tab);
    }
}

// This method provides access to the TabView from
// another page instance so you can add the torn-out tab.
public void AddTabToTabs(TabViewItem tab)
{
    tabView.TabItems.Add(tab);
}
```

Drag a torn-out tab onto another TabView

When a tab has been torn-out and placed into a new window, as shown in the previous steps, one of two things can happen:

- The user can drop the tab and it remains in the new window. The tear-out process ends here and no more events are raised.
- The user can continue to drag the torn-out tab back onto an existing TabView control. In this case, the process continues and several more events are raised to let you remove the tab from the original TabView and insert the external tab into an existing TabView.

When the tab is dragged over the existing TabView, the ExternalTornOutTabsDropping event it raised. In the event handler, you can determine whether inserting the tab into this TabView is allowed. In most cases, you only need to set the `args.`AllowDrop property to `true`. However, if you need to perform any checks before setting that property, you can do that here. If `AllowDrop` is set to `false`, the tab drag action continues and the ExternalTornOutTabsDropped event is not raised.

```
C#

private void TabView_ExternalTornOutTabsDropping(TabView sender,
                        TabViewExternalTornOutTabsDroppingEventArgs args)
```

```
{
    args.AllowDrop = true;
}
```

If `AllowDrop` is set to `true` in the `ExternalTornOutTabsDropping` event handler, the `ExternalTornOutTabsDropped` event is immediately raised.

> **Note**
>
> The *`Dropped`* in the event name does not directly correspond to the idea of a *drop* action in the drag-and-drop APIs. Here, the user does not need to release the tab to perform a *drop* action. The event is raised while the tab is held over the tab strip, and the code is executed to *drop* the tab into the TabView.

The ExternalTornOutTabsDropped event handler follows the same pattern as the TabTearOutRequested event, but inverted; you need to remove the tab from the originating TabView and insert it into the `sender` TabView.

The `sender` TabView is the control that the tab is being inserted into, so we use the `GetParentTabView` helper method to find the originating tab. It starts with the torn-out TabViewItem and uses VisualTreeHelper to walk up the visual tree and find the TabView the item belongs to. After the TabView is found, the TabViewItem is removed from it's TabItems collection and inserted into the `sender` TabView's `TabItems` collection at the index specified by `args.`DropIndex.

C#

```
private void TabView_ExternalTornOutTabsDropped(TabView sender,
                                TabViewExternalTornOutTabsDroppedEventArgs
args)
{
    if (args.Tabs.FirstOrDefault() is TabViewItem tab)
    {
        GetParentTabView(tab)?.TabItems.Remove(tab);
        sender.TabItems.Insert(args.DropIndex, tab);
    }
}

// Starting with the TabViewItem, walk up the
// visual tree until you get to the TabView.
private TabView? GetParentTabView(TabViewItem tab)
{
    DependencyObject current = tab;
    while (current != null)
    {
        if (current is TabView tabView)
```

```
        {
            return tabView;
        }
        current = VisualTreeHelper.GetParent(current);
    }
    return null;
}
```

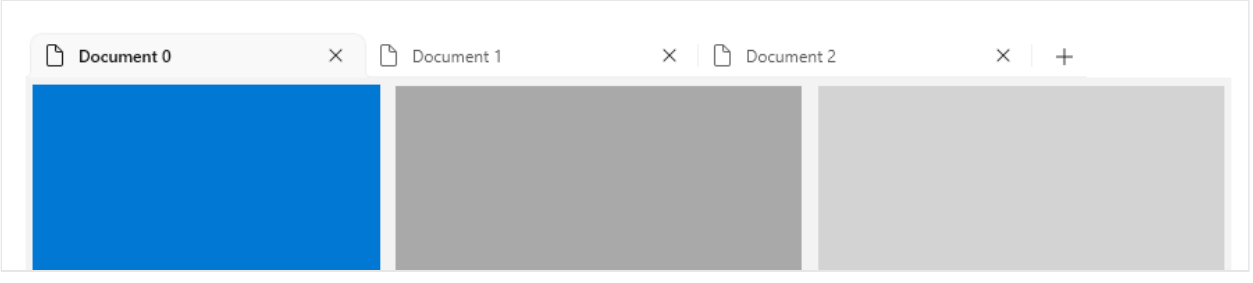
> **Tip**
>
> If you're using the Windows Community Toolkit, you can use the `FindAscendant` helper method in the toolkit's DependencyObjectExtensions instead of `GetParentTabView`.

Display TabView tabs in a window's title bar

Instead of having tabs occupy their own row below a Window's title bar, you can merge the two into the same area. This saves vertical space for your content, and gives your app a modern feel.

Because a user can drag a window by its title bar to reposition the Window, it is important that the title bar is not completely filled with tabs. Therefore, when displaying tabs in a title bar, you must specify a portion of the title bar to be reserved as a draggable area. If you do not specify a draggable region, the entire title bar will be draggable, which prevents your tabs from receiving input events. If your TabView will display in a window's title bar, you should always include a TabStripFooter in your TabView and mark it as a draggable region.

For more information, see Title bar customization



XAML

```
<TabView VerticalAlignment="Stretch">
    <TabViewItem Header="Home" IsClosable="False">
        <TabViewItem.IconSource>
            <SymbolIconSource Symbol="Home" />
        </TabViewItem.IconSource>
```

```
    </TabViewItem>

    <TabView.TabStripFooter>
        <Grid x:Name="CustomDragRegion" Background="Transparent" />
    </TabView.TabStripFooter>
</TabView>
```

C#

```
private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    App.Window.ExtendsContentIntoTitleBar = true;
    App.Window.SetTitleBar(CustomDragRegion);
    CustomDragRegion.MinWidth = 188;
}
```

> **Note**
>
> How you get a reference to the window may vary depending on how you track windows in your app. For more information, see Close the window when the last tab is closed and Create and track a new window in this article.

Keyboard guidance for developers

> **Tip**
>
> For more information about built-in keyboard support, see Keyboard navigation earlier in this article.

Some applications may require more advanced keyboard control. Consider implementing the following shortcuts if they are appropriate for your app.

> **Warning**
>
> If you are adding a TabView to an existing app, you may have already created keyboard shortcuts that map to the key combinations of the recommended TabView keyboard shortcuts. In this case, you will have to consider whether to keep your existing shortcuts or offer an intuitive tab experience for the user.

- `Ctrl + T` should open a new tab. Generally this tab is populated with a predefined document, or is created empty with a simple way to choose its content. If the user

must choose content for a new tab, consider giving input focus to the content selection control.

- `Ctrl + W` should close the selected tab. Remember, TabView will select the next tab automatically.
- `Ctrl + Shift + T` should open recently closed tabs (or more accurately, open new tabs with the same content as recently closed tabs). Start with the most recently closed tab, and move backwards in time for each subsequent time the shortcut is invoked. Note that this will require maintaining a list of recently closed tabs.
- `Ctrl + 1` should select the first tab in the tab list. Likewise, Ctrl + 2 should select the second tab, Ctrl + 3 should select the third, and so on through Ctrl + 8.
- `Ctrl + 9` should select the last tab in the tab list, regardless of how many tabs are in the list.
- If tabs offer more than just the close command (such as duplicating or pinning a tab), use a context menu to show all available actions that can be performed on a tab.

Implement browser-style keyboarding behavior

This example implements a number of the above recommendations on a TabView. Specifically, This example implements `Ctrl + T`, `Ctrl + W`, `Ctrl + 1-8`, and `Ctrl + 9`.

XAML

```
<TabView>
    <!-- ... some tabs ... -->
    <TabView.KeyboardAccelerators>
        <KeyboardAccelerator Key="T" Modifiers="Control"
                             Invoked="NewTabKeyboardAccelerator_Invoked" />
        <KeyboardAccelerator Key="W" Modifiers="Control"

Invoked="CloseSelectedTabKeyboardAccelerator_Invoked" />
        <KeyboardAccelerator Key="Number1" Modifiers="Control"

Invoked="NavigateToNumberedTabKeyboardAccelerator_Invoked" />
        <KeyboardAccelerator Key="Number2" Modifiers="Control"

Invoked="NavigateToNumberedTabKeyboardAccelerator_Invoked" />
        <KeyboardAccelerator Key="Number3" Modifiers="Control"

Invoked="NavigateToNumberedTabKeyboardAccelerator_Invoked" />
        <KeyboardAccelerator Key="Number4" Modifiers="Control"

Invoked="NavigateToNumberedTabKeyboardAccelerator_Invoked" />
        <KeyboardAccelerator Key="Number5" Modifiers="Control"

Invoked="NavigateToNumberedTabKeyboardAccelerator_Invoked" />
        <KeyboardAccelerator Key="Number6" Modifiers="Control"
```

```
Invoked="NavigateToNumberedTabKeyboardAccelerator_Invoked" />
        <KeyboardAccelerator Key="Number7" Modifiers="Control"

Invoked="NavigateToNumberedTabKeyboardAccelerator_Invoked" />
        <KeyboardAccelerator Key="Number8" Modifiers="Control"

Invoked="NavigateToNumberedTabKeyboardAccelerator_Invoked" />
        <KeyboardAccelerator Key="Number9" Modifiers="Control"

Invoked="NavigateToNumberedTabKeyboardAccelerator_Invoked" />
    </TabView.KeyboardAccelerators>
</TabView>
```

C#

```
private void NewTabKeyboardAccelerator_Invoked(KeyboardAccelerator sender,
                                               KeyboardAcceleratorInvokedEventArgs
args)
{
    // Create new tab.
    TabView senderTabView = (TabView)args.Element;
    if (senderTabView is not null)
    {
        // (Click handler defined in previous example.)
        TabView_AddTabButtonClick(senderTabView, new EventArgs());
    }
    args.Handled = true;
}

private void CloseSelectedTabKeyboardAccelerator_Invoked(KeyboardAccelerator
sender,

KeyboardAcceleratorInvokedEventArgs args)
{
    TabView tabView = (TabView)args.Element;
    TabViewItem tab = (TabViewItem)tabView.SelectedItem;
    if (tab is not null)
    {
        CloseSelectedTab(tabView, tab);
    }
    args.Handled = true;
}

private void TabView_TabCloseRequested(TabView sender,
TabViewTabCloseRequestedEventArgs args)
{
    CloseSelectedTab(sender, args.Tab);
}

private void CloseSelectedTab(TabView tabView, TabViewItem tab)
{
```

```
    // Only remove the selected tab if it can be closed.
    if (tab.IsClosable == true)
    {
        tabView.TabItems.Remove(tab);
    }
}


private void
NavigateToNumberedTabKeyboardAccelerator_Invoked(KeyboardAccelerator sender,

KeyboardAcceleratorInvokedEventArgs args)
{
    TabView tabView = (TabView)args.Element;
    int tabToSelect = 0;

    switch (sender.Key)
    {
        case Windows.System.VirtualKey.Number1:
            tabToSelect = 0;
            break;
        case Windows.System.VirtualKey.Number2:
            tabToSelect = 1;
            break;
        case Windows.System.VirtualKey.Number3:
            tabToSelect = 2;
            break;
        case Windows.System.VirtualKey.Number4:
            tabToSelect = 3;
            break;
        case Windows.System.VirtualKey.Number5:
            tabToSelect = 4;
            break;
        case Windows.System.VirtualKey.Number6:
            tabToSelect = 5;
            break;
        case Windows.System.VirtualKey.Number7:
            tabToSelect = 6;
            break;
        case Windows.System.VirtualKey.Number8:
            tabToSelect = 7;
            break;
        case Windows.System.VirtualKey.Number9:
            // Select the last tab
            tabToSelect = tabView.TabItems.Count - 1;
            break;
    }

    // Only select the tab if it is in the list.
    if (tabToSelect < tabView.TabItems.Count)
    {
        tabView.SelectedIndex = tabToSelect;
    }
}
```

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The TabView control for UWP apps is included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in the Microsoft.UI.Xaml.Controls namespace.

Tab tear-out APIs are not included in the WinUI 2 version of TabView.

- ✓ **WinUI 2 Apis:** TabView class, TabViewItem class
- ✓ Open the WinUI 2 Gallery app and see the TabView in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles, templates, and features for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:TabView />
```

Related articles

- List/Details
- NavigationView

Contact card

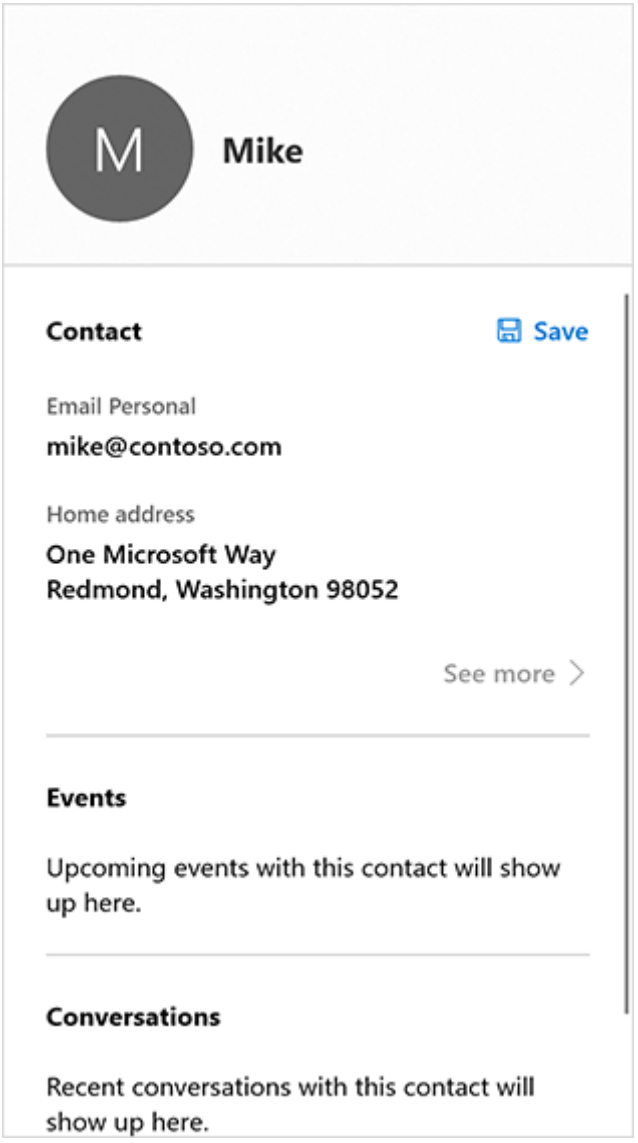
Article • 05/17/2022

The contact card displays contact information, such as the name, phone number, and address, for a Contact (the mechanism Windows uses to represent people and businesses). The contact card also lets the user edit contact info. You can choose to display a compact contact card, or a full contact card that contains additional information.

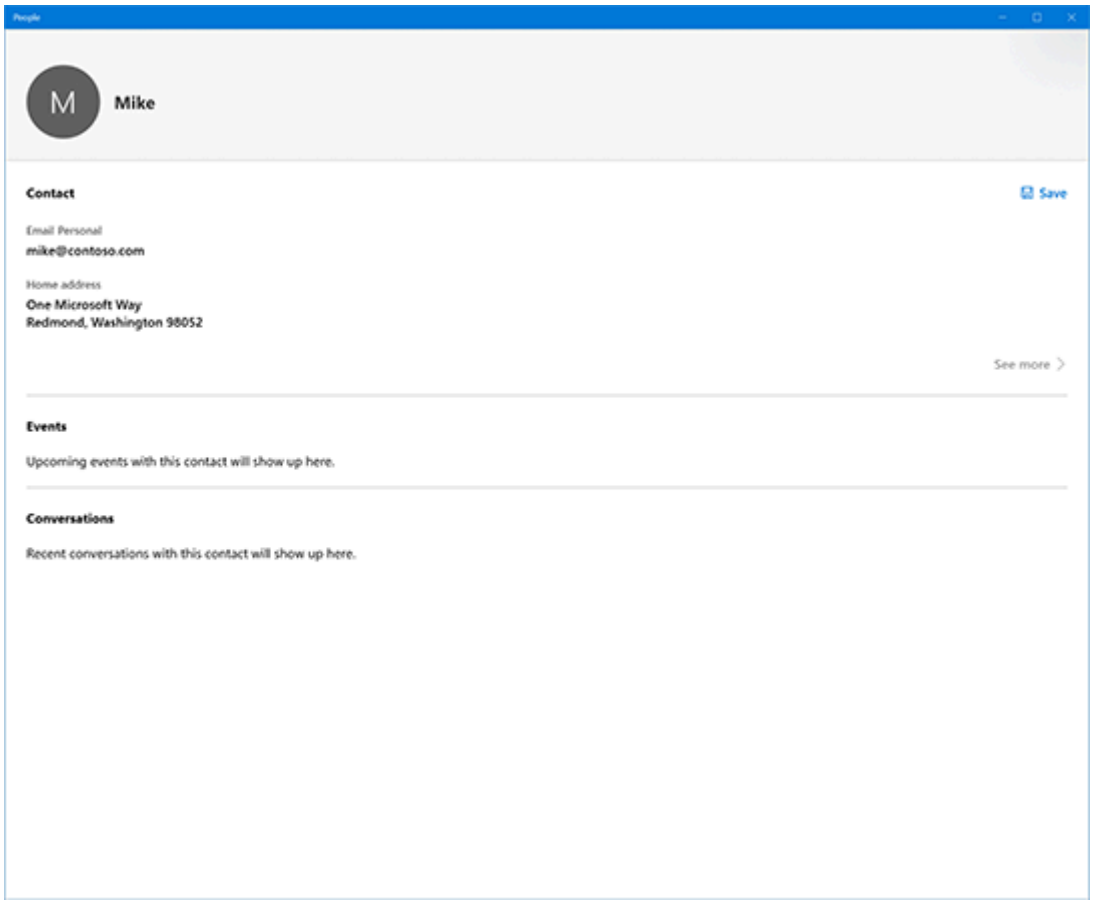
> **Important APIs**: ShowContactCard method, ShowFullContactCard method, IsShowContactCardSupported method, Contact class

There are two ways to display the contact card:

- As a standard contact card that appears in a flyout that is light-dismissable--the contact card disappears when the user clicks outside of it.
- As a full contact card that takes up more space and is not light-dismissable--the user must click **close** to close it.



The standard contact card



The full contact card

Is this the right control?

Use the contact card when you want to display contact info for a contact. If you only want to display the contact's name and picture, use the person picture control.

Show a standard contact card

1. Typically, you show a contact card because the user clicked something: a button or perhaps the person picture control. We don't want to hide the element. To avoid hiding it, we need to create a Rect that describes the location and size of the element.

   Let's create a utility function that does that for us--we'll use it later.

   C#

```
// Gets the rectangle of the element
public static Rect GetElementRectHelper(FrameworkElement element)
{
    // Passing "null" means set to root element.
    GeneralTransform elementTransform =
element.TransformToVisual(null);
    Rect rect = elementTransform.TransformBounds(new Rect(0, 0,
```

```
element.ActualWidth, element.ActualHeight));
    return rect;
}
```

2. Determine whether you can display the contact card by calling the ContactManager.IsShowContactCardSupported method. If it's not supported, display an error message. (This example assumes that you'll be showing the contact card in response to a click event .)

```
C#

// Contact and Contact Managers are existing classes
private void OnUserClickShowContactCard(object sender, RoutedEventArgs
e)
{
    if (ContactManager.IsShowContactCardSupported())
    {
```

3. Use the utility function you created in step 1 to get the bounds of the control that fired the event (so we don't cover it up with the contact card).

```
C#

        Rect selectionRect = GetElementRect((FrameworkElement)sender);
```

4. Get the Contact object you want to display. This example just creates a simple contact, but your code should retrieve an actual contact.

```
C#

            // Retrieve the contact to display
            var contact = new Contact();
            var email = new ContactEmail();
            email.Address = "jsmith@contoso.com";
            contact.Emails.Add(email);
```

5. Show the contact card by calling the ShowContactCard method.

```
C#

        ContactManager.ShowFullContactCard(
            contact, selectionRect, Placement.Default);
    }
}
```

Here's the complete code example:

C#

```
// Gets the rectangle of the element
public static Rect GetElementRect(FrameworkElement element)
{
    // Passing "null" means set to root element.
    GeneralTransform elementTransform = element.TransformToVisual(null);
    Rect rect = elementTransform.TransformBounds(new Rect(0, 0,
element.ActualWidth, element.ActualHeight));
    return rect;
}

// Display a contact in response to an event
private void OnUserClickShowContactCard(object sender, RoutedEventArgs e)
{
    if (ContactManager.IsShowContactCardSupported())
    {
        Rect selectionRect = GetElementRect((FrameworkElement)sender);

        // Retrieve the contact to display
        var contact = new Contact();
        var email = new ContactEmail();
        email.Address = "jsmith@contoso.com";
        contact.Emails.Add(email);

        ContactManager.ShowContactCard(
            contact, selectionRect, Placement.Default);
    }
}
```

Show a full contact card

To show the full contact card, call the ShowFullContactCard method instead of ShowContactCard.

C#

```
private void onUserClickShowContactCard()
{

    Contact contact = new Contact();
    ContactEmail email = new ContactEmail();
    email.Address = "jsmith@hotmail.com";
    contact.Emails.Add(email);


    // Setting up contact options.
```

```
    FullContactCardOptions fullContactCardOptions = new
FullContactCardOptions();

    // Display full contact card on mouse click.
    // Launch the People's App with full contact card
    fullContactCardOptions.DesiredRemainingView =
ViewSizePreference.UseLess;


    // Shows the full contact card by launching the People App.
    ContactManager.ShowFullContactCard(contact, fullContactCardOptions);
}
```

Retrieving "real" contacts

The examples in this article create a simple contact. In a real app, you'd probably want to retrieve an existing contact. For instructions, see the Contacts and calendar article.

Related articles

- Contacts and calendar
- Contact cards sample
- People picture control

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

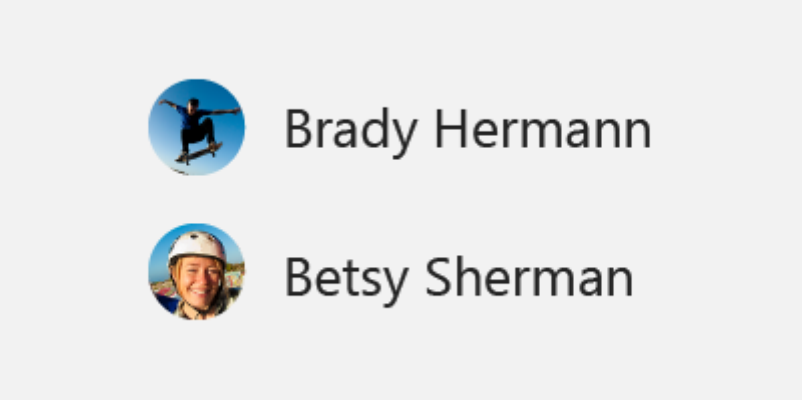
Provide product feedback

Person picture control

Article • 02/26/2025

The person picture control displays the avatar image for a person, if one is available; if not, it displays the person's initials or a generic glyph. You can use the control to display a Contact object, an object that manages a person's contact info, or you can manually provide contact information, such as a display name and profile picture.

Here are two person picture controls accompanied by two text block elements that display the users' names.

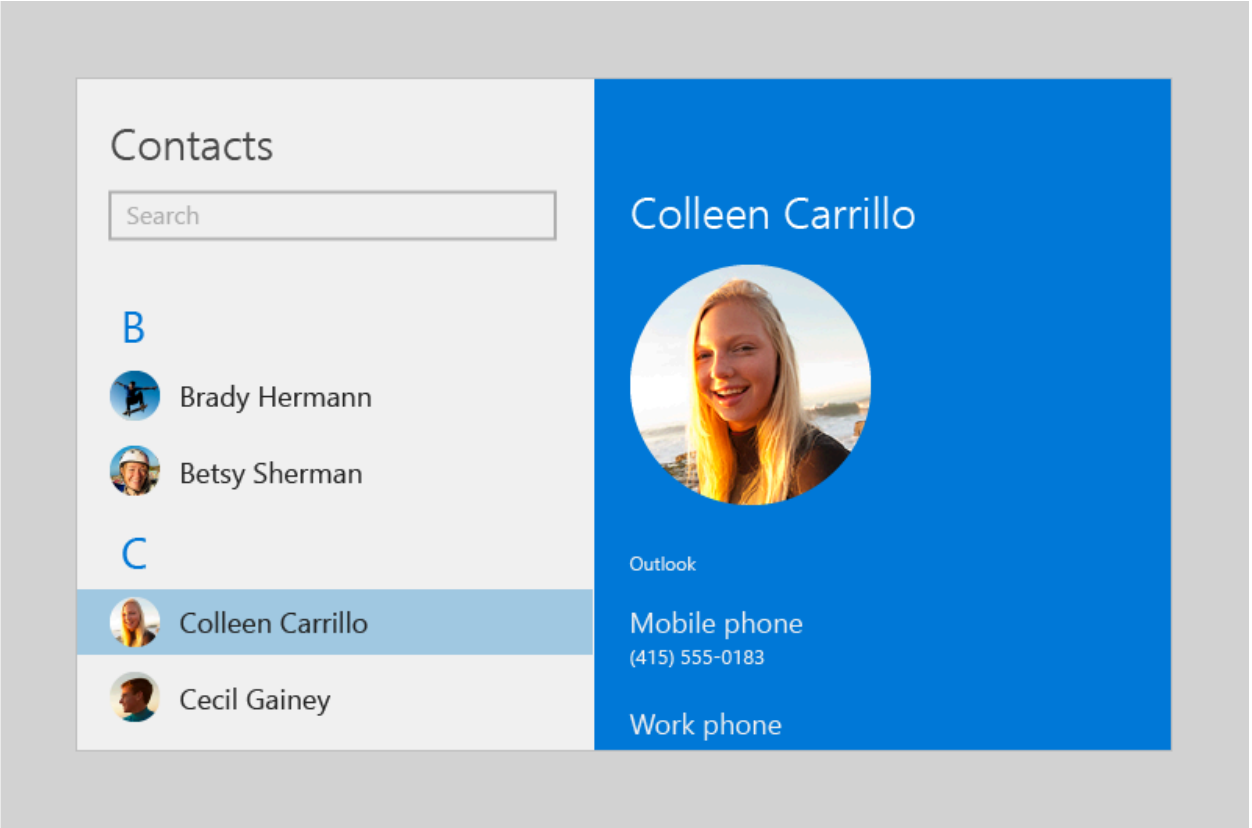


Is this the right control?

Use the person picture when you want to represent a person and their contact information. Here are some examples of when you might use the control:

- To display the current user
- To display contacts in an address book
- To display the sender of a message
- To display a social media contact

The illustration shows the person picture control in a list of contacts:



Create a person picture

✓ **Important APIs:** PersonPicture class

Open the WinUI 3 Gallery app and see PersonPicture in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

To create a person picture, you use the PersonPicture class. This example creates a PersonPicture control and manually provides the person's display name, profile picture, and initials:

XAML

```
<PersonPicture
    DisplayName="Betsy Sherman"
    ProfilePicture="Assets\BetsyShermanProfile.png"
    Initials="BS" />
```

Using the person picture control to display a Contact object

You can use the person picture control to display a Contact object:

XAML

```
<Page
    x:Class="SampleApp.PersonPictureContactExample"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:SampleApp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <StackPanel Background="{ThemeResource
ApplicationPageBackgroundThemeBrush}">

        <PersonPicture
            Contact="{x:Bind CurrentContact, Mode=OneWay}" />

        <Button Click="LoadContactButton_Click">Load contact</Button>
    </StackPanel>
</Page>
```

C#

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.InteropServices.WindowsRuntime;
using Windows.Foundation;
using Windows.Foundation.Collections;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Controls.Primitives;
using Windows.UI.Xaml.Data;
using Windows.UI.Xaml.Input;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;
using Windows.ApplicationModel.Contacts;

namespace SampleApp
{
    public sealed partial class PersonPictureContactExample : Page,
System.ComponentModel.INotifyPropertyChanged
    {
        public PersonPictureContactExample()
        {
```

```
            this.InitializeComponent();
        }

        private Windows.ApplicationModel.Contacts.Contact _currentContact;
        public Windows.ApplicationModel.Contacts.Contact CurrentContact
        {
            get => _currentContact;
            set
            {
                _currentContact = value;
                PropertyChanged?.Invoke(this,
                    new
System.ComponentModel.PropertyChangedEventArgs(nameof(CurrentContact)));
            }

        }
        public event System.ComponentModel.PropertyChangedEventHandler
PropertyChanged;

        public static async
System.Threading.Tasks.Task<Windows.ApplicationModel.Contacts.Contact>
CreateContact()
        {

            var contact = new Windows.ApplicationModel.Contacts.Contact();
            contact.FirstName = "Betsy";
            contact.LastName = "Sherman";

            // Get the app folder where the images are stored.
            var appInstalledFolder =
                Windows.ApplicationModel.Package.Current.InstalledLocation;
            var assets = await appInstalledFolder.GetFolderAsync("Assets");
            var imageFile = await assets.GetFileAsync("betsy.png");
            contact.SourceDisplayPicture = imageFile;

            return contact;
        }

        private async void LoadContactButton_Click(object sender,
RoutedEventArgs e)
        {
            CurrentContact = await CreateContact();
        }
    }
}
```

> **Note**
>
> To keep the code simple, this example creates a new Contact object. In a real app, you'd let the user select a contact or you'd use a ContactManager to query for a

list of contacts. For info on retrieving and managing contacts, see the Contacts and calendar articles.

Determining which info to display

When you provide a Contact object, the person picture control evaluates it to determine which info it can display.

If an image is available, the control displays the first image it finds, in this order:

1. LargeDisplayPicture
2. SmallDisplayPicture
3. Thumbnail

You can change which image is chosen by setting the PreferSmallImage property to true; this gives the SmallDisplayPicture a higher priority than LargeDisplayPicture.

If there isn't an image, the control displays the contact's name or initials; if there's isn't any name data, the control displays contact data, such as an email address or phone number.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The PersonPicture control for UWP apps is included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in both the Windows.UI.Xaml.Controls and Microsoft.UI.Xaml.Controls namespaces.

- ✓ **UWP APIs:** PersonPicture class, Contact class, ContactManager class
- ✓ **WinUI 2 Apis:** PersonPicture class
- ✓ Open the WinUI 2 Gallery app and see PersonPicture in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and

functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles, templates, and features for all controls.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:PersonPicture />
```

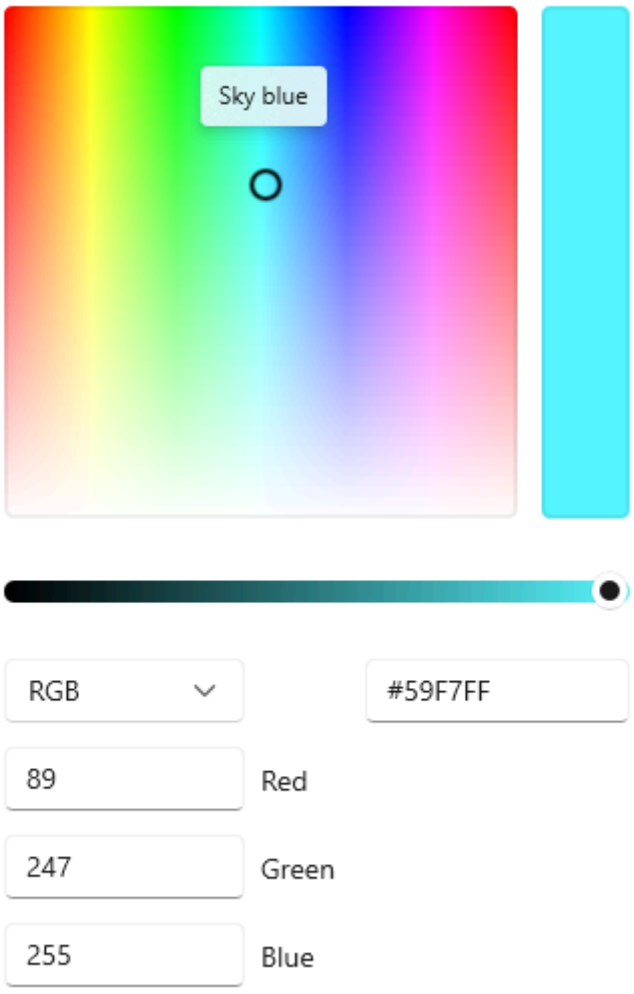
Related articles

- Contacts and calendar
- Contact cards sample

Color picker

Article • 02/26/2025

A color picker is used to browse through and select colors. By default, it lets a user navigate through colors on a color spectrum, or specify a color in either Red-Green-Blue (RGB), Hue-Saturation-Value (HSV), or Hexadecimal text boxes.



Is this the right control?

Use the color picker to let a user select colors in your app. For example, use it to change color settings, such as font colors, background, or app theme colors.

If your app is for drawing or similar tasks using pen, consider using Inking controls along with the color picker.

Recommendations

- Think about what kind of color picking experience is appropriate for your app. Some scenarios may not require granular color picking and would benefit from a simplified picker
- For the most accurate color picking experience, use the square spectrum and ensure it is at least 256x256px, or include the text input fields to let users refine their selected color.
- When used in a flyout, tapping in the spectrum or adjusting the slider alone should not commit the color selection. To commit the selection:
  - Provide commit and cancel buttons to apply or cancel the selection. Hitting the back button or tapping outside of the flyout will dismiss it, and not save the user's selection.
  - Or, commit the selection upon dismissing the flyout, by either tapping outside of the flyout or hitting the back button.

Create a color picker

✔ **Important APIs:** ColorPicker class, Color property, ColorChanged event

Open the WinUI 3 Gallery app and see the ColorPicker in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This example shows how to create a default color picker in XAML.

XAML

```
<ColorPicker x:Name="myColorPicker"/>
```

By default, the color picker shows a preview of the chosen color on the rectangular bar beside the color spectrum. You can use either the ColorChanged event or the Color property to access the selected color and use it in your app. See the following examples for detailed code.

Bind to the chosen color

When the color selection should take effect immediately, you can either use databinding to bind to the Color property, or handle the ColorChanged event to access the selected color in your code.

In this example, you bind the Color property of a SolidColorBrush that's used as the Fill for a Rectangle directly to the color picker's selected color. Any change to the color picker results in a live change to the bound property.

XAML

```
<ColorPicker x:Name="myColorPicker"
             ColorSpectrumShape="Ring"
             IsColorPreviewVisible="False"
             IsColorChannelTextInputVisible="False"
             IsHexInputVisible="False"/>

<Rectangle Height="50" Width="50">
    <Rectangle.Fill>
        <SolidColorBrush Color="{x:Bind myColorPicker.Color, Mode=OneWay}"/>
    </Rectangle.Fill>
</Rectangle>
```

This example uses a simplified color picker with just the circle and the slider, which is a common "casual" color picking experience. When the color change can be seen and happens in real-time on the affected object, you don't need to show the color preview bar. See the *Customize the color picker* section for more info.

Save the chosen color

In some cases, you don't want to apply the color change immediately. For example, when you host a color picker in a flyout, we recommend that you apply the selected color only after the user confirms the selection or closes the flyout. You can also save the selected color value to use later.

In this example, you host a color picker in a Flyout with Confirm and Cancel buttons. When the user confirms their color choice, you save the selected color to use later in your app.

XAML

```
<Page.Resources>
    <Flyout x:Key="myColorPickerFlyout">
        <RelativePanel>
            <ColorPicker x:Name="myColorPicker"
                         IsColorChannelTextInputVisible="False"
                         IsHexInputVisible="False"/>

            <Grid RelativePanel.Below="myColorPicker"
                  RelativePanel.AlignLeftWithPanel="True"
                  RelativePanel.AlignRightWithPanel="True">
                <Grid.ColumnDefinitions>
```

```
                    <ColumnDefinition/>
                    <ColumnDefinition/>
                </Grid.ColumnDefinitions>
                <Button Content="OK" Click="confirmColor_Click"
                        Margin="0,12,2,0" HorizontalAlignment="Stretch"/>
                <Button Content="Cancel" Click="cancelColor_Click"
                        Margin="2,12,0,0" HorizontalAlignment="Stretch"
                        Grid.Column="1"/>
            </Grid>
        </RelativePanel>
    </Flyout>
</Page.Resources>

<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Button x:Name="colorPickerButton"
            Content="Pick a color"
            Flyout="{StaticResource myColorPickerFlyout}"/>
</Grid>
```

C#

```
private Color myColor;

private void confirmColor_Click(object sender, RoutedEventArgs e)
{
    // Assign the selected color to a variable to use outside the popup.
    myColor = myColorPicker.Color;

    // Close the Flyout.
    colorPickerButton.Flyout.Hide();
}

private void cancelColor_Click(object sender, RoutedEventArgs e)
{
    // Close the Flyout.
    colorPickerButton.Flyout.Hide();
}
```

Configure the color picker

Not all fields are necessary to let a user pick a color, so the color picker is flexible. It provides a variety of options that let you configure the control to fit your needs.

For example, when the user doesn't need precise control, like picking a highlighter color in a note taking app, you can use a simplified UI. You can hide the text entry fields and change the color spectrum to a circle.

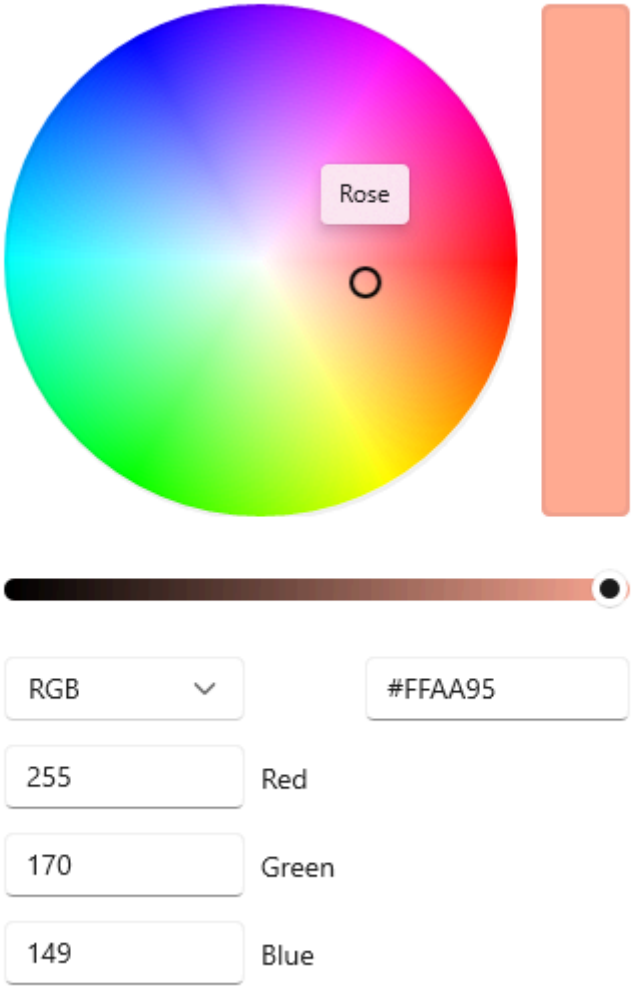
When the user does need precise control, like in a graphic design app, you can show both sliders and text entry fields for each aspect of the color.

Show the circle spectrum

This example shows how to use the ColorSpectrumShape property to configure the color picker to use a circular spectrum instead of the default square.

XAML

```
<ColorPicker x:Name="myColorPicker"
             ColorSpectrumShape="Ring"/>
```



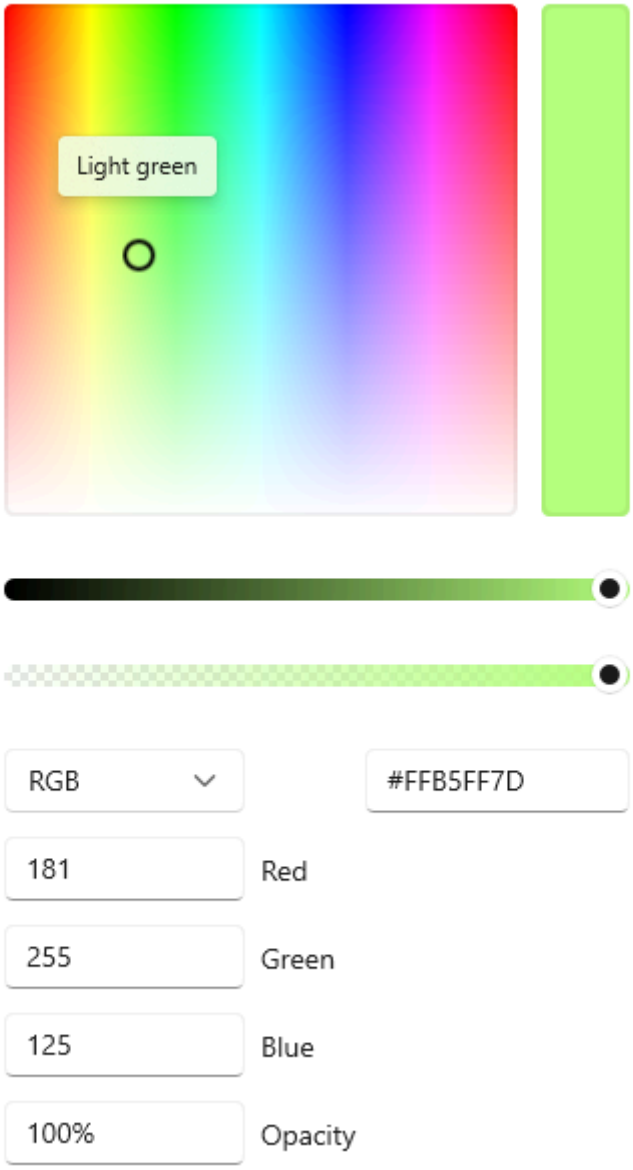
When you must choose between the square and circle color spectrum, a primary consideration is accuracy. A user has more control when they select a specific color using a square because more of the color gamut is shown. You should consider the circle spectrum as more of the "casual" color choosing experience.

Show the alpha channel

In this example, you enable an opacity slider and textbox on the color picker.

XAML

```
<ColorPicker x:Name="myColorPicker"
             IsAlphaEnabled="True"/>
```



Show a simple picker

This example shows how to configure the color picker with a simple UI for "casual" use. You show the circular spectrum and hide the default text input boxes. When the color change can be seen and happens in real-time on the affected object, you don't need to show the color preview bar. Otherwise, you should leave the color preview visible.

XAML

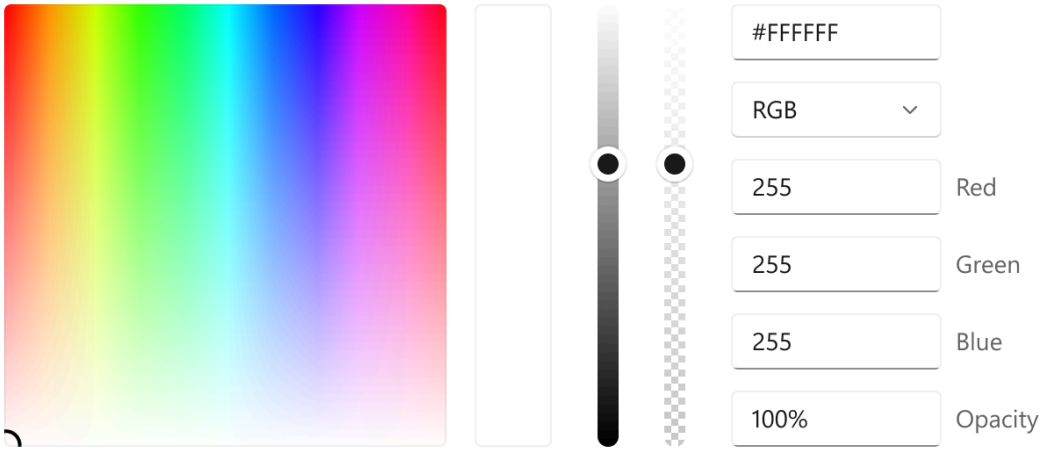
```
<ColorPicker x:Name="myColorPicker"
             ColorSpectrumShape="Ring"
             IsColorPreviewVisible="False"
             IsColorChannelTextInputVisible="False"
             IsHexInputVisible="False"/>
```

Specify the layout direction

Use the Orientation property to specify whether the ColorPicker should align vertically or horizontally. This affects the location of the editing controls relative to the color spectrum. By default, the orientation is Vertical.

XAML

```
<ColorPicker IsAlphaEnabled="True"
             Orientation="Horizontal"/>
```



> **Note**
>
> If the Orientation is set to Horizontal, the ColorPicker does **not** apply the IsMoreButtonVisible property.

Show or hide additional features

This table shows all the options you can use to configure the ColorPicker control.

Expand table

Feature	Properties
Color spectrum	IsColorSpectrumVisible, ColorSpectrumShape, ColorSpectrumComponents
Color preview	IsColorPreviewVisible
Color values	IsColorSliderVisible, IsColorChannelTextInputVisible
Opacity values	IsAlphaEnabled, IsAlphaSliderVisible, IsAlphaTextInputVisible
Hex values	IsHexInputVisible

> **Note**
>
> IsAlphaEnabled must be **true** in order to show the opacity textbox and slider. The visibility of the input controls can then be modified using IsAlphaTextInputVisible and IsAlphaSliderVisible properties. See the API documentation for details.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The ColorPicker control for UWP apps is included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in both the Windows.UI.Xaml.Controls and Microsoft.UI.Xaml.Controls namespaces.

- ✓ **UWP APIs:** ColorPicker class, Color property, ColorChanged event
- ✓ **WinUI 2 Apis:** ColorPicker class, Color property, ColorChanged event
- ✓ Open the WinUI 2 Gallery app and see the ColorPicker in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles, templates, and features for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:ColorPicker />
```

Related articles

- Pen and stylus interactions in Windows apps
- Inking

Calendar, date, and time controls

Article • 06/09/2022

Date and time controls give you standard, localized ways to let a user view and set date and time values in your app. This article provides design guidelines and helps you pick the right control.

- ✓ Open the WinUI 2 Gallery app and see Date and Time controls in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

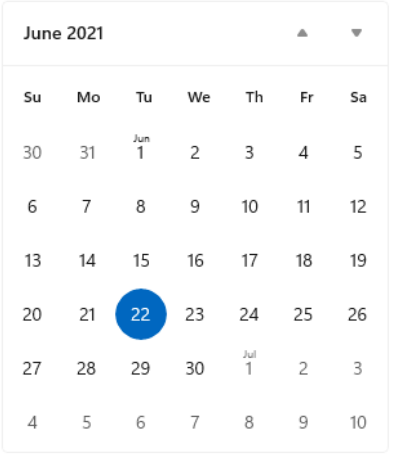
Open the WinUI 3 Gallery app and see Date and Time controls in action

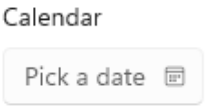
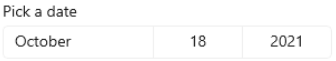
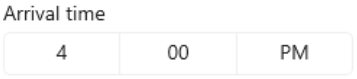
> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Which date or time control should you use?

There are four date and time controls to choose from; the control you use depends on your scenario. Use this info to pick the right control to use in your app.

Expand table

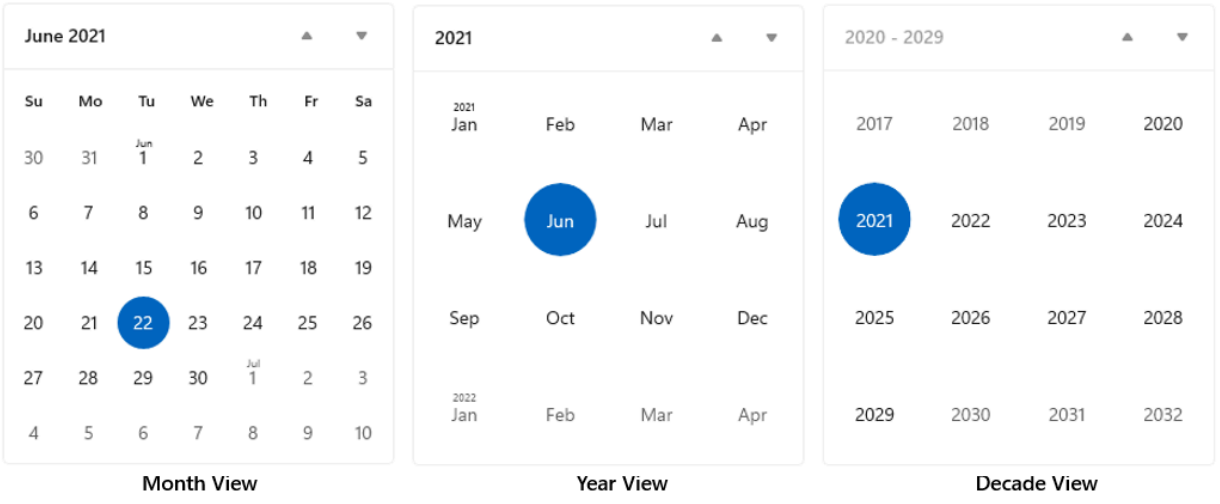
Control	Example	Description
Calendar view		Use to pick a single date or a range of dates from an always visible calendar.

Control	Example	Description
Calendar date picker		Use to pick a single date from a contextual calendar.
Date picker		Use to pick a single known date when contextual info isn't important.
Time picker		Use to pick a single time value.

Calendar view

CalendarView lets a user view and interact with a calendar that they can navigate by month, year, or decade. A user can select a single date or a range of dates. It doesn't have a picker surface and the calendar is always visible.

The calendar view is made up of 3 separate views: the month view, year view, and decade view. By default, it starts with the month view open, but you can specify any view as the startup view.



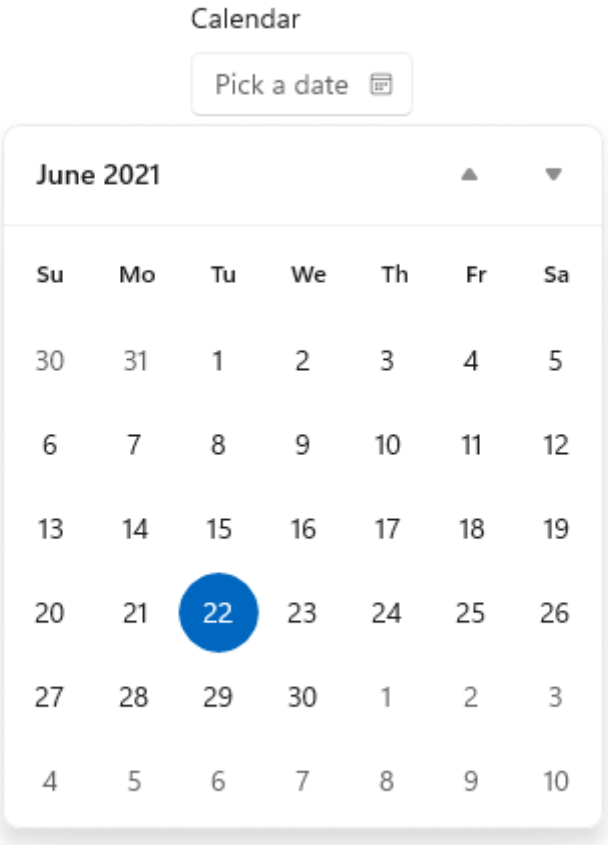
Month View **Year View** **Decade View**

- If you need to let a user select multiple dates, you must use a **CalendarView**.
- If you need to let a user pick only a single date and don't need a calendar to be always visible, consider using a **CalendarDatePicker** or **DatePicker** control.

Calendar date picker

CalendarDatePicker is a drop down control that's optimized for picking a single date from a calendar view where contextual information like the day of the week or fullness of the calendar is important. You can modify the calendar to provide additional context or to limit available dates.

The entry point displays placeholder text if a date has not been set; otherwise, it displays the chosen date. When the user selects the entry point, a calendar view expands for the user to make a date selection. The calendar view overlays other UI; it doesn't push other UI out of the way.



- Use a calendar date picker for things like choosing an appointment or departure date.

Date picker

The **DatePicker** control provides a standardized way to choose a specific date.

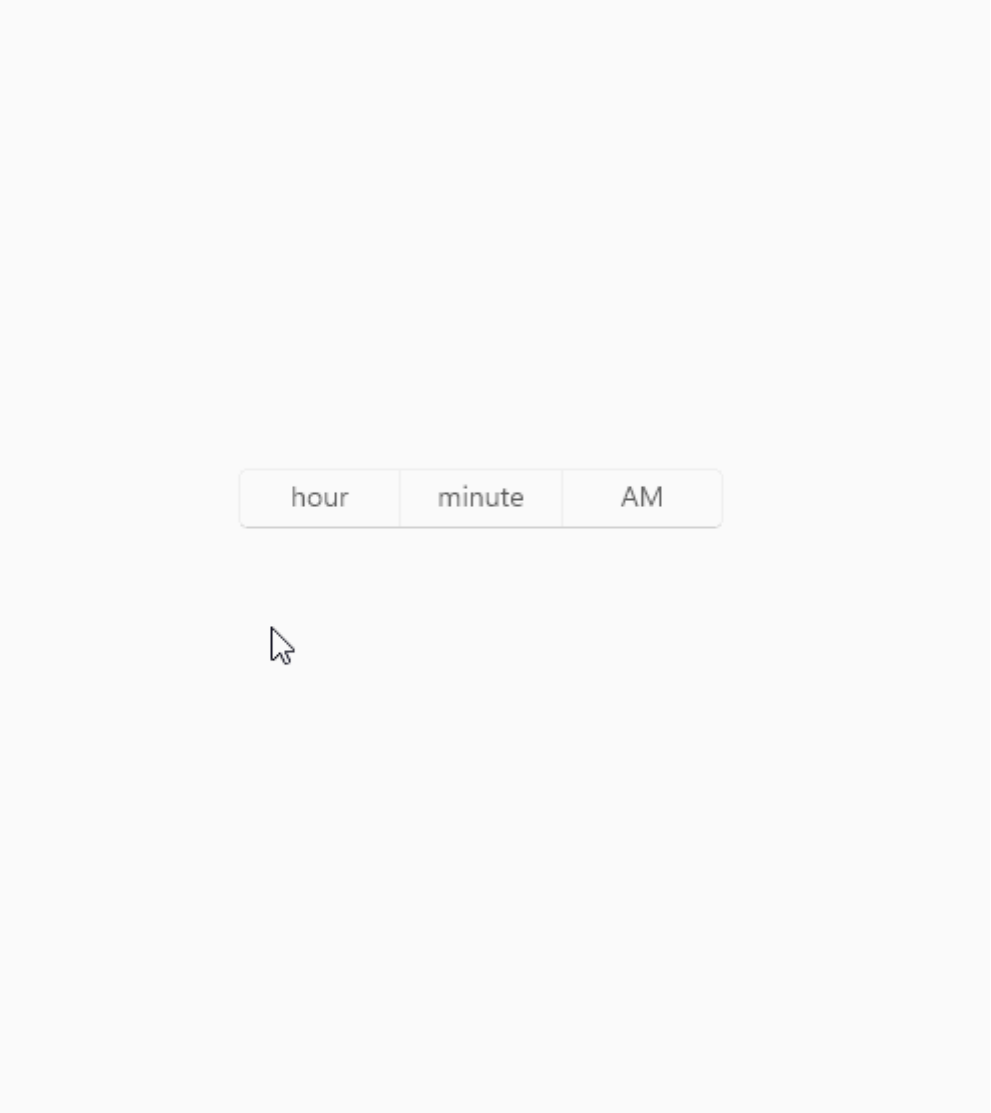
The entry point displays the chosen date, and when the user selects the entry point, a picker surface expands vertically from the middle for the user to make a selection. The date picker overlays other UI; it doesn't push other UI out of the way.

- Use a date picker to let a user pick a known date, such as a date of birth, where the context of the calendar is not important.

Time picker

The **TimePicker** is used to select a single time value for things like appointments or a departure time. It's a static display that is set by the user or in code, but it doesn't update to display the current time.

The entry point displays the chosen time, and when the user selects the entry point, a picker surface expands vertically from the middle for the user to make a selection. The time picker overlays other UI; it doesn't push other UI out of the way.



- Use a time picker to let a user pick a single time value.

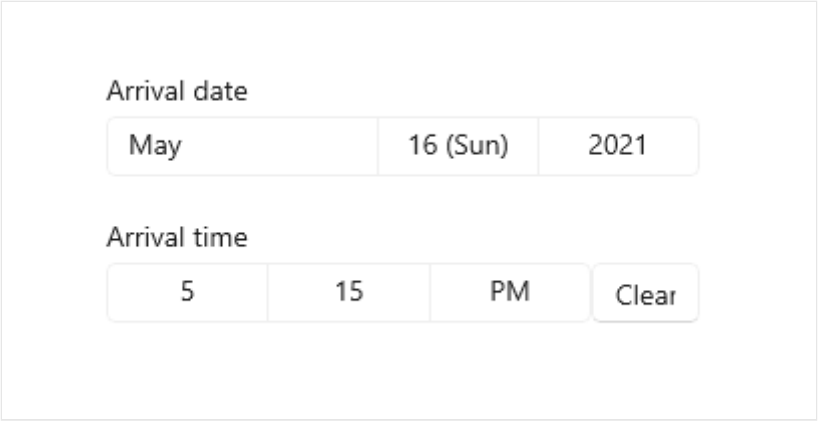
Create a date or time control

See these articles for info and examples specific to each date and time control.

- Calendar view
- Calendar date picker
- Date picker
- Time Picker

Use a date picker and time picker together

This example shows how to use a `DatePicker` and `TimePicker` together to let a user select their arrival date and time. You handle the `SelectedDateChanged` and `SelectedTimeChanged` events to update a single DateTime instance named `arrivalDateTime`. The user can also clear the date and time pickers after they have been set.



XAML

```
<StackPanel>
    <DatePicker x:Name="arrivalDatePicker" Header="Arrival date"
                DayFormat="{}{day.integer} ({dayofweek.abbreviated})"

SelectedDateChanged="ArrivalDatePicker_SelectedDateChanged"/>
    <StackPanel Orientation="Horizontal">
        <TimePicker x:Name="arrivalTimePicker" Header="Arrival time"
                MinuteIncrement="15"

SelectedTimeChanged="ArrivalTimePicker_SelectedTimeChanged"/>
        <Button Content="Clear" Click="ClearDateButton_Click"
                VerticalAlignment="Bottom" Height="30" Width="54"/>
    </StackPanel>
    <TextBlock x:Name="arrivalText" Margin="0,12"/>
</StackPanel>
```

C#

```
public sealed partial class MainPage : Page
{
    DateTime arrivalDateTime;

    public MainPage()
    {
        this.InitializeComponent();

        // Set minimum to the current year and maximum to five years from
now.
        arrivalDatePicker.MinYear = DateTimeOffset.Now;
        arrivalDatePicker.MaxYear = DateTimeOffset.Now.AddYears(5);
    }

    private void ArrivalTimePicker_SelectedTimeChanged(TimePicker sender,
TimePickerSelectedValueChangedEventArgs args)
    {
        if (arrivalTimePicker.SelectedTime != null)
        {
            arrivalDateTime = new DateTime(arrivalDateTime.Year,
arrivalDateTime.Month, arrivalDateTime.Day,
```

```
                                                args.NewTime.Value.Hours,
args.NewTime.Value.Minutes, args.NewTime.Value.Seconds);
            }
            arrivalText.Text = arrivalDateTime.ToString();
        }

        private void ArrivalDatePicker_SelectedDateChanged(DatePicker sender,
DatePickerSelectedValueChangedEventArgs args)
        {
            if (arrivalDatePicker.SelectedDate != null)
            {
                if
(VerifyDateIsFuture((DateTimeOffset)arrivalDatePicker.SelectedDate) == true)
                {
                    arrivalDateTime = new DateTime(args.NewDate.Value.Year,
args.NewDate.Value.Month, args.NewDate.Value.Day,
                                                   arrivalDateTime.Hour,
arrivalDateTime.Minute, arrivalDateTime.Second);
                    arrivalText.Text = arrivalDateTime.ToString();
                }
                else
                {
                    arrivalDatePicker.SelectedDate = null;
                    arrivalText.Text = "Arrival date must be later than today.";
                }
            }
        }

        private bool VerifyDateIsFuture(DateTimeOffset date)
        {
            if (date > DateTimeOffset.Now)
            {
                return true;
            }
            return false;
        }

        private void ClearDateButton_Click(object sender, RoutedEventArgs e)
        {
            arrivalDateTime = new DateTime();
            arrivalDatePicker.SelectedDate = null;
            arrivalTimePicker.SelectedTime = null;
            arrivalText.Text = string.Empty;
        }
    }
```

Globalization

The XAML date controls support each of the calendar systems supported by Windows. These calendars are specified in the Windows.Globalization.CalendarIdentifiers class.

Each control uses the correct calendar for your app's default language, or you can set the **CalendarIdentifier** property to use a specific calendar system.

The time picker control supports each of the clock systems specified in the Windows.Globalization.ClockIdentifiers class. You can set the ClockIdentifier property to use either a 12-hour clock or 24-hour clock. The type of the property is String, but you must use values that correspond to the static string properties of the ClockIdentifiers class. These are: TwelveHour (the string "12HourClock")and TwentyFourHour (the string "24HourClock"). "12HourClock" is the default value.

DateTime and Calendar values

The date objects used in the XAML date and time controls have a different representation depending on your programming language.

- C# and Visual Basic use the System.DateTimeOffset structure that is part of .NET.
- C++/CX uses the Windows::Foundation::DateTime structure.

A related concept is the Calendar class, which influences how dates are interpreted in context. All Windows Runtime apps can use the Windows.Globalization.Calendar class. C# and Visual Basic apps can alternatively use the System.Globalization.Calendar class, which has very similar functionality. (Windows Runtime apps can use the base .NET Calendar class but not the specific implementations; for example, GregorianCalendar.)

.NET also supports a type named DateTime, which is implicitly convertible to a DateTimeOffset. So you might see a "DateTime" type being used in .NET code that's used to set values that are really DateTimeOffset. For more info on the difference between DateTime and DateTimeOffset, see Remarks in the DateTimeOffset class.

> ⓘ **Note**
>
> Properties that take date objects can't be set as a XAML attribute string, because the Windows Runtime XAML parser doesn't have a conversion logic for converting strings to dates as DateTime/DateTimeOffset objects. You typically set these values in code. Another possible technique is to define a date that's available as a data object or in the data context, then set the property as a XAML attribute that references a **{Binding} markup extension** expression that can access the date as data.

Get the sample code

- WinUI Gallery sample
- Calendar sample
- Date and time formatting sample

Related topics

For developers (XAML)

- CalendarView class
- CalendarDatePicker class
- DatePicker class
- TimePicker class

Calendar date picker

Article • 02/26/2025

The calendar date picker is a drop down control that's optimized for picking a single date from a calendar view where contextual information like the day of the week or fullness of the calendar is important. You can modify the calendar to provide additional context or to limit available dates.

Is this the right control?

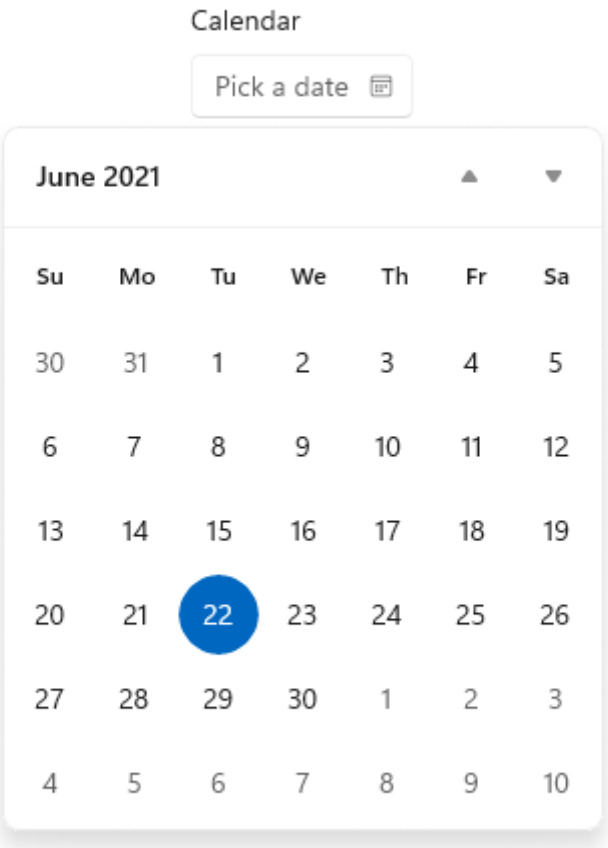
Use a **calendar date picker** to let a user pick a single date from a contextual calendar view. Use it for things like choosing an appointment or departure date.

To let a user pick a known date, such as a date of birth, where the context of the calendar is not important, consider using a date picker.

For more info about choosing the right control, see the Date and time controls article.

Examples

The entry point displays placeholder text if a date has not been set; otherwise, it displays the chosen date. When the user selects the entry point, a calendar view expands for the user to make a date selection. The calendar view overlays other UI; it doesn't push other UI out of the way.



Create a calendar date picker

✓ **Important APIs:** CalendarDatePicker class, Date property, DateChanged event

Open the WinUI 3 Gallery app and see the CalendarDatePicker in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

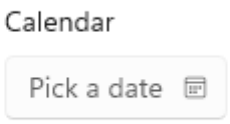
XAML

```
<CalendarDatePicker x:Name="arrivalCalendarDatePicker" Header="Calendar"/>
```

C#

```
CalendarDatePicker arrivalCalendarDatePicker = new CalendarDatePicker();
arrivalCalendarDatePicker.Header = "Calendar";
```

The resulting calendar date picker looks like this:



The calendar date picker has an internal CalendarView for picking a date. A subset of CalendarView properties, like IsTodayHighlighted and FirstDayOfWeek, exist on CalendarDatePicker and are forwarded to the internal CalendarView to let you modify it.

However, you can't change the SelectionMode of the internal CalendarView to allow multiple selection. If you need to let a user pick multiple dates or need a calendar to be always visible, consider using a calendar view instead of a calendar date picker. See the Calendar view article for more info on how you can modify the calendar display.

Selecting dates

Use the Date property to get or set the selected date. By default, the Date property is **null**. When a user selects a date in the calendar view, this property is updated. A user can clear the date by clicking the selected date in the calendar view to deselect it.

You can set the date in your code like this.

```
C#

myCalendarDatePicker.Date = new DateTime(1977, 1, 5);
```

When you set the Date in code, the value is constrained by the MinDate and MaxDate properties.

- If **Date** is smaller than **MinDate**, the value is set to **MinDate**.
- If **Date** is greater than **MaxDate**, the value is set to **MaxDate**.

You can handle the DateChanged event to be notified when the Date value has changed.

> **Note**
>
> For important info about date values, see DateTime and Calendar values in the Date and time controls article.

Setting a header and placeholder text

You can add a Header (or label) and PlaceholderText (or watermark) to the calendar date picker to give the user an indication of what it's used for. To customize the look of the header, you can set the HeaderTemplate property instead of Header.

The default placeholder text is "select a date". You can remove this by setting the PlaceholderText property to an empty string, or you can provide custom text as shown here.

XAML

```
<CalendarDatePicker x:Name="arrivalCalendarDatePicker" Header="Arrival date"
                    PlaceholderText="Choose your arrival date"/>
```

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** CalendarDatePicker class, Date property, DateChanged event
- ✓ Open the WinUI 2 Gallery app and see the CalendarDatePicker in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related articles

- Date and time controls
- Calendar view
- Date picker
- Time picker

Calendar view

Article • 02/26/2025

A calendar view lets a user view and interact with a calendar that they can navigate by month, year, or decade. A user can select a single date or a range of dates. It doesn't have a picker surface and the calendar is always visible.

Is this the right control?

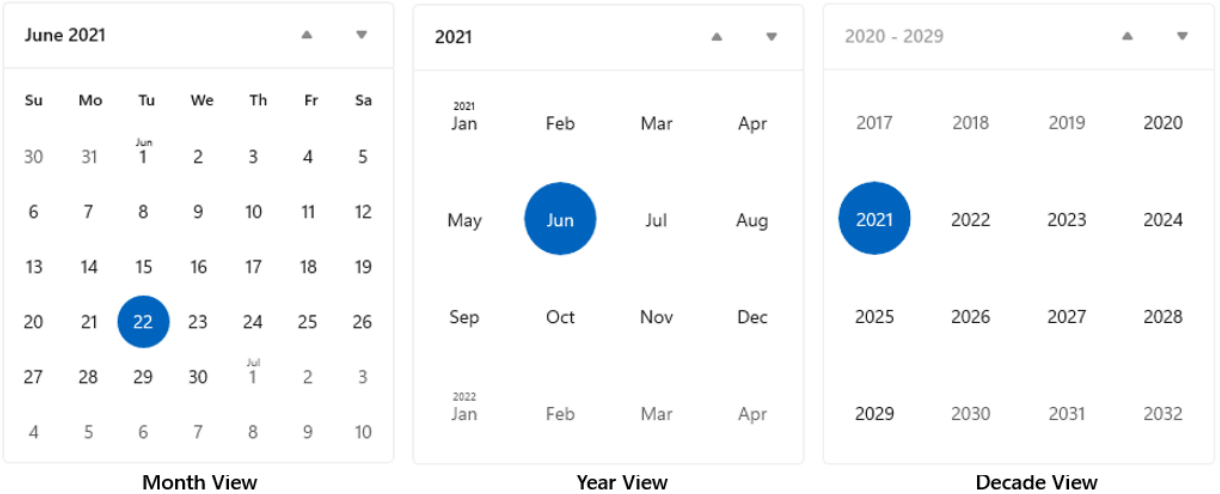
Use a calendar view to let a user pick a single date or a range of dates from an always visible calendar.

If you need to let a user select multiple dates at one time, you must use a calendar view. If you need to let a user pick only a single date and don't need a calendar to be always visible, consider using a calendar date picker or date picker control.

For more info about choosing the right control, see the Date and time controls article.

Examples

The calendar view is made up of 3 separate views: the month view, year view, and decade view. By default, it starts with the month view open. You can specify a startup view by setting the DisplayMode property.



Month View | Year View | Decade View

Users click the header in the month view to open the year view, and click the header in the year view to open the decade view. Users pick a year in the decade view to return to the year view, and pick a month in the year view to return to the month view. The two arrows to the side of the header navigate forward or backward by month, by year, or by decade.

Create a calendar view

✔ **Important APIs:** CalendarView class, SelectedDatesChanged event

Open the WinUI 3 Gallery app and see the CalendarView in action

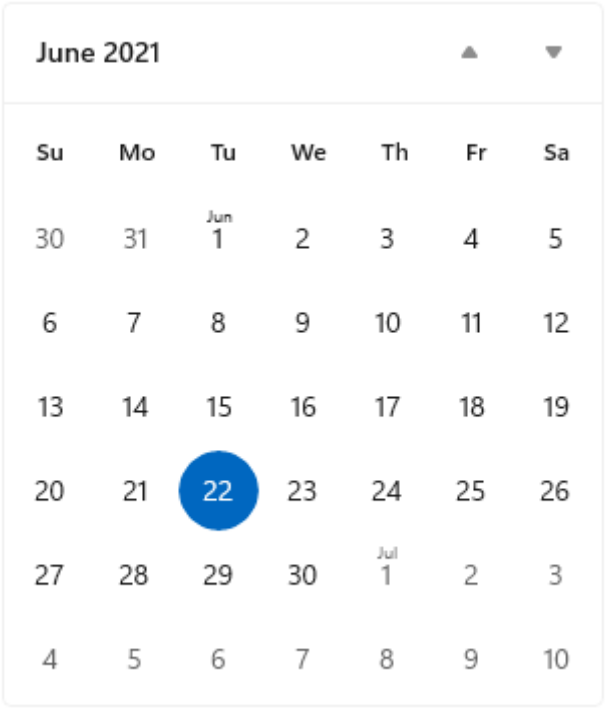
> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This example shows how to create a simple calendar view.

XAML

```
<CalendarView/>
```

The resulting calendar view looks like this:



Selecting dates

By default, the SelectionMode property is set to **Single**. This lets a user pick a single date in the calendar. Set SelectionMode to **None** to disable date selection.

Set SelectionMode to **Multiple** to let a user select multiple dates. You can select multiple dates programmatically by adding DateTime/DateTimeOffset objects to the

SelectedDates collection, as shown here:

```
C#

calendarView1.SelectedDates.Add(DateTimeOffset.Now);
calendarView1.SelectedDates.Add(new DateTime(1977, 1, 5));
```

A user can deselect a selected date by clicking or tapping it in the calendar grid.

You can handle the SelectedDatesChanged event to be notified when the SelectedDates collection has changed.

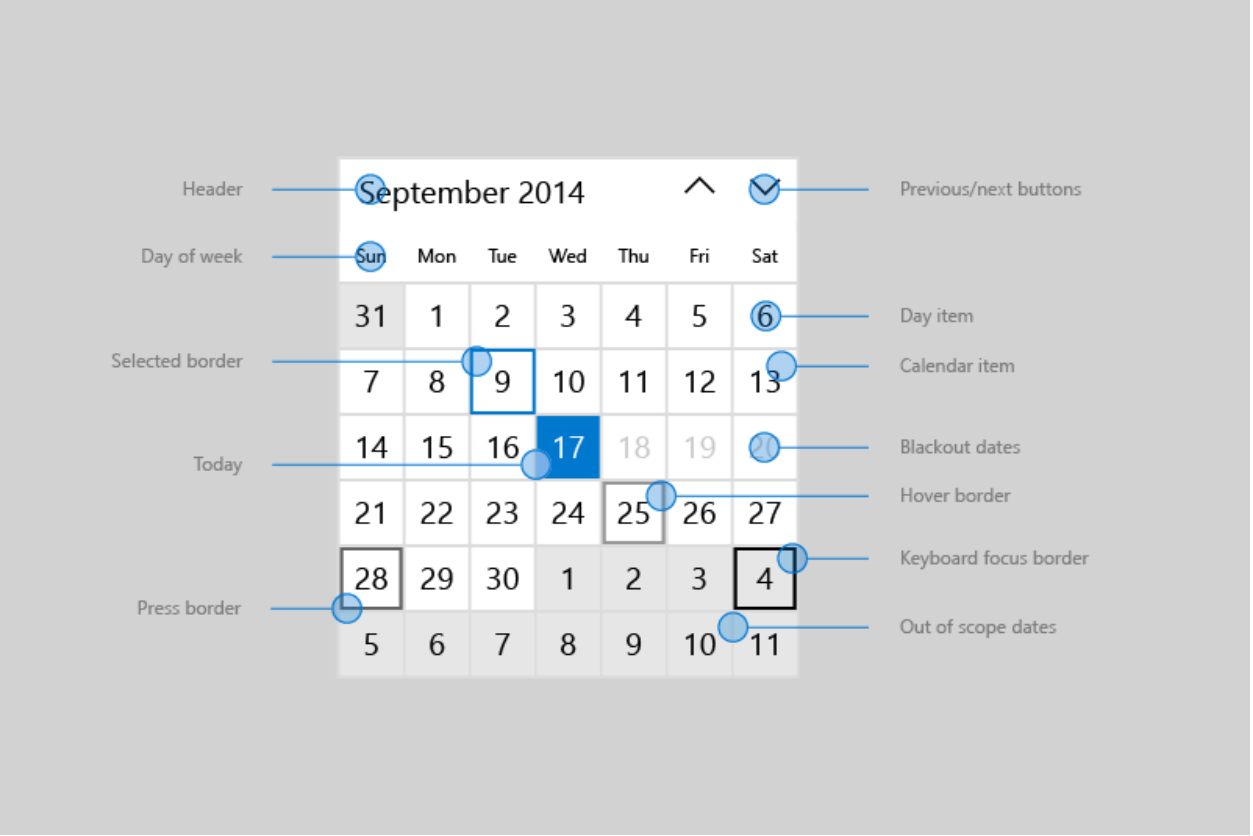
> **Note**
>
> For important info about date values, see **DateTime and Calendar values** in the Date and time controls article.

Customizing the calendar view's appearance

The calendar view is composed of both XAML elements defined in the ControlTemplate and visual elements rendered directly by the control.

- The XAML elements defined in the control template include the border that encloses the control, the header, previous and next buttons, and DayOfWeek elements. You can style and re-template these elements like any XAML control.
- The calendar grid is composed of CalendarViewDayItem objects. You can't style or re-template these elements, but various properties are provided to let you to customize their appearance.

This diagram shows the elements that make up the month view of the calendar. For more info, see the Remarks on the CalendarViewDayItem class.



This table lists the properties you can change to modify the appearance of calendar elements.

Expand table

Element	Properties
DayOfWeek	DayOfWeekFormat
CalendarItem	CalendarItemBackground, CalendarItemBorderBrush, CalendarItemBorderThickness, CalendarItemForeground
DayItem	DayItemFontFamily, DayItemFontSize, DayItemFontStyle, DayItemFontWeight, HorizontalDayItemAlignment, VerticalDayItemAlignment, CalendarViewDayItemStyle
MonthYearItem (in the year and decade views, equivalent to DayItem)	MonthYearItemFontFamily, MonthYearItemFontSize, MonthYearItemFontStyle, MonthYearItemFontWeight
FirstOfMonthLabel	FirstOfMonthLabelFontFamily, FirstOfMonthLabelFontSize, FirstOfMonthLabelFontStyle, FirstOfMonthLabelFontWeight, HorizontalFirstOfMonthLabelAlignment, VerticalFirstOfMonthLabelAlignment, IsGroupLabelVisible
FirstofYearDecadeLabel (in the year and decade views,	FirstOfYearDecadeLabelFontFamily, FirstOfYearDecadeLabelFontSize,

Element	Properties
equivalent to FirstOfMonthLabel)	FirstOfYearDecadeLabelFontStyle, FirstOfYearDecadeLabelFontWeight
Visual State Borders	FocusBorderBrush, HoverBorderBrush, PressedBorderBrush, SelectedBorderBrush, SelectedForeground, SelectedHoverBorderBrush, SelectedPressedBorderBrush
OutofScope	IsOutOfScopeEnabled, OutOfScopeBackground, OutOfScopeForeground
Today	IsTodayHighlighted, TodayFontWeight, TodayForeground

By default, the month view shows 6 weeks at a time. You can change the number of weeks shown by setting the NumberOfWeeksInView property. The minimum number of weeks to show is 2; the maximum is 8.

By default, the year and decade views show in a 4x4 grid. To change the number of rows or columns, call SetYearDecadeDisplayDimensions with the your desired number of rows and columns. This will change the grid for both the year and decade views.

Here, the year and decade views are set to show in a 3x4 grid.

C#

```
calendarView1.SetYearDecadeDisplayDimensions(3, 4);
```

By default, the minimum date shown in the calendar view is 100 years prior to the current date, and the maximum date shown is 100 years past the current date. You can change the minimum and maximum dates that the calendar shows by setting the MinDate and MaxDate properties.

C#

```
calendarView1.MinDate = new DateTime(2000, 1, 1);
calendarView1.MaxDate = new DateTime(2099, 12, 31);
```

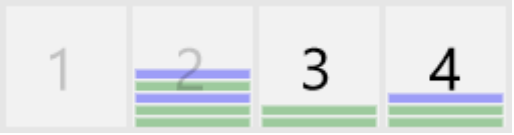
Updating calendar day items

Each day in the calendar is represented by a CalendarViewDayItem object. To access an individual day item and use its properties and methods, handle the CalendarViewDayItemChanging event and use the Item property of the event args to access the CalendarViewDayItem.

You can make a day not selectable in the calendar view by setting its CalendarViewDayItem.IsBlackout property to **true**.

You can show contextual information about the density of events in a day by calling the CalendarViewDayItem.SetDensityColors method. You can show from 0 to 10 density bars for each day, and set the color of each bar.

Here are some day items in a calendar. Days 1 and 2 are blacked out. Days 2, 3, and 4 have various density bars set.



Phased rendering

A calendar view can contain a large number of CalendarViewDayItem objects. To keep the UI responsive and enable smooth navigation through the calendar, calendar view supports phased rendering. This lets you break up processing of a day item into phases. If a day is moved out of view before all the phases are complete, no more time is used trying to process and render that item.

This example shows phased rendering of a calendar view for scheduling appointments.

- In phase 0, the default day item is rendered.
- In phase 1, you blackout dates that can't be booked. This includes past dates, Sundays, and dates that are already fully booked.
- In phase 2, you check each appointment that's booked for the day. You show a green density bar for each confirmed appointment and a blue density bar for each tentative appointment.

The `Bookings` class in this example is from a fictitious appointment booking app, and is not shown.

XAML

```
<CalendarView
CalendarViewDayItemChanging="CalendarView_CalendarViewDayItemChanging"/>
```

C#

```
private void CalendarView_CalendarViewDayItemChanging(CalendarView sender,
                                      CalendarViewDayItemChangingEventArgs
args)
{
```

```
    // Render basic day items.
    if (args.Phase == 0)
    {
        // Register callback for next phase.

args.RegisterUpdateCallback(CalendarView_CalendarViewDayItemChanging);
    }
    // Set blackout dates.
    else if (args.Phase == 1)
    {
        // Blackout dates in the past, Sundays, and dates that are fully
booked.
        if (args.Item.Date < DateTimeOffset.Now ||
            args.Item.Date.DayOfWeek == DayOfWeek.Sunday ||
            Bookings.HasOpenings(args.Item.Date) == false)
        {
            args.Item.IsBlackout = true;
        }
        // Register callback for next phase.

args.RegisterUpdateCallback(CalendarView_CalendarViewDayItemChanging);
    }
    // Set density bars.
    else if (args.Phase == 2)
    {
        // Avoid unnecessary processing.
        // You don't need to set bars on past dates or Sundays.
        if (args.Item.Date > DateTimeOffset.Now &&
            args.Item.Date.DayOfWeek != DayOfWeek.Sunday)
        {
            // Get bookings for the date being rendered.
            var currentBookings = Bookings.GetBookings(args.Item.Date);

            List<Color> densityColors = new List<Color>();
            // Set a density bar color for each of the days bookings.
            // It's assumed that there can't be more than 10 bookings in a
day. Otherwise,
            // further processing is needed to fit within the max of 10
density bars.
            foreach (booking in currentBookings)
            {
                if (booking.IsConfirmed == true)
                {
                    densityColors.Add(Colors.Green);
                }
                else
                {
                    densityColors.Add(Colors.Blue);
                }
            }
            args.Item.SetDensityColors(densityColors);
        }
    }
}
```

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** CalendarView class, SelectedDatesChanged event
- ✓ Open the WinUI 2 Gallery app and see the CalendarView in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related articles

- Date and time controls
- Calendar date picker
- Date picker
- Time picker

Date picker

Article • 02/26/2025

The date picker gives you a standardized way to let users pick a localized date value using touch, mouse, or keyboard input.

Is this the right control?

Use a date picker to let a user pick a known date, such as a date of birth, where the context of the calendar is not important.

If the context of a calendar is important, consider using a calendar date picker or calendar view.

For more info about choosing the right date control, see the Date and time controls article.

Examples

The entry point displays the chosen date, and when the user selects the entry point, a picker surface expands vertically from the middle for the user to make a selection. The date picker overlays other UI; it doesn't push other UI out of the way.

Create a date picker

✓ **Important APIs:** DatePicker class, SelectedDate property

Open the WinUI 3 Gallery app and see the DatePicker in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This example shows how to create a simple date picker with a header.

XAML

```
<DatePicker x:Name="exampleDatePicker" Header="Pick a date"/>
```

C#

```
DatePicker exampleDatePicker = new DatePicker();
```

```
exampleDatePicker.Header = "Pick a date";
```

The resulting date picker looks like this:

Formatting the date picker

By default, the date picker shows the day, month, and year. If your scenario for the date picker doesn't require all the fields, you can hide the ones you don't need. To hide a field, set its corresponding *field*Visible property to `false`: DayVisible, MonthVisible, or YearVisible.

Here, only the year is needed, so the day and month fields are hidden.

XAML

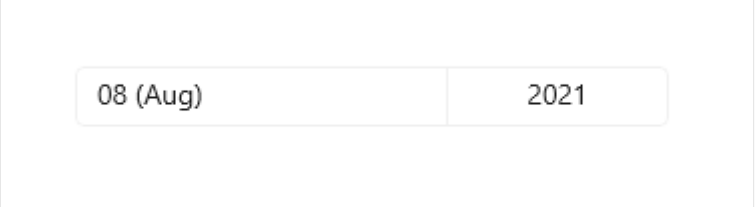
```
<DatePicker x:Name="yearDatePicker" Header="In what year was Microsoft
founded?"
            MonthVisible="False" DayVisible="False"/>
```

The string content of each `ComboBox` in the `DatePicker` is created by a DateTimeFormatter. You inform the `DateTimeFormatter` how to format the date value by providing a string that is either a *format template* or a *format pattern*. For more info, see the DayFormat, MonthFormat, and YearFormat properties.

Here, a *format pattern* is used to show the month as an integer and abbreviation. You can add literal strings to the format pattern, such as the parentheses around the month abbreviation: `({month.abbreviated})`.

XAML

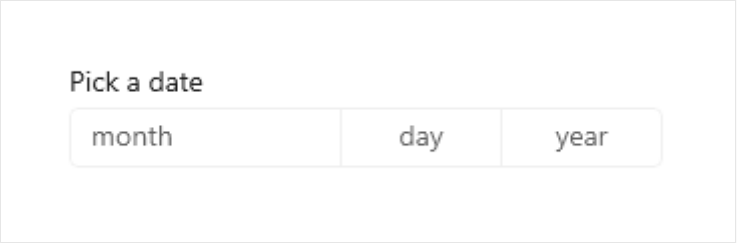
```
<DatePicker MonthFormat="{}{month.integer(2)} ({month.abbreviated})"
DayVisible="False"/>
```



Date values

The date picker control has both Date/DateChanged and SelectedDate/SelectedDateChanged APIs. The difference between these is that `Date` is not nullable, while `SelectedDate` is nullable.

The value of `SelectedDate` is used to populate the date picker and is `null` by default. If `SelectedDate` is `null`, the `Date` property is set to 12/31/1600; otherwise, the `Date` value is synchronized with the `SelectedDate` value. When `SelectedDate` is `null`, the picker is 'unset' and shows the field names instead of a date.



You can set the MinYear and MaxYear properties to restrict the date values in the picker. By default, `MinYear` is set to 100 years prior to the current date and `MaxYear` is set to 100 years past the current date.

If you set only `MinYear` or `MaxYear`, you need to ensure that a valid date range is created by the date you set and the default value of the other date; otherwise, no date will be available to select in the picker. For example, setting only `yearDatePicker.MaxYear = new DateTimeOffset(new DateTime(900, 1, 1));` creates an invalid date range with the default value of `MinYear`.

Initializing a date value

The date properties can't be set as a XAML attribute string, because the Windows Runtime XAML parser doesn't have a conversion logic for converting strings to dates as

DateTime / DateTimeOffset objects. Here are some suggested ways these objects can be defined in code and set to a date other than the current date.

- DateTime: Instantiate a Windows.Globalization.Calendar object (it is initialized to the current date). Set Year, or call AddYears, to adjust the date. Then, call Calendar.GetDateTime and use the returned `DateTime` to set the date property.
- DateTimeOffset: Call the constructor. For the inner System.DateTime, use the constructor signature. Or, construct a default DateTimeOffset (it is initialized to the current date) and call AddYears.

Another possible technique is to define a date that's available as a data object or in the data context, then set the date property as a XAML attribute that references a {Binding} markup extension that can access the date as data.

> ⓘ **Note**
>
> For important info about date values, see **DateTime and Calendar values** in the Date and time controls article.

This example demonstrates setting the `SelectedDate`, `MinYear`, and `MaxYear` properties on different `DatePicker` controls.

XAML

```
<DatePicker x:Name="yearDatePicker" MonthVisible="False"
DayVisible="False"/>
<DatePicker x:Name="arrivalDatePicker" Header="Arrival date"/>
```

C#

```
public MainPage()
{
    this.InitializeComponent();

    // Set minimum year to 1900 and maximum year to 1999.
    yearDatePicker.SelectedDate = new DateTimeOffset(new DateTime(1950, 1,
1));
    yearDatePicker.MinYear = new DateTimeOffset(new DateTime(1900, 1, 1));
    // Using a different DateTimeOffset constructor.
    yearDatePicker.MaxYear = new DateTimeOffset(1999, 12, 31, 0, 0, 0, new
TimeSpan());

    // Set minimum to the current year and maximum to five years from now.
    arrivalDatePicker.MinYear = DateTimeOffset.Now;
```

```
    arrivalDatePicker.MaxYear = DateTimeOffset.Now.AddYears(5);
}
```

Using the date values

To use the date value in your app, you typically use a data binding to the SelectedDate property, or handle the SelectedDateChanged event.

> For an example of using a `DatePicker` and `TimePicker` together to update a single `DateTime` value, see Calendar, date, and time controls - Use a date picker and time picker together.

Here, you use a `DatePicker` to let a user select their arrival date. You handle the `SelectedDateChanged` event to update a DateTime instance named `arrivalDateTime`.

XAML

```
<StackPanel>
    <DatePicker x:Name="arrivalDatePicker" Header="Arrival date"
                DayFormat="{}{day.integer} ({dayofweek.abbreviated})"

SelectedDateChanged="arrivalDatePicker_SelectedDateChanged"/>
    <Button Content="Clear" Click="ClearDateButton_Click"/>
    <TextBlock x:Name="arrivalText" Margin="0,12"/>
</StackPanel>
```

C#

```
public sealed partial class MainPage : Page
{
    DateTime arrivalDateTime;

    public MainPage()
    {
        this.InitializeComponent();

        // Set minimum to the current year and maximum to five years from
now.
        arrivalDatePicker.MinYear = DateTimeOffset.Now;
        arrivalDatePicker.MaxYear = DateTimeOffset.Now.AddYears(5);
    }

    private void arrivalDatePicker_SelectedDateChanged(DatePicker sender,
DatePickerSelectedValueChangedEventArgs args)
    {
        if (arrivalDatePicker.SelectedDate != null)
        {
```

```
            arrivalDateTime = new DateTime(args.NewDate.Value.Year,
args.NewDate.Value.Month, args.NewDate.Value.Day);
        }
        arrivalText.Text = arrivalDateTime.ToString();
    }

    private void ClearDateButton_Click(object sender, RoutedEventArgs e)
    {
        arrivalDatePicker.SelectedDate = null;
        arrivalText.Text = string.Empty;
    }
}
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** DatePicker class, SelectedDate property
- ✓ Open the WinUI 2 Gallery app and see the DatePicker in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

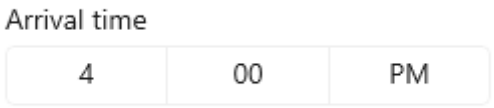
Related articles

- Date and time controls
- Calendar date picker
- Calendar view
- Time picker

Time picker

Article • 02/26/2025

The time picker gives you a standardized way to let users pick a time value using touch, mouse, or keyboard input.



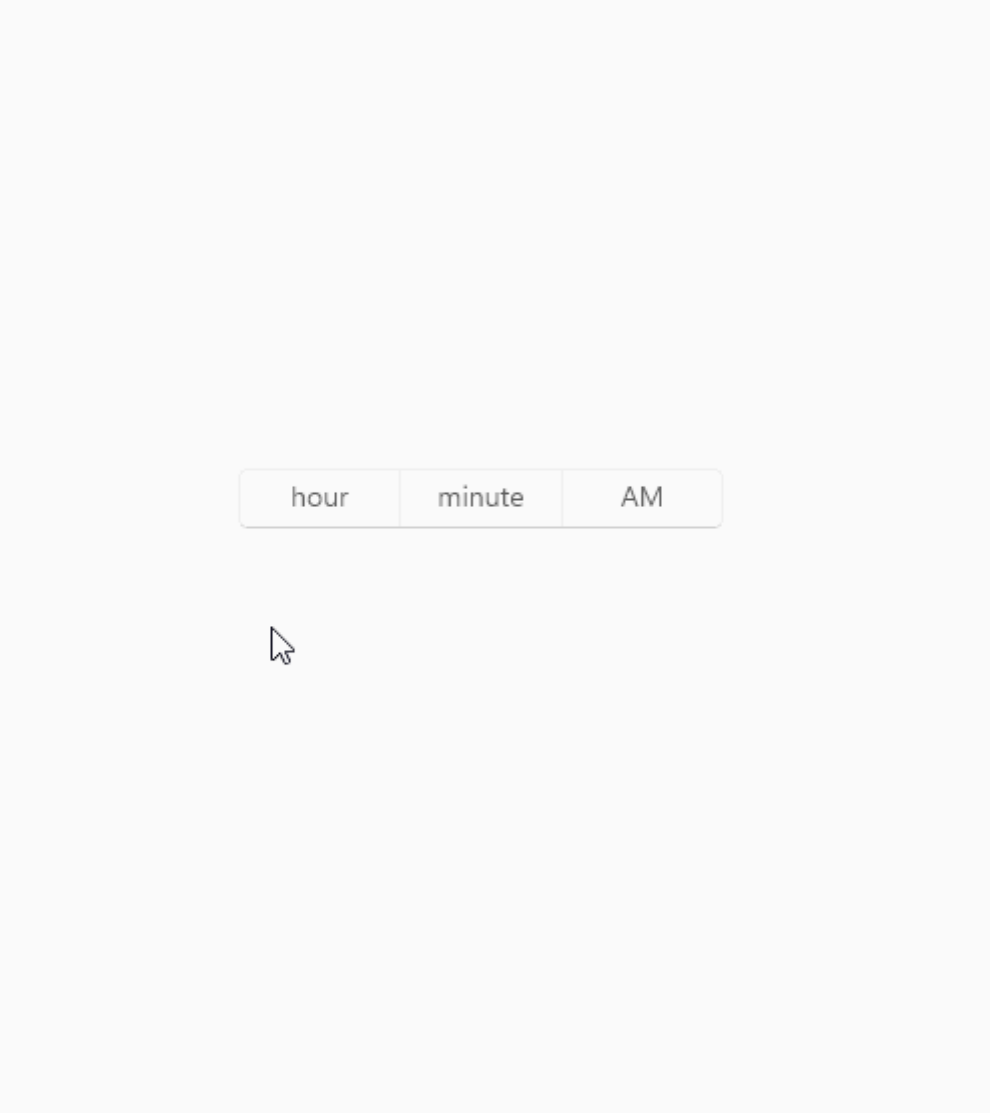
Is this the right control?

Use a time picker to let a user pick a single time value.

For more info about choosing the right control, see the Date and time controls article.

Examples

The entry point displays the chosen time, and when the user selects the entry point, a picker surface expands vertically from the middle for the user to make a selection. The time picker overlays other UI; it doesn't push other UI out of the way.



Create a time picker

✔ **Important APIs:** TimePicker class, SelectedTime property

Open the WinUI 3 Gallery app and see the TimePicker in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This example shows how to create a simple time picker with a header.

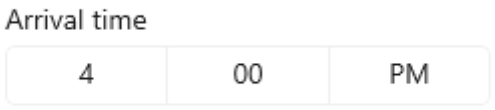
XAML

```
<TimePicker x:Name="arrivalTimePicker" Header="Arrival time"/>
```

C#

```
TimePicker arrivalTimePicker = new TimePicker();
arrivalTimePicker.Header = "Arrival time";
```

The resulting time picker looks like this:



Formatting the time picker

By default, the time picker shows a 12-hour clock with an AM/PM selector. You can set the ClockIdentifier property to "24HourClock" to show a 24-hour clock instead.

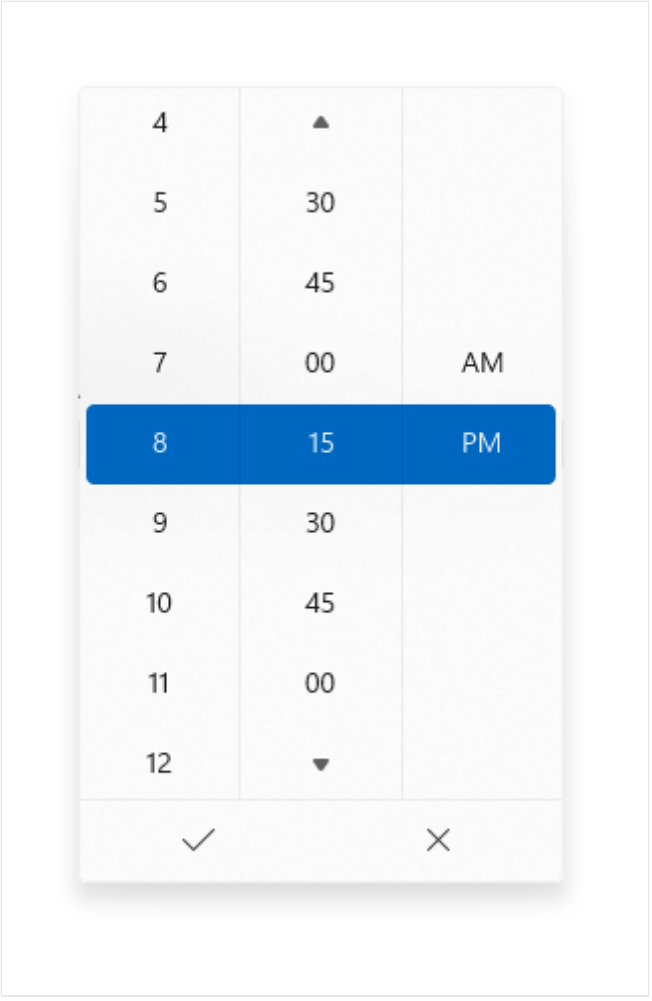
XAML

```
<TimePicker Header="24HourClock" SelectedTime="18:21"
ClockIdentifier="24HourClock"/>
```

You can set the MinuteIncrement property to indicate the time increments shown in the minute picker. For example, 15 specifies that the `TimePicker` minute control displays only the choices 00, 15, 30, 45.

XAML

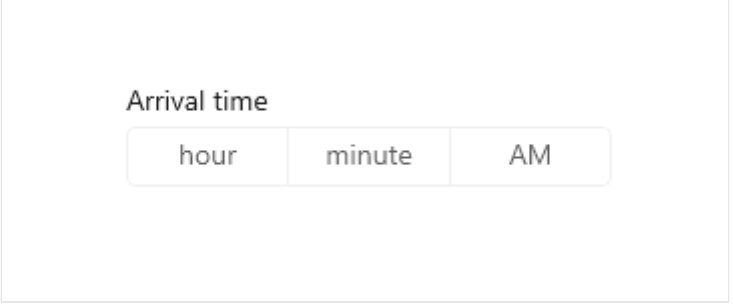
```
<TimePicker MinuteIncrement="15"/>
```



Time values

The time picker control has both Time/TimeChanged and SelectedTime/SelectedTimeChanged APIs. The difference between these is that `Time` is not nullable, while `SelectedTime` is nullable.

The value of `SelectedTime` is used to populate the time picker and is `null` by default. If `SelectedTime` is `null`, the `Time` property is set to a TimeSpan of 0; otherwise, the `Time` value is synchronized with the `SelectedTime` value. When `SelectedTime` is `null`, the picker is 'unset' and shows the field names instead of a time.



Initializing a time value

In code, you can initialize the time properties to a value of type `TimeSpan`:

```
C#

TimePicker timePicker = new TimePicker
{
    SelectedTime = new TimeSpan(14, 15, 00) // Seconds are ignored.
};
```

You can set the time value as an attribute in XAML. This is probably easiest if you're already declaring the `TimePicker` object in XAML and aren't using bindings for the time value. Use a string in the form *Hh:Mm* where *Hh* is hours and can be between 0 and 23 and *Mm* is minutes and can be between 0 and 59.

```
XAML

<TimePicker SelectedTime="14:15"/>
```

> **Note**
>
> For important info about date and time values, see DateTime and Calendar values in the *Date and time controls* article.

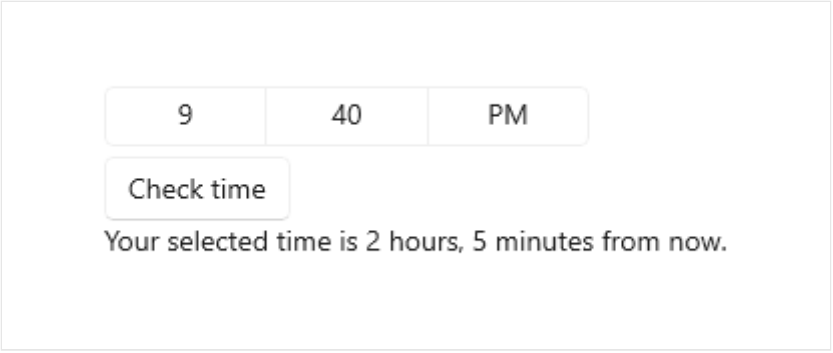
Using the time values

To use the time value in your app, you typically use a data binding to the SelectedTime or Time property, use the time properties directly in your code, or handle the SelectedTimeChanged or TimeChanged event.

> For an example of using a `DatePicker` and `TimePicker` together to update a single `DateTime` value, see Calendar, date, and time controls - Use a date picker and time picker together.

Here, the `SelectedTime` property is used to compare the selected time to the current time.

Notice that because the `SelectedTime` property is nullable, you have to explicitly cast it to `DateTime`, like this: `DateTime myTime = (DateTime)(DateTime.Today + checkTimePicker.SelectedTime);`. The `Time` property, however, could be used without a cast, like this: `DateTime myTime = DateTime.Today + checkTimePicker.Time;`.



XAML

```
<StackPanel>
    <TimePicker x:Name="checkTimePicker"/>
    <Button Content="Check time" Click="{x:Bind CheckTime}"/>
    <TextBlock x:Name="resultText"/>
</StackPanel>
```

C#

```
private void CheckTime()
{
    // Using the Time property.
    // DateTime myTime = DateTime.Today + checkTimePicker.Time;
    // Using the SelectedTime property (nullable requires cast to DateTime).
    DateTime myTime = (DateTime)(DateTime.Today +
checkTimePicker.SelectedTime);
    if (DateTime.Now >= myTime)
    {
        resultText.Text = "Your selected time has already past.";
    }
    else
    {
        string hrs = (myTime - DateTime.Now).Hours.ToString();
        string mins = (myTime - DateTime.Now).Minutes.ToString();
        resultText.Text = string.Format("Your selected time is {0} hours,
{1} minutes from now.", hrs, mins);
    }
}
```

UWP and WinUI 2

ⓘ Important

The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** TimePicker class, SelectedTime property
- ✓ Open the WinUI 2 Gallery app and see the TimePicker in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related topics

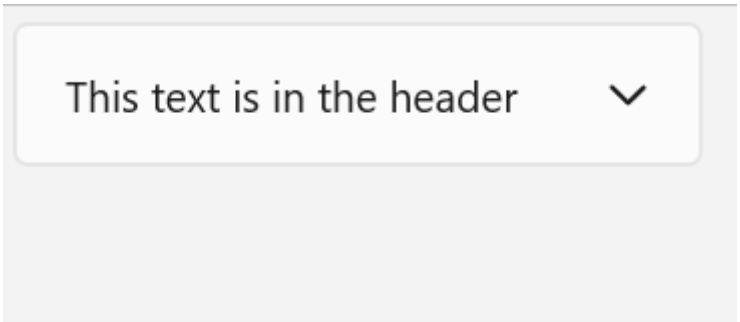
- Date and time controls
- Calendar date picker
- Calendar view
- Date picker

Expander

Article • 02/26/2025

The Expander control lets you show or hide less important content that's related to a piece of primary content that's always visible. Items contained in the Header are always visible. The user can expand and collapse the Content area, where secondary content is displayed, by interacting with the header. When the content area is expanded, it pushes other UI elements out of the way; it does not overlay other UI. The `Expander` can expand upwards or downwards.

Both the `Header` and `Content` areas can contain any content, from simple text to complex UI layouts. For example, you can use the control to show additional options for an item.



Is this the right control?

Use an `Expander` when some primary content should always be visible, but related secondary content may be hidden until needed. This UI is commonly used when display space is limited and when information or options can be grouped together. Hiding the secondary content until it's needed can also help to focus the user on the most important parts of your app.

Create an Expander

✓ **Important APIs:** Expander class, Header property, Content property

Open the WinUI 3 Gallery app and see the Expander in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This example shows how to create a simple Expander with the default styling. The Header property defines the element that is always visible. The Content property defines the element that can be collapsed and expanded. This example creates an `Expander` that looks like the previous illustration.

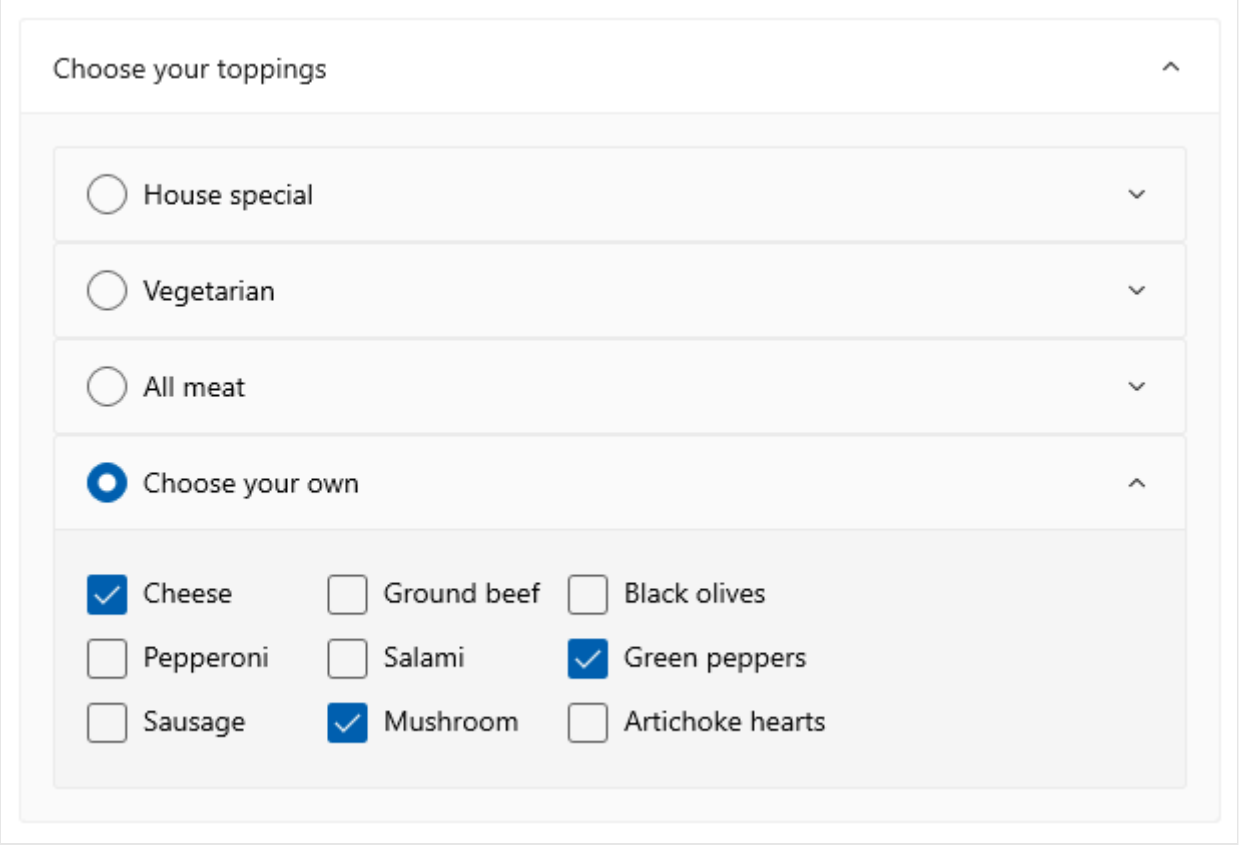
XAML

```
<Expander Header="This text is in the header"
                Content="This is in the content"/>
```

Expander content

The Content property of an `Expander` can be any type of object, but is typically a string or UIElement. For more details about setting the `Content` property, see the Remarks section of the ContentControl class.

You can use complex, interactive UI as the content of the `Expander`, including nested `Expander` controls in the content of a parent `Expander` as shown here.



Content alignment

You can align content by setting the HorizontalContentAlignment and VerticalContentAlignment properties on the `Expander` control. When you set these

properties, the alignment applies only to the expanded content, not the header.

Control the size of an Expander

By default, the Header and Content areas automatically size to fit their contents. It's important to use the correct techniques to control the size of the `Expander` to avoid undesirable appearance or behavior.

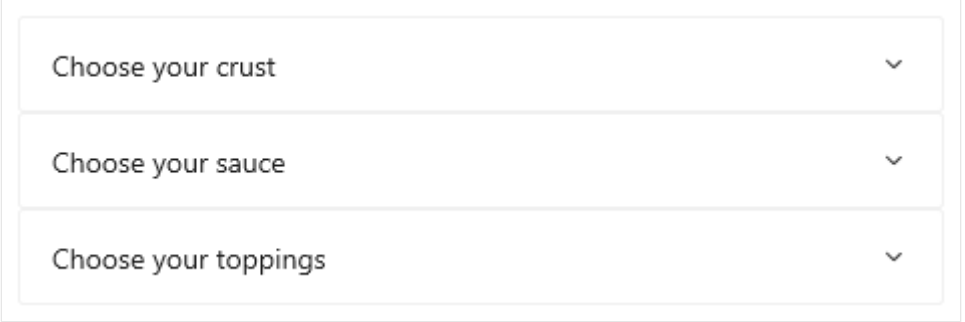
Width

If the content is wider than the header, the header width increases to match the content area when expanded, and shrinks when the content area is collapsed. To prevent the control width from changing when expanded or collapsed, you can set an explicit width, or, if the control is the child of a Panel, set HorizontalAlignment to Stretch and let the layout panel control the sizing.

Here, a series of related `Expander` controls are placed in a StackPanel. The `HorizontalAlignment` of each `Expander` in the `StackPanel` is set to `Stretch` using a Style in the `StackPanel` Resources, and the width of the `StackPanel` determines the width of the `Expander` controls.

XAML

```
<StackPanel x:Name="ExpanderStack" MaxWidth="600">
    <StackPanel.Resources>
        <Style TargetType="Expander">
            <Setter Property="HorizontalAlignment" Value="Stretch"/>
            <Setter Property="HorizontalContentAlignment" Value="Stretch"/>
        </Style>
    </StackPanel.Resources>
    <Expander Header="Choose your crust"> ... </Expander>
    <Expander Header="Choose your sauce"> ... </Expander>
    <Expander Header="Choose your toppings"> ... </Expander>
 </StackPanel>
```



Height

Do not specify a Height on the `Expander`. If you do, the control will reserve that space even when the content area is collapsed, which defeats the purpose of the `Expander`. To specify the size of the expanded content area, set size dimensions on the content of the `Expander`. If you need to, you can constrain the `Height` of the content and make the content scrollable.

Scrollable content

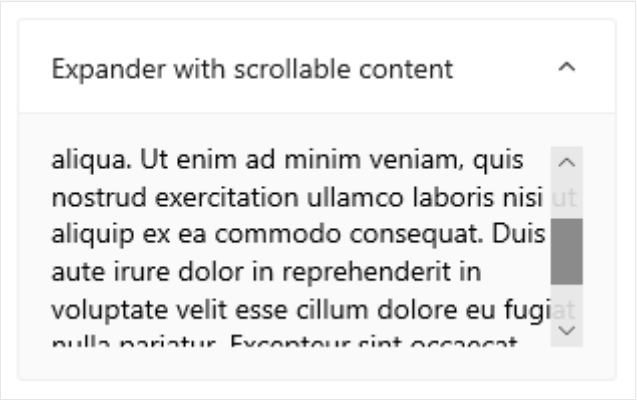
If your content is too large for the size of the content area, you can wrap the content a ScrollViewer to make the content area scrollable. The `Expander` control does not automatically provide scrolling capability.

When you place a `ScrollViewer` in the `Expander` content, set the height on the `ScrollViewer` control to the required height for the content area. If you instead set the height dimension on the content inside the `ScrollViewer`, `ScrollViewer` doesn't recognize this setting and therefore does not provide scrollable content.

The following example shows how to create an `Expander` control that contains scrollable text as its content.

XAML

```
<Expander Header="Expander with scrollable content">
    <ScrollViewer MaxHeight="200">
        <Grid>
            <TextBlock TextWrapping="Wrap">
                Lorem ipsum dolor sit amet, consectetur adipisicing elit,
                sed do eiusmod tempor incididunt ut labore et dolore magna
                aliqua. Ut enim ad minim veniam, quis nostrud exercitation
                ullamco laboris nisi ut aliquip ex ea commodo consequat.
                Duis aute irure dolor in reprehenderit in voluptate velit
                esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
                occaecat cupidatat non proident, sunt in culpa qui officia
                deserunt mollit anim id est laborum.
            </TextBlock>
        </Grid>
    </ScrollViewer>
</Expander>
```



Expanding and collapsing the content area

By default, the Expander is collapsed and expands downwards.

- Set the IsExpanded property to `true` to have the content area initially expanded.
- Set the ExpandDirection property to Up to make the content expand upward.

XAML

```
<Expander IsExpanded="True" ExpandDirection="Up">
```

An `Expander` is expanded or collapsed either programmatically by setting the `IsExpanded` property, or by interacting with the `Header`; it cannot be light-dismissed.

> **Tip**
>
> Transient UI, such as a `Flyout` or the open drop-down of a `ComboBox`, closes when you click or tap outside of it. This is called *light-dismiss*. The content area of an `Expander` is not considered transient and does not overlay other UI, so it does not support light-dismiss.

You can also handle the Expanding and Collapsed events to take an action when the content is shown or hidden. Here are some examples of these events.

Expanding event

In this example, you have a group of expanders and want to have only one open at a time. When the user opens an `Expander`, you handle the Expanding event and collapse all `Expander` controls in the group other than the one the user clicked.

> **Caution**

Depending on your app and user experience, it might be a convenience to automatically collapse `Expander` controls when the user expands a different one. However, this also takes control away from the user. If the behavior might be useful, consider making it an option that the user can easily set.

XAML

```
<StackPanel x:Name="ExpanderStack">
    <Expander Header="Choose your crust"
                  Expanding="Expander_Expanding"> ... </Expander>
    <Expander Header="Choose your sauce"
                  Expanding="Expander_Expanding"> ... </Expander>
    <Expander Header="Choose your toppings"
                  Expanding="Expander_Expanding"> ... </Expander>
 </StackPanel>
```

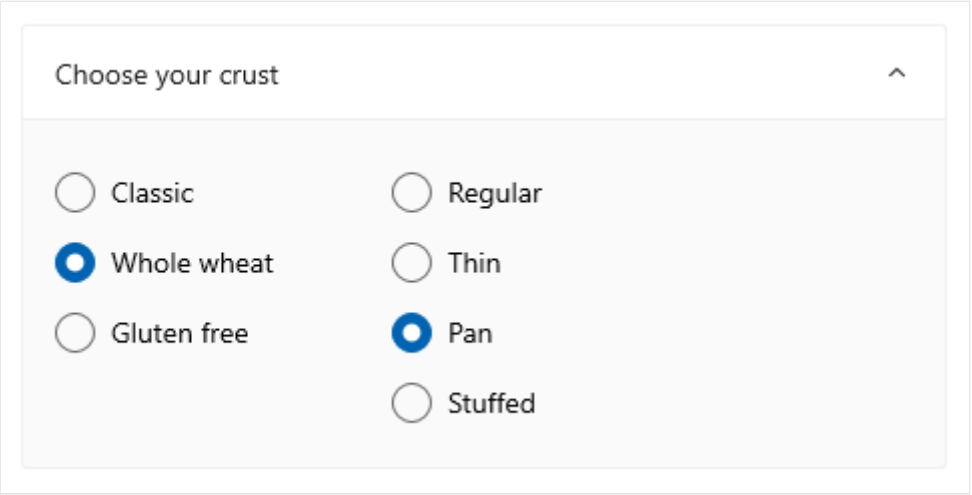
C#

```
// Let the user opt out of custom behavior.
private bool _autoCollapse = true;

private void Expander_Expanding(muxc.Expander sender,
                                muxc.ExpanderExpandingEventArgs args)
{
    if (_autoCollapse == true)
    {
        foreach (muxc.Expander ex in ExpanderStack.Children)
        {
            if (ex != sender && ex.IsExpanded)
                ex.IsExpanded = false;
        }
    }
}
```

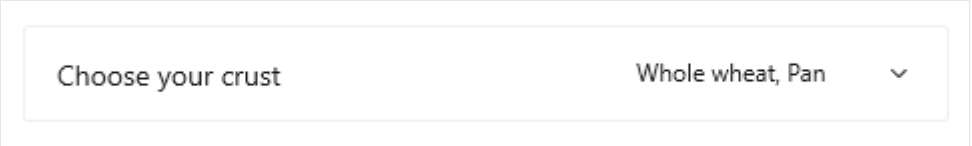
Collapsed event

In this example, you handle the Collapsed event and populate the `Header` with a summary of options that are selected in the `Content`.

This image shows the `Expander` with the content expanded and options selected.



When collapsed, the selected options are summarized in the header so that the user can still see them without opening the `Expander`.



XAML

```
<Expander IsExpanded="True"
        Expanding="Expander_Expanding"
        Collapsed="Expander_Collapsed">
    <Expander.Header>
        <Grid>
            <TextBlock Text="Choose your crust"/>
            <TextBlock x:Name="tbCrustSelections"
                       HorizontalAlignment="Right"
                       Style="{StaticResource CaptionTextBlockStyle}"/>
        </Grid>
    </Expander.Header>
    <StackPanel Orientation="Horizontal">
        <RadioButtons x:Name="rbCrustType" SelectedIndex="0">
            <x:String>Classic</x:String>
            <x:String>Whole wheat</x:String>
            <x:String>Gluten free</x:String>
        </RadioButtons>
        <RadioButtons x:Name="rbCrustStyle" SelectedIndex="0"
                      Margin="48,0,0,0">
            <x:String>Regular</x:String>
            <x:String>Thin</x:String>
            <x:String>Pan</x:String>
            <x:String>Stuffed</x:String>
        </RadioButtons>
    </StackPanel>
</Expander>
```

C#

```
private void Expander_Collapsed(muxc.Expander sender,
                                muxc.ExpanderCollapsedEventArgs args)
{
    // Update the header with options selected in the content.
    tbCrustSelections.Text = rbCrustType.SelectedItem.ToString() +
        ", " + rbCrustStyle.SelectedItem.ToString();
}
```

Lightweight styling

You can modify the default `Style` and `ControlTemplate` to give the control a unique appearance. See the Control Style and Template section of the Expander API docs for a list of the available theme resources. For more info, see the Light-weight styling section of the Styling controls article.

Recommendations

- Use an `Expander` when display space is limited and some secondary content may be hidden until the user requests it.

Code examples

This XAML creates the group of `Expander` controls shown in other parts of this article. The code for the `Expanding` and `Collapsed` event handlers is also shown in previous sections.

XAML

```
<StackPanel x:Name="ExpanderStack" MaxWidth="600">
    <StackPanel.Resources>
        <Style TargetType="Expander">
            <Setter Property="HorizontalAlignment" Value="Stretch"/>
            <Setter Property="HorizontalContentAlignment" Value="Stretch"/>
        </Style>
    </StackPanel.Resources>
    <Expander IsExpanded="True"
                  Expanding="Expander_Expanding"
                  Collapsed="Expander_Collapsed">
        <Expander.Header>
            <Grid>
                <TextBlock Text="Choose your crust"/>
                <TextBlock x:Name="tbCrustSelections"
                           HorizontalAlignment="Right"
        Style="{StaticResource CaptionTextBlockStyle}"/>
            </Grid>
```

```
        </Expander.Header>
        <StackPanel Orientation="Horizontal">
            <RadioButtons x:Name="rbCrustType" SelectedIndex="0">
                <x:String>Classic</x:String>
                <x:String>Whole wheat</x:String>
                <x:String>Gluten free</x:String>
            </RadioButtons>
            <RadioButtons x:Name="rbCrustStyle" SelectedIndex="0"
                   Margin="48,0,0,0">
                <x:String>Regular</x:String>
                <x:String>Thin</x:String>
                <x:String>Pan</x:String>
                <x:String>Stuffed</x:String>
            </RadioButtons>
        </StackPanel>
    </Expander>

    <Expander Header="Choose your sauce" Margin="24"
            Expanding="Expander_Expanding">
        <RadioButtons SelectedIndex="0" MaxColumns="2">
            <x:String>Classic red</x:String>
            <x:String>Garlic</x:String>
            <x:String>Pesto</x:String>
            <x:String>Barbecue</x:String>
        </RadioButtons>
    </Expander>

    <Expander Header="Choose your toppings"
                    Expanding="Expander_Expanding">
        <StackPanel>
            <Expander>
                <Expander.Header>
                    <RadioButton GroupName="Toppings" Content="House
special"/>
                </Expander.Header>
                <TextBlock Text="Cheese, pepperoni, sausage, black olives,
mushrooms"
                           TextWrapping="WrapWholeWords"/>
            </Expander>
            <Expander>
                <Expander.Header>
                    <RadioButton GroupName="Toppings" Content="Vegetarian"/>
                </Expander.Header>
                <TextBlock Text="Cheese, mushrooms, black olives, green
peppers, artichoke hearts"
                           TextWrapping="WrapWholeWords"/>
            </Expander>
            <Expander>
                <Expander.Header>
                    <RadioButton GroupName="Toppings" Content="All meat"/>
                </Expander.Header>
                <TextBlock Text="Cheese, pepperoni, sausage, ground beef,
salami"
                           TextWrapping="WrapWholeWords"/>
            </Expander>
```

```
            <Expander>
                <Expander.Header>
                    <RadioButton GroupName="Toppings" Content="Choose your
own"/>
                </Expander.Header>
                <StackPanel Orientation="Horizontal">
                    <StackPanel>
                        <CheckBox Content="Cheese"/>
                        <CheckBox Content="Pepperoni"/>
                        <CheckBox Content="Sausage"/>
                    </StackPanel>
                    <StackPanel>
                        <CheckBox Content="Ground beef"/>
                        <CheckBox Content="Salami"/>
                        <CheckBox Content="Mushroom"/>
                    </StackPanel>
                    <StackPanel>
                        <CheckBox Content="Black olives"/>
                        <CheckBox Content="Green peppers"/>
                        <CheckBox Content="Artichoke hearts"/>
                    </StackPanel>
                </StackPanel>
            </Expander>
        </StackPanel>
    </Expander>
</StackPanel>
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The Expander for UWP apps requires WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in the Microsoft.UI.Xaml.Controls namespace.

- ✓ **WinUI 2 Apis:** Expander class, Header property, Content property
- ✓ Open the WinUI 2 Gallery app and see the Expander in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and

functionality. Get the app from the Microsoft Store or get the source code on GitHub.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:Expander />
```

Related articles

- Page layout
- Expander class

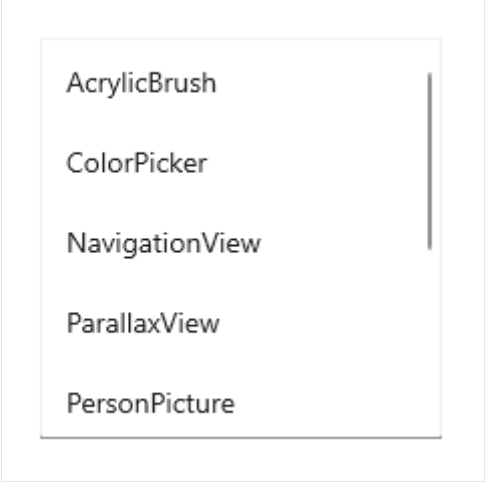
Scroll viewer controls

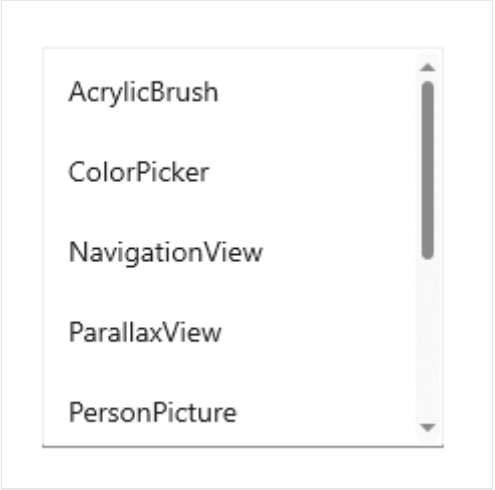
Article • 02/26/2025

When there is more UI content to show than you can fit in an area, use a scroll viewer control.

Scroll viewers enable content to extend beyond the bounds of the viewport (visible area). Users reach this content by manipulating the scroll viewer surface through touch, mousewheel, keyboard, or a gamepad, or by using the mouse or pen cursor to interact with the scroll viewer's scrollbar.

Depending on the situation, the scroll viewer's scrollbar uses two different visualizations, shown in the following illustration: the panning indicator (left) and the traditional scrollbar thumb (right).





ⓘ Important

WinUI 3 has two different scroll viewer controls available: ScrollViewer and ScrollView. Whenever we speak generically about *scroll viewer* controls, the

information applies to both controls.

Scrolling, panning, and zooming

Use a scroll viewer control to allow *scrolling*, *panning*, and *zooming* of your content.

- *Scrolling*: Moving the content vertically or horizontally by dragging the scrollbar thumb or using the scroll wheel on a mouse.
- *Panning*: Moving content vertically or horizontally using touch or pen input.
  - For more info about scrolling and panning, see Guidelines for panning.
- *Zooming*: Optically increasing or decreasing the scale of the content.
  - For more info about zooming, see Optical zoom and resizing.

The scrollbar is conscious of the user's input method and uses it to determine which visualization to display.

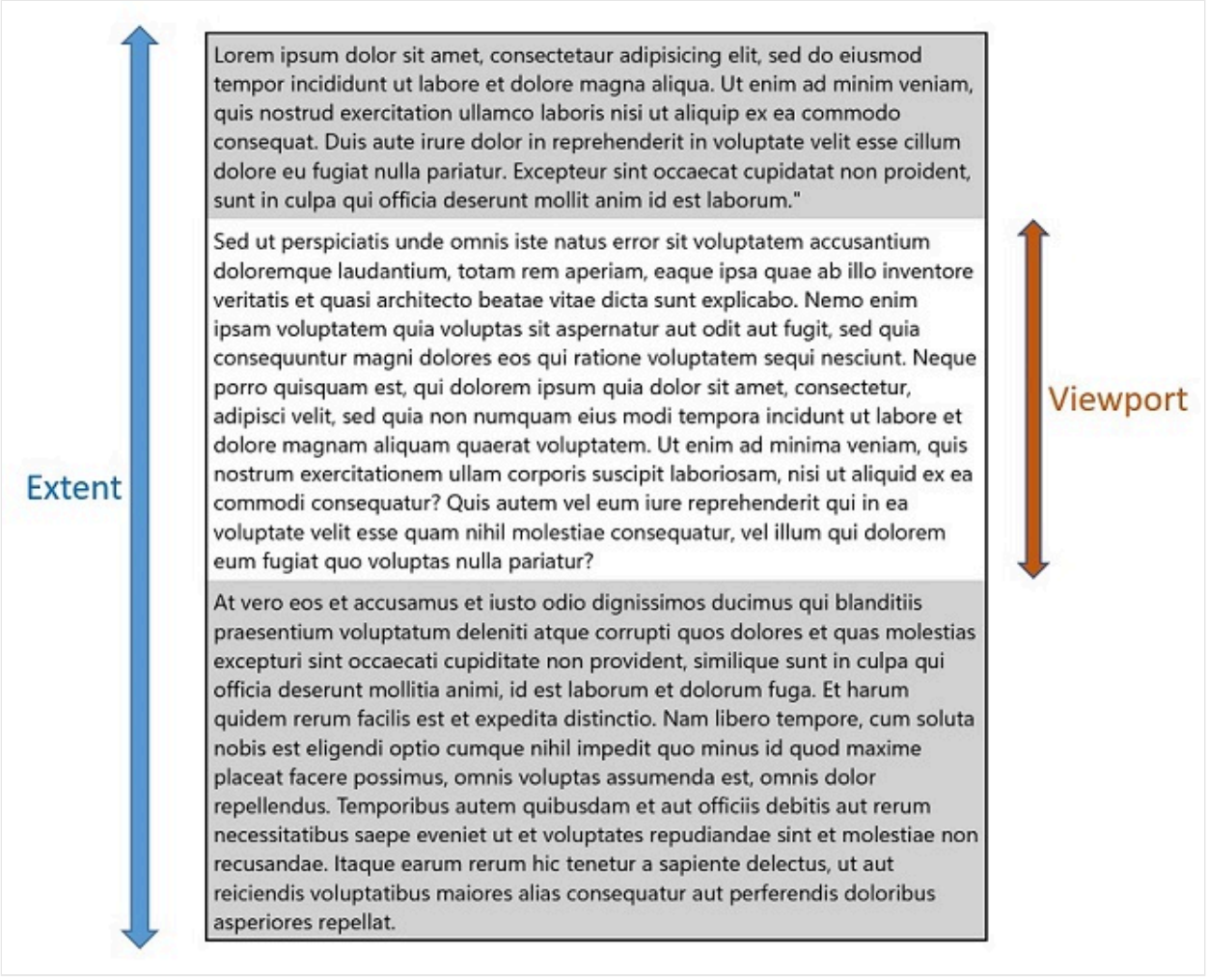
- When the region is scrolled without manipulating the scrollbar directly, for example, by touch, the panning indicator appears, displaying the current scroll position.
- When the mouse or pen cursor moves over the panning indicator, it morphs into the traditional scrollbar. Dragging the scrollbar thumb manipulates the scrolling region.

> **Note**
>
> When the scrollbar is visible it is overlaid as 16px on top of the content inside your ScrollViewer. In order to ensure good UX design you will want to ensure that no interactive content is obscured by this overlay. Additionally if you would prefer not

to have UX overlap, leave 16px of padding on the edge of the viewport to allow for the scrollbar.

Viewport and extent

A scroll viewer is composed of two main regions that are important to understanding its functionality. The area that includes all of the scrollable content, both hidden and visible, is the *extent*. The visible area of the control where the content is shown is the *viewport*.



Various APIs are available that let you get the height and width of these regions, as well as the *scrollable* height and width, which is the difference between the extent size and the viewport size.

Recommendations

- Whenever possible, design for vertical scrolling rather than horizontal.
- Use one-axis panning for content regions that extend beyond one viewport boundary (vertical or horizontal). Use two-axis panning for content regions that extend beyond both viewport boundaries (vertical and horizontal).

- Use the built-in scroll functionality in the items view, list view, grid view, combo box, list box, text input box, and hub controls. With those controls, if there are too many items to show all at once, the user is able to scroll either horizontally or vertically over the list of items.
- If you want the user to pan in both directions around a larger area, and possibly to zoom, too, for example, if you want to allow the user to pan and zoom over a full-sized image (rather than an image sized to fit the screen) then place the image inside a scroll viewer.
- If the user will scroll through a long passage of text, configure the scroll viewer to scroll vertically only.
- Use a scroll viewer to contain one object only. Note that the one object can be a layout panel, in turn containing any number of objects of its own.
- If you need to handle pointer events for a UIElement in a scrollable view (such as a ScrollViewer or ListView), you must explicitly disable support for manipulation events on the element in the view by calling UIElement.CancelDirectManipulation. To re-enable manipulation events in the view, call UIElement.TryStartDirectManipulation.

Create a scroll viewer

✓ **Important APIs:** ScrollView class, ScrollViewer class, ScrollBar class

Open the WinUI 3 Gallery app and see the ScrollView in action

Open the WinUI 3 Gallery app and see the ScrollViewer in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

A scroll viewer control can be used to make content scrollable by explicitly wrapping the content in the scroll viewer, or by placing a scroll viewer in the control template of a content control.

Scroll viewer in a control template

It's typical for a scroll viewer control to exist as a composite part of other controls. A scroll viewer part will display a viewport along with scrollbars only when the host control's layout space is being constrained smaller than the expanded content size.

ItemsView includes a `ScrollView` control in its template. You can access the `ScrollView` though the ItemsView.ScrollView property.

ListView and GridView templates always include a `ScrollViewer`. TextBox and RichEditBox also include a `ScrollViewer` in their templates. To influence some of the behavior and properties of the built in `ScrollViewer` part, `ScrollViewer` defines a number of XAML attached properties that can be set in styles and used in template bindings. For more info about attached properties, see Attached properties overview.

Set scrollable content

Content inside of a scroll viewer becomes scrollable when it's larger than the scroll viewer's viewport

This example sets a `Rectangle` as the content of the `ScrollView` control. The user only sees a 500x400 portion of that rectangle and can scroll to see the rest of it.

ScrollView

XAML

```
<ScrollView Width="500" Height="400">
    <Rectangle Width="1000" Height="800">
        <Rectangle.Fill>
            <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
                <GradientStop Color="Yellow" Offset="0.0" />
                <GradientStop Color="Red" Offset="0.25" />
                <GradientStop Color="Blue" Offset="0.75" />
                <GradientStop Color="LimeGreen" Offset="1.0" />
            </LinearGradientBrush>
        </Rectangle.Fill>
    </Rectangle>
</ScrollView>
```

Layout

In the previous example, the size of the rectangle is explicitly set to be larger than the scroll viewer. In cases were the scroll viewer content is allowed to grow naturally, like in a list or text block, you can configure the scroll viewer to let its content (the *extent*) expand vertically, horizontally, both, or neither.

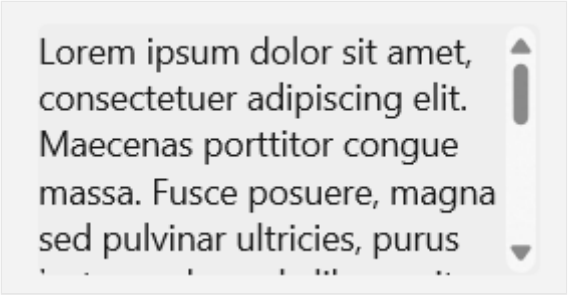
For example, this text block will grow horizontally until its parent container constrains it, then wrap the text and grow vertically.

XAML

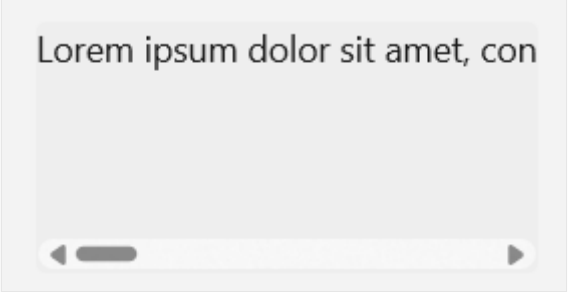
```
<TextBlock Text="{x:Bind someText}" TextWrapping="WrapWholeWords"/>
```

When the text block is wrapped in a scroll viewer, the scroll viewer constrains its horizontal and vertical growth.

Vertical means that the content is constrained horizontally, but can grow vertically beyond the viewport bounds, and the user can scroll the content up and down.



Horizontal means that the content is constrained vertically, but can grow horizontally beyond the viewport bounds, and the user can scroll the content left and right.



Scrollbar visibility

The ScrollViewer and ScrollView controls use slightly different means to configure the horizontal and vertical scrolling of content.

- In the ScrollViewer control, the VerticalScrollBarVisibility and HorizontalScrollBarVisibility properties control both the visibility of the scrollbars and whether scrolling in a particular direction is allowed. When a property is set to `Disabled`, content can't be scrolled in that direction by user interaction.
  - The defaults are: `VerticalScrollBarVisibility="Auto"`, `HorizontalScrollBarVisibility="Disabled"`
- In the ScrollView control, the VerticalScrollBarVisibility and HorizontalScrollBarVisibility properties control only the visibility of the scrollbars.

- The defaults are: `VerticalScrollBarVisibility="Auto"`, `HorizontalScrollBarVisibility="Auto"`

This table describes the visibility options for these properties.

Expand table

Value	Description
Auto	A scrollbar appears only when the viewport cannot display all of the content.
Disabled (ScrollViewer only)	A scrollbar does not appear even when the viewport cannot display all of the content. Scrolling by user interaction is disabled. (Programmatic scrolling is still possible.)
Hidden	A scrollbar does not appear even when the viewport cannot display all of the content. Scrolling is still enabled, and can occur through touch, keyboard, or mouse wheel interaction.
Visible	A scrollbar always appears. (In current UX designs, the scrollbar appears only when the mouse cursor is over it unless the viewport cannot display all of the content.)

(ScrollViewer uses the ScrollBarVisibility enum; ScrollView uses the ScrollingScrollBarVisibility enum.)

Orientation

The ScrollView control has a ContentOrientation property that let's you control the layout of content. This property determines how content can grow when it's not explicitly constrained. If `Height` and `Width` are explicitly set on the content, `ContentOrientation` has no effect.

This table shows the `ContentOrientation` options for ScrollView and the equivalent settings for ScrollViewer.

Expand table

Orientation	ScrollView	ScrollViewer
Vertical	`ContentOrientation="Vertical"`	`VerticalScrollBarVisibility="[Auto][Visible][Hidden]"` `HorizontalScrollBarVisibility="Disabled"`
Horizontal	`ContentOrientation="Horizontal"`	`VerticalScrollBarVisibility="Disabled"` `HorizontalScrollBarVisibility="[Auto]`

Orientation	ScrollView	ScrollViewer
		`[Visible][Hidden]"`
Both	`ContentOrientation="Both"`	`VerticalScrollBarVisibility="[Auto][Visible][Hidden]"` `HorizontalScrollBarVisibility="[Auto][Visible][Hidden]"`
None	`ContentOrientation="None"`	`VerticalScrollBarVisibility="Disabled"` `HorizontalScrollBarVisibility="Disabled"`

Vertical layout

By default, a scroll viewer's content layout (orientation) is vertical.

In this example, an ItemsRepeater is used as the ScrollView's `Content`. The UniformGridLayout for the ItemsRepeater positions the items horizontally in a row until it runs out of space (500px in this example), then positions the next item on the next row. The ItemsRepeater may be taller than the 400px that the user can see, but the user can then scroll the content vertically.

ScrollView

The default `ContentOrientation` value is `Vertical`, so no changes are needed on the ScrollView.

XAML

```
<ScrollView Width="500" Height="400">
    <ItemsRepeater ItemsSource="{x:Bind Albums}"
                   ItemTemplate="{StaticResource MyTemplate}">
        <ItemsRepeater.Layout>
            <UniformGridLayout RowSpacing="8" ColumnSpacing="8"/>
        </ItemsRepeater.Layout>
    </ItemsRepeater>
</ScrollView>
```

Horizontal layout

In this example the content is a `StackPanel` that is laying out its items horizontally. The scroll viewer configuration is changed to support horizontal scrolling and disable vertical scrolling.

ScrollView

The `ScrollView`'s `ContentOrientation` property is set to `Horizontal` to allow the content to grow horizontally as much as needed.

XAML

```
<ScrollView Width="500" Height="400" ContentOrientation="Horizontal">
    <StackPanel Orientation="Horizontal">
        <Button Width="200" Content="Button 1"/>
        <Button Width="200" Content="Button 2"/>
        <Button Width="200" Content="Button 3"/>
        <Button Width="200" Content="Button 4"/>
        <Button Width="200" Content="Button 5"/>
    </StackPanel>
</ScrollView>
```

Programmatic scrolling

The scroll viewer's *offset* properties are read-only, but methods are provided to let you scroll programmatically.

ScrollView

For the ScrollView control, call the ScrollTo method and pass the horizontal and vertical offsets to scroll to. In this case, the scrolling is only vertical, so the current HorizontalOffset value is used. To scroll to the top, a VerticalOffset of 0 is used. To scroll to the bottom, the `VerticalOffset` is the same as the ScrollableHeight.

XAML

```
<Button Content="Scroll to top" Click="ScrollTopButton_Click"/>
<Button Content="Scroll to bottom" Click="ScrollBottomButton_Click"/>
<ScrollView x:Name="scrollView" Width="500" Height="400">
    <StackPanel>
        <Button Width="200" Content="Button 1"/>
        <Button Width="200" Content="Button 2"/>
        <Button Width="200" Content="Button 3"/>
        <Button Width="200" Content="Button 4"/>
        <Button Width="200" Content="Button 5"/>
    </StackPanel>
</ScrollView>
```

C#

```
private void ScrollTopButton_Click(object sender, RoutedEventArgs e)
{
    scrollView.ScrollTo(
        horizontalOffset: scrollView.HorizontalOffset,
        verticalOffset: 0);
}

private void ScrollBottomButton_Click(object sender, RoutedEventArgs e)
{
    scrollView.ScrollTo(
        horizontalOffset: scrollView.HorizontalOffset,
        verticalOffset: scrollView.ScrollableHeight);
}
```

ScrollView also provides a ScrollBy method that lets you scroll vertically or horizontally by a specified delta from the current offset.

Zoom

You can use a scroll viewer to let a user optically zoom in and out of content. Optical zoom interactions are performed through the pinch and stretch gestures (moving fingers farther apart zooms in and moving them closer together zooms out), or by pressing the Ctrl key while scrolling the mouse scroll wheel. For more info about zooming, see Optical zoom and resizing.

To enable zoom by user interaction, set the `ZoomMode` property to `Enabled` (it's `Disabled` by default). Changes to the `ZoomMode` property take effect immediately and may affect an on-going user interaction.

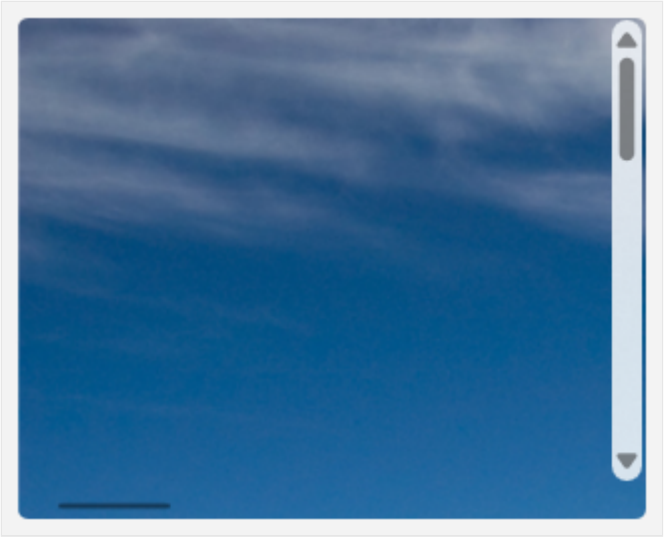
This example shows an Image wrapped in a scroll viewer that's configured to allow zooming.

ScrollView

XAML

```
<ScrollView Width="500" Height="400"
            ContentOrientation="Both"
            ZoomMode="Enabled">
    <Image Source="Assets/rainier.jpg"/>
</ScrollView>
```

In this case, the Image is unconstrained by the scroll viewer, so it's initially shown at its native size. If the image source is larger than the viewport, the user will need to zoom out to see the entire image, which may not be the intended behavior.



The next example shows how to configure the scroll viewer to constrain the image to the viewport so that it's initially loaded zoomed out, and the user can zoom in and scroll if they want to.

ScrollView

To constrain the image to the ScrollView's viewport, set the ContentOrientation property to `None`. Because the scrollbar visibility is not tied to this constraint, scrollbars appear automatically when the user zooms in.

XAML

```
<ScrollView Width="500" Height="400"
            ContentOrientation="None"
            ZoomMode="Enabled">
```

```
    <Image Source="Assets/rainier.jpg"/>
</ScrollView>
```

Zoom factor

Use the `MinZoomFactor` and `MaxZoomFactor` properties to control the amount the user can zoom the content. These properties are effective for both user interactions and programmatic zooming.

- The defaults are: `MinZoomFactor="0.1"`, `MaxZoomFactor="10.0"`

ScrollView

XAML

```
<ScrollView Width="500" Height="400"
            ContentOrientation="Both"
            ZoomMode="Enabled"
            MinZoomFactor="1.0" MaxZoomFactor="8.0">
    <Image Source="Assets/rainier.png"/>
</ScrollView>
```

Programmatic zoom

The `ZoomFactor` property is read-only, but methods are provided to let you zoom programmatically. A typical use for this is to connect the scroll viewer to a Slider that controls the zoom amount, or a button to reset the zoom level. (See ScrollViewer in the WinUI 3 Gallery app to see an example of a zoom slider.)

ScrollView

For the ScrollView control, call the ZoomTo method and pass the new zoom factor as the first parameter.

XAML

```
<Slider Header="Zoom" IsEnabled="True"
        Maximum="{x:Bind scrollControl.MaxZoomFactor, Mode=OneWay}"
        Minimum="{x:Bind scrollControl.MinZoomFactor, Mode=OneWay}"
        StepFrequency="0.1"
        ValueChanged="ZoomSlider_ValueChanged" />
<ScrollView Width="500" Height="400"
            ContentOrientation="None"
```

```
            ZoomMode="Enabled">
    <Image Source="Assets/rainier.png"/>
</ScrollView>
```

C#

```
private void ZoomSlider_ValueChanged(object sender,
RangeBaseValueChangedEventArgs e)
{
    if (scrollControl != null)
    {
        scrollControl.ZoomTo(
            zoomFactor: (float)e.NewValue,
            centerPoint: null)
    }
}
```

ScrollView also provides a ZoomBy method that lets you zoom in and out by a specified delta from the current zoom level.

UWP and WinUI 2

> **Note**
>
> The `ScrollView` control is available only in WinUI 3. For UWP and WinUI 2, use the `ScrollViewer` control.

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** ScrollViewer class, ScrollBar class
- ✓ Open the WinUI 2 Gallery app and see the ScrollViewer in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and

functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

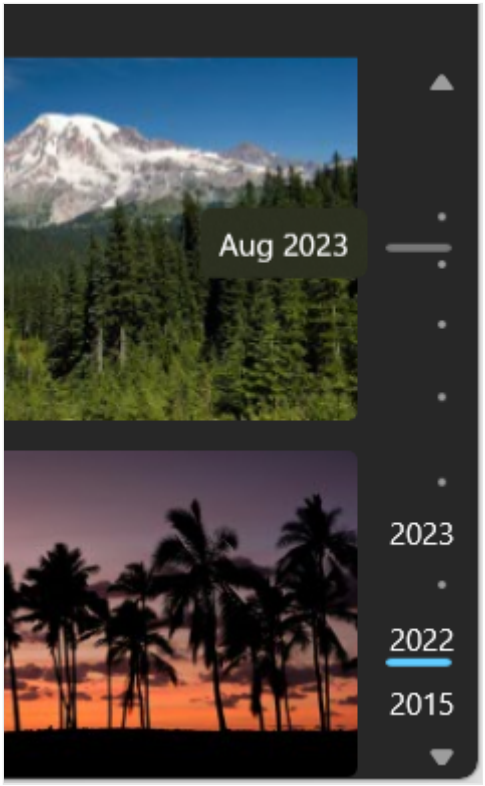
Related topics

- ScrollViewer class

Annotated scrollbar control

Article • 10/01/2023

An annotated scrollbar helps users easily navigate through a large collection of items. It replaces the default scrollbar and can be used in conjunction with any scrollable container. This vertical scrollbar allows users to scroll up and down through the items in a collection. Users can see category labels along the scrollbar area to provide a visual reference of where they are within their collection. A panning indicator (accent-colored line) indicates the user's current position in the collection. A tooltip is revealed when a user hovers on the scrollbar area. This tooltip contains a label to give more information to the user about their current position in the collection. Scroll arrows are located at the top and bottom of the scrollbar area. They can be used to move the current position by a small increment



The annotated scrollbar can be used in conjunction with an ItemsView control to recreate a photo gallery experience. The category labels can be set to different years to help users navigate to a specific location within their photo collection.

Interactions

On hover, a tooltip is revealed to show a preview of that location. Users can then click to navigate to that specific location. Users can also click and drag anywhere on the scrollbar area to navigate to a new position within their collection. As they drag, the

panning indicator stays attached to the mouse cursor and the content scrolls to reflect the new position. Users can also scroll through their collection by using the mouse wheel. Scrolling up or down on the mouse wheel moves their position in the collection and the corresponding panning indicator up or down by a fixed amount.

> **Note**
>
> Unlike in a ScrollBar control, you can't click-and-drag the thumb. The thumb is a non-interactive element that is only for visualizing the current viewport position. The thumb has a fixed height which is not representative of the viewport height.

Label behavior

- For experiences optimized for touch input, users can tap and hold on the scrollbar area to see the tooltip. The tooltip will be slightly more elevated to make it easier to read for touch users. Tapping and dragging anywhere on the scrollbar area will work similarly to a click and drag with mouse.
- The tooltip label is never truncated as its purpose is to display text that's explanatory. Instead, the text is wrapped if it goes over the tooltip's max width of 360px.
- Category labels are clipped if the text is longer than the width of the scrollbar's area.
- When there is a collision between two category labels (for example, when the window size decreases and two labels overlap each other), the top label is removed. As an exception to this rule, the first label of the collection is always kept as it helps to indicate the range of the collection to users. Category labels should always have a minimum of 4px (2px above, 2px below) in between, otherwise a label-collision is triggered.

Is this the right control?

We recommend using an annotated scrollbar, rather than a default scrollbar, when dealing with a large collection of items or when a large amount of scrolling is expected. The annotated scrollbar will provide an easy way for users to locate themselves in their collection and to traverse it.

For collections that only have a few items or that only require a small amount of scrolling, we recommend using a default scrollbar.

Recommendations

- Only add a category label if it provides helpful information to users
- Keep the strings used for the category labels and the tooltip as concise as possible
- The annotated scrollbar is best used when there is sufficient vertical space. Using the annotated scrollbar in a confined vertical space will reduce the number of visible labels
- Do not use the annotated scrollbar as the only way for users to locate themselves within a collection. We recommend using sticky headers or category labels throughout your collection to complement the annotated scrollbar.

Create an annotated scrollbar

✓ **Important APIs:** AnnotatedScrollBar class, ScrollView class, IScrollController interface

Open the WinUI 3 Gallery app and see the AnnotatedScrollBar in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

To use an AnnotatedScrollBar, you need to complete several steps:

- Connect the AnnotatedScrollBar to a scrolling container.
- Add labels to the scrollbar.
- Add detail labels (tooltips).

Connect the AnnotatedScrollBar to a scrolling container

To use an AnnotatedScrollBar, you connect it to a scrollable container via the IScrollController interface. The AnnotatedScrollBar provides an IScrollController implementation through its ScrollController property, while the ScrollView consumes it through its ScrollPresenter property (specifically, ScrollPresenter.VerticalScrollController).

> ⓘ **Note**
>
> The ScrollView control has built in support for consuming the IScrollController interface. For other scrolling containers, like ScrollViewer, you must write an adapter for IScrollController. See an example later in this article.

Here, the VerticalScrollController property of an ItemsView is bound to the ScrollController of an AnnotatedScrollBar. (The ItemsView.VerticalScrollPresenter property is passed through to the ScrollPresenter.VerticalScrollController value of the ItemsView's inner ScrollView.)

XAML

```
<Grid ColumnDefinitions="*,Auto">
    <ItemsView VerticalScrollController="{x:Bind
annotatedScrollBar.ScrollController}"/>
    <AnnotatedScrollBar x:Name="annotatedScrollBar" Grid.Column="1"/>
</Grid>
```

You can also connect them in code, as shown here. In this example, a ScrollView is used to wrap an ItemsRepeater and provide scrolling functionality for it. The AnnotatedScrollBar replaces the ScrollView's default scrollbar.

XAML

```
<Grid ColumnDefinitions="*, Auto">
    <ScrollView x:Name="scrollView"
                Background="LightGray"
                MaxWidth="800" MaxHeight="500"
                VerticalScrollBarVisibility="Hidden">
        <ItemsRepeater x:Name="itemsRepeater"
                ItemsSource="{x:Bind ColorCollection}"
                Margin="2"
                SizeChanged="ItemsRepeater_SizeChanged">
            <ItemsRepeater.Layout>
                <UniformGridLayout/>
            </ItemsRepeater.Layout>
            <ItemsRepeater.ItemTemplate>
                <DataTemplate x:DataType="media:SolidColorBrush">
                    <Grid Background="{x:Bind}" Width="112" Height="82"
                          CornerRadius="4" Margin="4"/>
                </DataTemplate>
            </ItemsRepeater.ItemTemplate>
        </ItemsRepeater>
    </ScrollView>

    <AnnotatedScrollBar x:Name="annotatedScrollBar" Grid.Column="1"
        Margin="4,0,48,0" MaxHeight="500" HorizontalAlignment="Right"
        DetailLabelRequested="AnnotatedScrollBar_DetailLabelRequested"/>
</Grid>
```

C#

```
private void AnnotatedScrollBarPage_Loaded(object sender,
Microsoft.UI.Xaml.RoutedEventArgs e)
```

```
{
    scrollView.ScrollPresenter.VerticalScrollController =
annotatedScrollBar.ScrollController;
}
```

Labels

The AnnotatedScrollBar can display two kinds of labels, both of which are optional.

Category labels

You add labels by populating the Labels collection. Each Label is represented by the AnnotatedScrollBarLabel class and requires two pieces of information:

- Content: The content that is displayed on the scrollbar.
- ScrollOffset: The offset value at which the label content is displayed.

Labels (if specified) are always visible on the scrollbar, unless they collide with other labels or extend past the bounds of the rail. (See Label behavior for more info.)

The calculation of the label offset value depends on the details of your app and its data. For one example of how the offset can be calculated, see the WinUI Gallery example on GitHub.

Detail labels

A detail label is a tooltip element that is shown when the cursor is over the scrollbar. To create a detail label, you handle the DetailLabelRequested event. The event args provide the ScrollOffset where the tooltip label will be shown. Use this value to determine the correct label to show for that position, and set the label as the Content property of the event args.

Scrolling

The user can scroll the AnnotatedScrollBar by clicking the arrow buttons at the top and bottom of the scrollbar. You can set the SmallChange property to specify the amount by which the arrow buttons scroll the content.

You can handle the Scrolling event to determine how the scrolling is being performed, take an action when scrolling occurs, or to Cancel the scroll action.

Get the sample code

- WinUI Gallery sample - See all the XAML controls in an interactive format.

Related topics

- ScrollView class

Semantic zoom

Article • 02/26/2025

Semantic zoom lets the user switch between two different views of the same content so that they can quickly navigate through a large set of grouped data.

- The zoomed-in view is the main view of the content. This is the main view where you show individual data items.
- The zoomed-out view is a higher-level view of the same content. You typically show the group headers for a grouped data set in this view.

For example, when viewing an address book, the user could zoom out to quickly jump to the letter "W", and zoom in on that letter to see the names associated with it.

Features:

- The size of the zoomed-out view is constrained by the bounds of the semantic zoom control.
- Tapping on a group header toggles views. Pinching as a way to toggle between views can be enabled.
- Active headers switch between views.

Is this the right control?

Use a **SemanticZoom** control when you need to show a grouped data set that's large enough that it can't all be shown on one or two pages.

Don't confuse semantic zooming with optical zooming. While they share both the same interaction and basic behavior (displaying more or less detail based on a zoom factor), optical zoom refers to the adjustment of magnification for a content area or object such as a photograph. For info about a control that performs optical zooming, see the ScrollViewer control.

Recommendations

- When using semantic zoom in your app, be sure that the item layout and panning direction don't change based on the zoom level. Layouts and panning interactions should be consistent and predictable across zoom levels.
- Semantic zoom enables the user to jump quickly to content, so limit the number of pages/screens to three in the zoomed-out mode. Too much panning diminishes the practicality of semantic zoom.

- Avoid using semantic zoom to change the scope of the content. For example, a photo album shouldn't switch to a folder view in File Explorer.
- Use a structure and semantics that are essential to the view.
- Use group names for items in a grouped collection.
- Use sort ordering for a collection that is ungrouped but sorted, such as chronological for dates or alphabetical for a list of names.

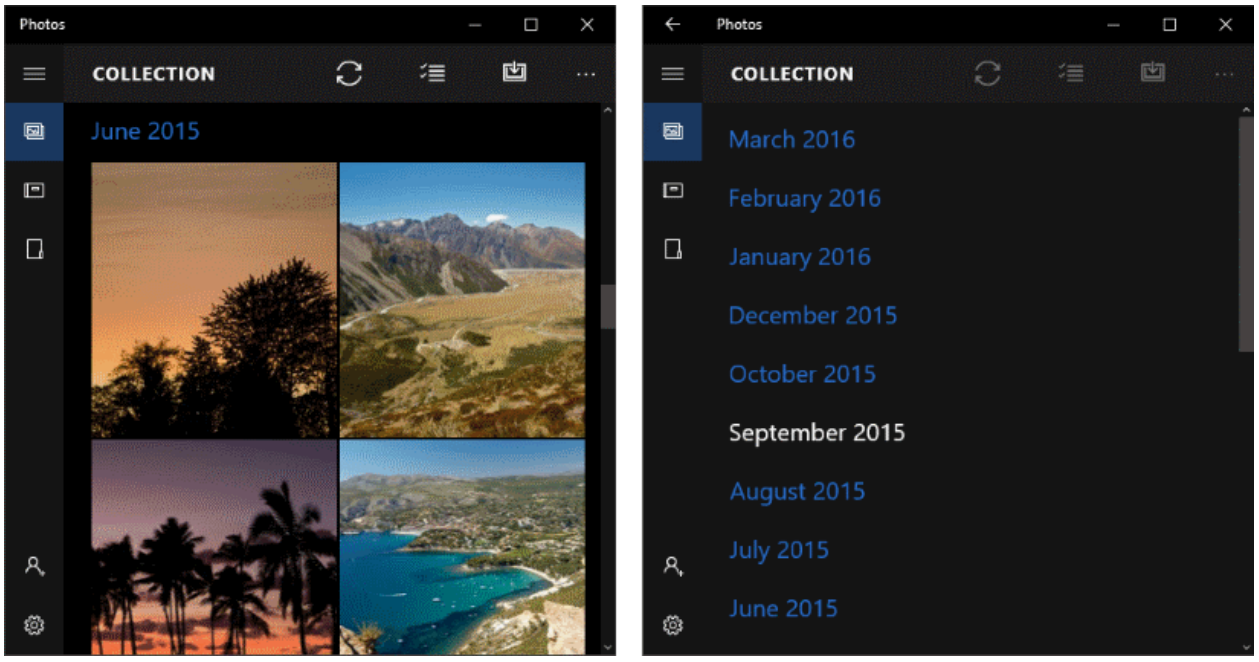
Examples

WinUI Gallery

The SemanticZoom section within the Controls Gallery demonstrates a navigation experience that allows users to quickly zoom in and out of grouped sections of control types.

Photos app

Here's a semantic zoom used in the Photos app. Photos are grouped by month. Selecting a month header in the default grid view zooms out to the month list view for quicker navigation.



Create a semantic zoom

✓ **Important APIs:** SemanticZoom class, ListView class, GridView class

Open the WinUI 3 Gallery app and see SemanticZoom in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

The **SemanticZoom** control doesn't have any visual representation of its own. It's a host control that manages the transition between 2 other controls that provide the views of your content, typically **ListView** or **GridView** controls. You set the view controls to the ZoomedInView and ZoomedOutView properties of the SemanticZoom.

The 3 elements you need for a semantic zoom are:

- A grouped data source. (Groups are defined by the GroupStyle definition in the zoomed-in view.)
- A zoomed-in view that shows the item-level data.
- A zoomed-out view that shows the group-level data.

Before you use a semantic zoom, you should understand how to use a list view with grouped data. For more info, see List view and grid view.

> ⓘ **Note**

To define the zoomed-in view and the zoomed-out view of the SemanticZoom control, you can use any two controls that implement the ISemanticZoomInformation interface. The XAML framework provides 3 controls that implement this interface: ListView, GridView, and Hub.

This XAML shows the structure of the SemanticZoom control. You assign other controls to the ZoomedInView and ZoomedOutView properties.

XAML

```
<SemanticZoom>
   <SemanticZoom.ZoomedInView>
       <!-- Put the GridView for the zoomed in view here. -->
   </SemanticZoom.ZoomedInView>

   <SemanticZoom.ZoomedOutView>
       <!-- Put the ListView for the zoomed out view here. -->
   </SemanticZoom.ZoomedOutView>
</SemanticZoom>
```

The examples here are taken from the SemanticZoom page of the WinUI Gallery sample. You can download the sample to see the complete code including the data source. This semantic zoom uses a GridView to supply the zoomed-in view and a ListView for the zoomed-out view.

Define the zoomed-in view

Here's the GridView control for the zoomed-in view. The zoomed-in view should display the individual data items in groups. This example shows how to display the items in a grid with an image and text.

XAML

```
<SemanticZoom.ZoomedInView>
    <GridView ItemsSource="{x:Bind cvsGroups.View}"
              ScrollViewer.IsHorizontalScrollChainingEnabled="False"
              SelectionMode="None"
              ItemTemplate="{StaticResource ZoomedInTemplate}">
        <GridView.GroupStyle>
            <GroupStyle HeaderTemplate="{StaticResource
ZoomedInGroupHeaderTemplate}"/>
        </GridView.GroupStyle>
    </GridView>
</SemanticZoom.ZoomedInView>
```

The look of the group headers is defined in the `ZoomedInGroupHeaderTemplate` resource. The look of the items is defined in the `ZoomedInTemplate` resource.

XAML

```
<DataTemplate x:Key="ZoomedInGroupHeaderTemplate"
x:DataType="data:ControlInfoDataGroup">
    <TextBlock Text="{x:Bind Title}"
               Foreground="{ThemeResource ApplicationForegroundThemeBrush}"
               Style="{StaticResource SubtitleTextBlockStyle}"/>
</DataTemplate>

<DataTemplate x:Key="ZoomedInTemplate"
x:DataType="data:ControlInfoDataItem">
    <StackPanel Orientation="Horizontal" MinWidth="200" Margin="12,6,0,6">
        <Image Source="{x:Bind ImagePath}" Height="80" Width="80"/>
        <StackPanel Margin="20,0,0,0">
            <TextBlock Text="{x:Bind Title}"
                       Style="{StaticResource BaseTextBlockStyle}"/>
            <TextBlock Text="{x:Bind Subtitle}"
                       TextWrapping="Wrap" HorizontalAlignment="Left"
                       Width="300" Style="{StaticResource
BodyTextBlockStyle}"/>
        </StackPanel>
    </StackPanel>
</DataTemplate>
```

Define the zoomed-out view

This XAML defines a ListView control for the zoomed-out view. This example shows how to display the group headers as text in a list.

XAML

```
<SemanticZoom.ZoomedOutView>
    <ListView ItemsSource="{x:Bind cvsGroups.View.CollectionGroups}"
              SelectionMode="None"
              ItemTemplate="{StaticResource ZoomedOutTemplate}" />
</SemanticZoom.ZoomedOutView>
```

The look is defined in the `ZoomedOutTemplate` resource.

XAML

```
<DataTemplate x:Key="ZoomedOutTemplate"
x:DataType="wuxdata:ICollectionViewGroup">
    <TextBlock Text="{x:Bind Group.(data:ControlInfoDataGroup.Title)}"
               Style="{StaticResource SubtitleTextBlockStyle}"
```

```
TextWrapping="Wrap"/>
</DataTemplate>
```

Synchronize the views

The zoomed-in view and zoomed-out view should be synchronized, so if a user selects a group in the zoomed-out view, the details of that same group are shown in the zoomed-in view. You can use a CollectionViewSource or add code to synchronize the views.

Any controls that you bind to the same CollectionViewSource always have the same current item. If both views use the same CollectionViewSource as their data source, the CollectionViewSource synchronizes the views automatically. For more info, see CollectionViewSource.

If you don't use a CollectionViewSource to synchronize the views, you should handle the ViewChangeStarted event and synchronize the items in the event handler, as shown here.

XAML

```
<SemanticZoom x:Name="semanticZoom"
ViewChangeStarted="SemanticZoom_ViewChangeStarted">
```

C#

```
private void SemanticZoom_ViewChangeStarted(object sender,
SemanticZoomViewChangedEventArgs e)
{
    if (e.IsSourceZoomedInView == false)
    {
        e.DestinationItem.Item = e.SourceItem.Item;
    }
}
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that

use **WinUI 2**. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** SemanticZoom class, ListView class, GridView class
- ✓ Open the WinUI 2 Gallery app and see the SemanticZoom in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls.

Related articles

- Navigation design basics
- List view and grid view
- Item containers and templates

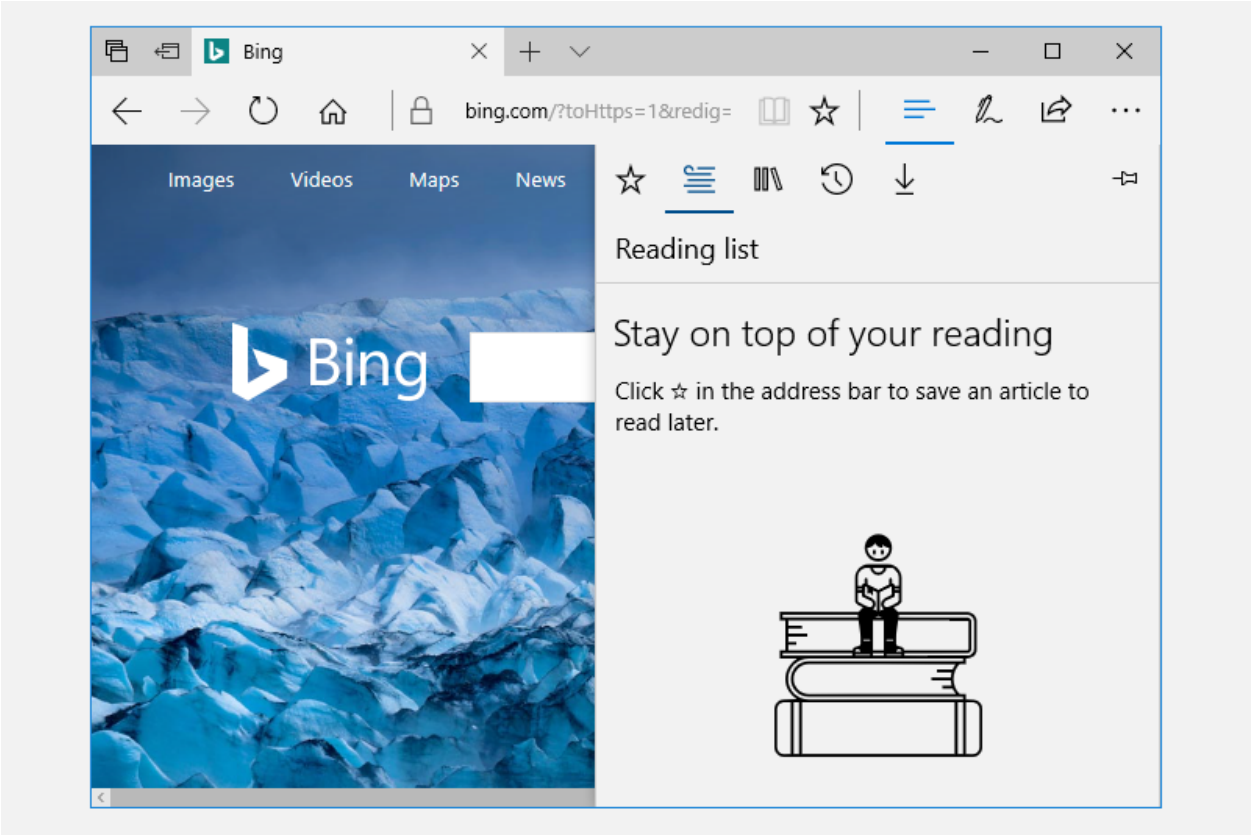
Split view control

Article • 06/09/2022

A split view control has an expandable/collapsible pane and a content area.

> **Important APIs**: SplitView class

Here is an example of the Microsoft Edge app using SplitView to show its Hub.



A split view's content area is always visible. The pane can expand and collapse or remain in an open state, and can present itself from either the left side or right side of an app window. The pane has four modes:

- **Overlay**

  The pane is hidden until opened. When open, the pane overlays the content area.

- **Inline**

  The pane is always visible and doesn't overlay the content area. The pane and content areas divide the available screen real estate.

- **CompactOverlay**

A narrow portion of the pane is always visible in this mode, which is just wide enough to show icons. The default closed pane width is 48px, which can be modified with `CompactPaneLength`. If the pane is opened, it will overlay the content area.

- **CompactInline**

  A narrow portion of the pane is always visible in this mode, which is just wide enough to show icons. The default closed pane width is 48px, which can be modified with `CompactPaneLength`. If the pane is opened, it will reduce the space available for content, pushing the content out of its way.

Is this the right control?

The split view control can be used to create any "drawer" experience where users can open and close the supplemental pane. For example, you can use SplitView to build the list/details pattern.

If you'd like to build a navigation menu with an expand/collapse button and a list of navigation items, then use the NavigationView control.

Examples

WinUI 2 Gallery

If you have the **WinUI 2 Gallery** app installed, click here to open the app and see the SplitView in action.

- Get the WinUI 2 Gallery app (Microsoft Store)
- Get the source code (GitHub)

Create a split view

Here's a SplitView control with an open Pane appearing inline next to the Content.

XAML

```
<SplitView IsPaneOpen="True"
           DisplayMode="Inline"
           OpenPaneLength="296">
    <SplitView.Pane>
        <TextBlock Text="Pane"
                   FontSize="24"
```

```
                    VerticalAlignment="Center"
                    HorizontalAlignment="Center"/>
    </SplitView.Pane>

    <Grid>
        <TextBlock Text="Content"
                   FontSize="24"
                   VerticalAlignment="Center"
                   HorizontalAlignment="Center"/>
    </Grid>
</SplitView>
```

Get the sample code

- WinUI 2 Gallery sample - See all the XAML controls in an interactive format.

Related topics

- Nav pane pattern
- List view
- List/details

Two-pane view

Article • 06/09/2022

TwoPaneView is a layout control that helps you manage the display of apps that have 2 distinct areas of content, like a list/detail view.

> **Important**
>
> This article describes functionality and guidance that is in public preview and may be substantially modified before it's generally available. Microsoft makes no warranties, express or implied, with respect to the information provided here.

While it works on all Windows devices, the TwoPaneView control is designed to help you take full advantage of dual-screen devices automatically, with no special coding needed. On a dual-screen device, the two-pane view ensures that the user interface (UI) is split cleanly when it spans the gap between screens, so that your content is presented on either side of the gap.

Get WinUI

The **TwoPaneView** control is included as part of WinUI, a NuGet package that contains new controls and UI features for Windows apps. For more info, including installation instructions, see WinUI.

> **WinUI APIs:** TwoPaneView class

> **Tip**
>
> Throughout this document, we use the **muxc** alias in XAML to represent WinUI APIs that we have included in our project. We have added this to our **Page** element: `xmlns:muxc="using:Microsoft.UI.Xaml.Controls"`
>
> In the code-behind, we also use the **muxc** alias in C# to represent WinUI APIs that we have included in our project. We have added this **using** statement at the top of the file: `using muxc = Microsoft.UI.Xaml.Controls;`

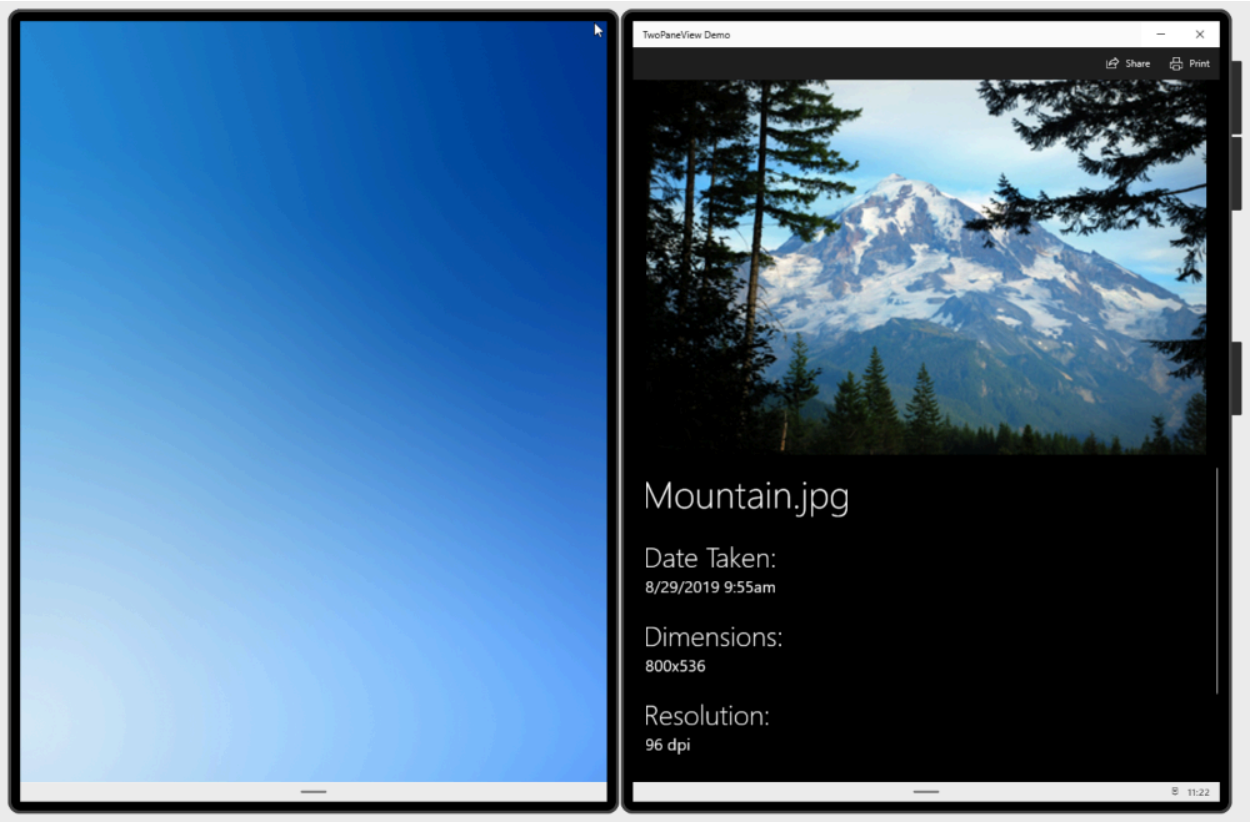
Is this the right control?

Use the two-pane view when you have 2 distinct areas of content and:

- The content should automatically rearrange and resize to best fit the window.
- The secondary area of content should show/hide based on available space.
- The content should split cleanly between the 2 screens of a dual-screen device.

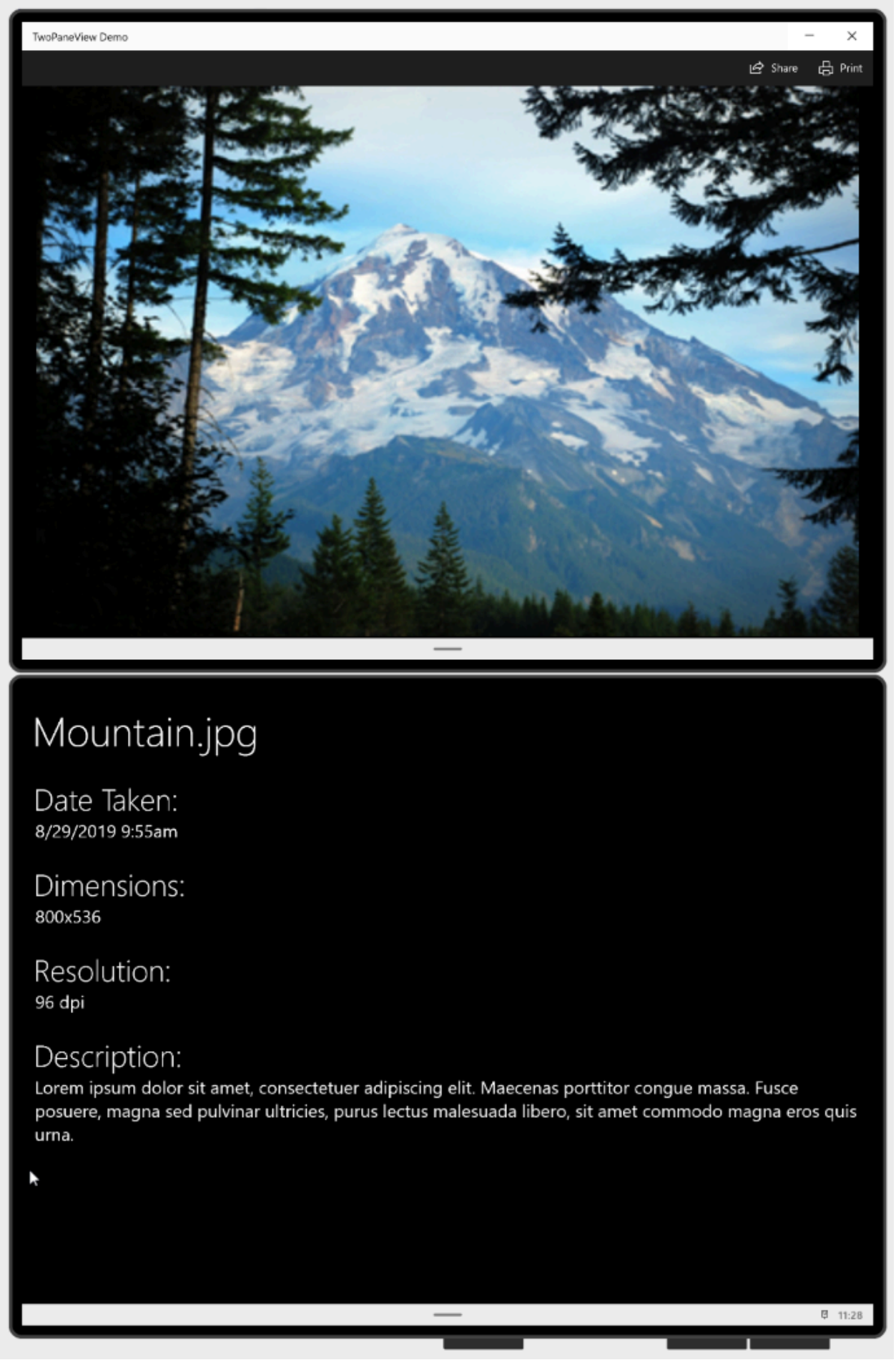
Examples

These images show an app running on a single-screen and spanned across dual-screens. The two-pane view adapts the app UI to the various screen configurations.



> *App on a single-screen.*

> *App spanning a dual-screen device in wide mode.*



> *App spanning a dual-screen device in tall mode.*

How it works

The two-pane view has two panes where you place your content. It adjusts the size and arrangement of the panes depending on the space available to the window. The possible pane layouts are defined by the TwoPaneViewMode enumeration:

Expand table

Enum value	Description
`SinglePane`	Only one pane is shown, as specified by the PanePriority property.
`Wide`	Panes are shown side-by-side, or a single pane is shown, as specified by the WideModeConfiguration property.
`Tall`	Panes are shown top-bottom, or a single pane is shown, as specified by the TallModeConfiguration property.

You configure the two-pane view by setting the PanePriority to specify which pane is shown when there is space for only one pane. Then, you specify whether `Pane1` is shown on the top or bottom for tall windows, or on the left or right for wide windows.

The two-pane view handles the size and arrangement of the panes, but you still need to make the content inside the pane adapt to the changes in size and orientation. See Responsive layouts with XAML and Layout panels for more info about creating an adaptive UI.

The TwoPaneView manages the display of the panes based on the spanning state of the app.

- On a single-screen

  When your app is on only a single screen, the `TwoPaneView` adjusts the size and position of its panes based on property settings you specify. We explain these properties in more detail in the next section. The only difference between devices is that some devices, like desktop PCs, allow re-sizable windows, while other devices don't.

- Spanned across dual-screens

  The `TwoPaneView` is designed to make it easy to optimize your UI for spanning on dual-screen devices. The window sizes itself to use all the available space on the screens. When your app spans both screens of a dual-screen device, each screen displays the content of one of the panes and properly spans content across the gap. Spanning-awareness is built-in when you use two-pane view. You only need

to set the tall/wide configuration to specify which pane is shown on which screen. The two-pane view takes care of the rest.

How to use the two-pane view control

The TwoPaneView doesn't have to be the root element of your page layout. In fact, you'll often use it inside a NavigationView control that provides the overall navigation for your app. The `TwoPaneView` adapts appropriately regardless of where it is in the XAML tree; however, we do recommend that you not nest a `TwoPaneView` inside another `TwoPaneView`. (If you do, only the outer `TwoPaneView` is spanning-aware.)

Add content to the panes

Each pane of a two-pane view can hold a single XAML `UIElement`. To add content, you typically place a XAML layout panel in each pane, and then add other controls and content to the panel. The panes can change size and switch between wide and tall modes, so you need to make sure the content in each pane can adapt to these changes. See Responsive layouts with XAML and Layout panels for more info about creating an adaptive UI.

This example creates the simple picture/info app UI shown previously in the *Examples* section. When the app is spanned across dual-screens, the picture and the info are shown on separate screens. On a single screen, the content can be shown in two panes, or combined into a single pane, depending on how much space is available. (When there's only space for one pane, you move the content of Pane2 into Pane1, and let the user scroll to see any hidden content. You'll see the code for this later in the *Responding to mode changes* section.)

XAML

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" MinHeight="40"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <CommandBar DefaultLabelPosition="Right">
        <AppBarButton x:Name="Share" Icon="Share" Label="Share"
Click="Share_Click"/>
        <AppBarButton x:Name="Print" Icon="Print" Label="Print"
Click="Print_Click"/>
    </CommandBar>

    <muxc:TwoPaneView
        x:Name="MyTwoPaneView"
        Grid.Row="1"
        MinWideModeWidth="959"
        MinTallModeHeight="863"
        ModeChanged="TwoPaneView_ModeChanged">

        <muxc:TwoPaneView.Pane1>
            <Grid x:Name="Pane1Root">
                <ScrollViewer>
                    <StackPanel x:Name="Pane1StackPanel">
                        <Image Source="Assets\LandscapeImage8.jpg"
                               VerticalAlignment="Top"
HorizontalAlignment="Center"
                               Margin="16,0"/>
                    </StackPanel>
                </ScrollViewer>
            </Grid>
        </muxc:TwoPaneView.Pane1>

        <muxc:TwoPaneView.Pane2>
            <Grid x:Name="Pane2Root">
                <ScrollViewer x:Name="DetailsContent">
                    <StackPanel Padding="16">
                        <TextBlock Text="Mountain.jpg" MaxLines="1"
                                   Style="{ThemeResource
HeaderTextBlockStyle}"/>
                        <TextBlock Text="Date Taken:"
                                   Style="{ThemeResource
SubheaderTextBlockStyle}"
                                   Margin="0,24,0,0"/>
                        <TextBlock Text="8/29/2019 9:55am"
                                   Style="{ThemeResource
SubtitleTextBlockStyle}"/>
                        <TextBlock Text="Dimensions:"
                                   Style="{ThemeResource
SubheaderTextBlockStyle}"
                                   Margin="0,24,0,0"/>
                        <TextBlock Text="800x536"
                                   Style="{ThemeResource
SubtitleTextBlockStyle}"/>
```

```
                        <TextBlock Text="Resolution:"
                                   Style="{ThemeResource
SubheaderTextBlockStyle}"
                                   Margin="0,24,0,0"/>
                        <TextBlock Text="96 dpi"
                                   Style="{ThemeResource
SubtitleTextBlockStyle}"/>
                        <TextBlock Text="Description:"
                                   Style="{ThemeResource
SubheaderTextBlockStyle}"
                                   Margin="0,24,0,0"/>
                        <TextBlock Text="Lorem ipsum dolor sit amet,
consectetuer adipiscing elit. Maecenas porttitor congue massa. Fusce
posuere, magna sed pulvinar ultricies, purus lectus malesuada libero, sit
amet commodo magna eros quis urna."
                                   Style="{ThemeResource
SubtitleTextBlockStyle}"
                                   TextWrapping="Wrap"/>
                    </StackPanel>
                </ScrollViewer>
            </Grid>
        </muxc:TwoPaneView.Pane2>
    </muxc:TwoPaneView>

    <VisualStateManager.VisualStateGroups>
        <VisualStateGroup x:Name="TwoPaneViewStates">
            <VisualState x:Name="Normal"/>
            <VisualState x:Name="Wide">
                <VisualState.Setters>
                    <Setter Target="MyTwoPaneView.Pane1Length"
                            Value="2*"/>
                </VisualState.Setters>
            </VisualState>
        </VisualStateGroup>
    </VisualStateManager.VisualStateGroups>
</Grid>
```

Specify which pane to display

When the two-pane view can only display a single pane, it uses the PanePriority property to determine which pane to display. By default, PanePriority is set to **Pane1**. Here's how you can set this property in XAML or in code.

XAML

```
<muxc:TwoPaneView x:Name="MyTwoPaneView" PanePriority="Pane2">
```

C#

```
MyTwoPaneView.PanePriority =
Microsoft.UI.Xaml.Controls.TwoPaneViewPriority.Pane2;
```

Pane sizing

On a single screen, the size of the panes is determined by the Pane1Length and Pane2Length properties. These use GridLength values that support *auto* and *star*(*) sizing. See the *Layout properties* section of Responsive layouts with XAML for an explanation of auto and star sizing.

By default, `Pane1Length` is set to `Auto` and it sizes itself to fit its content. `Pane2Length` is set to `*` and it uses all the remaining space.

> *Panes with default sizing*

The default values are useful for a typical list/detail layout, where you have a list of items in `Pane1`, and a lot of details in `Pane2`. However, depending on your content, you might prefer to divide the space differently. Here, `Pane1Length` is set to `2*` so it gets twice as much space as `Pane2`.

XAML

```
<muxc:TwoPaneView x:Name="MyTwoPaneView" Pane1Length="2*">
```

Panes sized 2 and **

> **Note**
>
> As mentioned previously, when the app is spanned across dual-screens, these properties are ignored and each pane fills one of the screens.

If you set a pane to use auto sizing, you can control the size by setting the height and width of the Panel that holds the pane's content. In this case, you might need to handle the ModeChanged event and set the height and width constraints of the content as appropriate for the current mode.

Display in wide or tall mode

On a single screen, the two-pane view's display Mode is determined by the MinWideModeWidth and MinTallModeHeight properties. Both properties have a default value of 641px, the same as NavigationView.CompactThresholdWidth.

This table shows how the Height and Width of the TwoPaneView determine which display mode is used.

Expand table

TwoPaneView condition	Mode
`Width` > `MinWideModeWidth`	`Wide` mode is used
`Width` <= `MinWideModeWidth`, and `Height` > `MinTallModeHeight`	`Tall` mode is used
`Width` <= `MinWideModeWidth`, and `Height` <= `MinTallModeHeight`	`SinglePane` mode is used

> **Note**

As mentioned previously, when the app is spanned across dual-screens, these properties are ignored and display mode is determined based on the device *posture*.

Wide configuration options

The two-pane view enters `Wide` mode when there's a single display that's wider than the `MinWideModeWidth` property. `MinWideModeWidth` controls when the two-pane view enters wide mode. The default value is 641px, but you can change it to whatever you want. In general, you should set this property to whatever you want the minimum width of your pane to be.

When the two-pane view is in wide mode, the WideModeConfiguration property determines what to show:

Expand table

Enum value	Description
`SinglePane`	A single pane (as determined by `PanePriority`). The pane takes up the full size of the `TwoPaneView` (ie, it's star sized in both directions).
`LeftRight`	`Pane1` on the left/`Pane2` on the right. Both panes are star sized vertically, `Pane1`'s width is autosized, and `Pane2`'s width is star sized.
`RightLeft`	`Pane1` on the right/`Pane2` on the left. Both panes are star sized vertically, `Pane2`'s width is autosized, and `Pane1`'s width is star sized.

The default setting is `LeftRight`.

Expand table

LeftRight	RightLeft

> **TIP:** When the device uses a right-to-left (RTL) language, the two-pane view automatically swaps the order: `RightLeft` renders as `LeftRight`, and `LeftRight` renders as `RightLeft`.

Tall configuration options

The two-pane view enters `Tall` mode when there's a single display that's narrower than `MinWideModeWidth`, and taller than `MinTallModeHeight`. The default value is 641px, but you can change it to whatever you want. In general, you should set this property to whatever you want the minimum height of your pane to be.

When the two-pane view is in tall mode, the TallModeConfiguration property determines what to show:

Expand table

Enum value	Description
`SinglePane`	A single pane (as determined by `PanePriority`). The pane takes up the full size of the `TwoPaneView` (ie, it's star sized in both directions).
`TopBottom`	`Pane1` on the top/`Pane2` on the bottom. Both panes are star sized horizontally, `Pane1`'s height is autosized, and `Pane2`'s height is star sized.
`BottomTop`	`Pane1` on the bottom/`Pane2` on the top. Both panes are star sized horizontally, `Pane2`'s height is autosized, and `Pane1`'s height is star sized.

The default is `TopBottom`.

Expand table

TopBottom	BottomTop

Special values for MinWideModeWidth and MinTallModeHeight

You can use the `MinWideModeWidth` property to prevent the two-pane view from entering Wide mode - just set `MinWideModeWidth` to Double.PositiveInfinity.

If you set `MinTallModeHeight` to Double.PositiveInfinity, it prevents the two-pane view from entering Tall mode.

If you set `MinTallModeHeight` to 0, it prevents the two-pane view from entering `SinglePane` mode.

Responding to mode changes

You can use the read-only Mode property to get the current display mode. Whenever the two-pane view changes which pane or panes it's displaying, the ModeChanged event occurs before it renders the updated content. You can handle the event to respond to changes in the display mode.

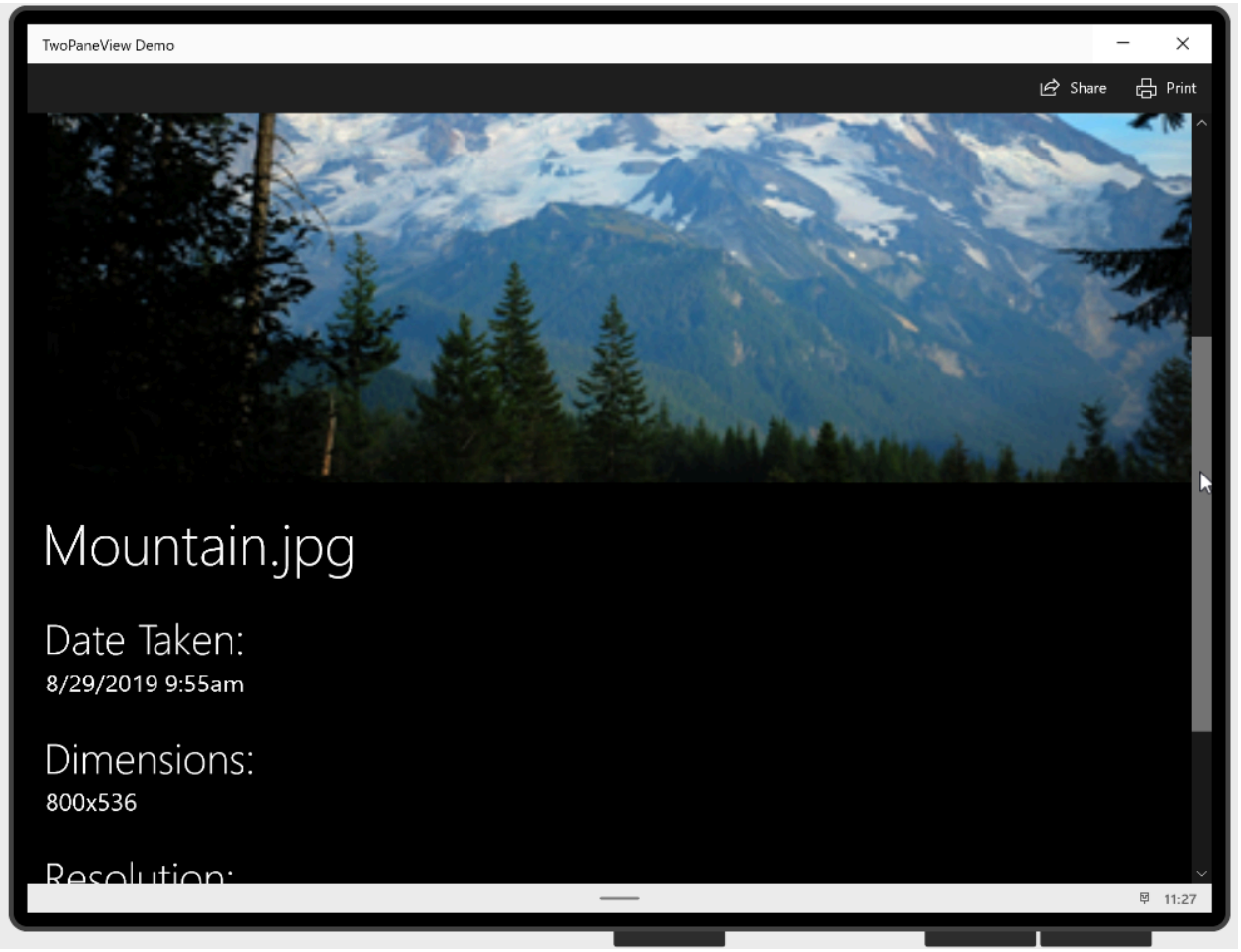
> **Tip**

The `ModeChanged` event does not occur when the page is initially loaded, so your default XAML should represent the UI as it should appear when first loaded.

One way you can use this event is to update your app's UI so users can view all the content in `SinglePane` mode. For example, the example app has a primary pane (the image) and an info pane.

> *Tall mode*

When there's only enough space to display one pane, you can move the content of `Pane2` into `Pane1` so the user can scroll to see all the content. It looks like this.



SinglePane mode

Remember that the `MinWideModeWidth` and `MinTallModeHeight` properties determine when the display mode changes, so you can change when the content is moved between panes by adjusting the values of these properties.

Here's the `ModeChanged` event handler code that moves the content between `Pane1` and `Pane2`. It also sets a VisualState to constrain the width of the image in Wide mode.

C#

```
private void TwoPaneView_ModeChanged(Microsoft.UI.Xaml.Controls.TwoPaneView
sender, object args)
{
    // Remove details content from it's parent panel.
    ((Panel)DetailsContent.Parent).Children.Remove(DetailsContent);
    // Set Normal visual state.
    Windows.UI.Xaml.VisualStateManager.GoToState(this, "Normal", true);

    // Single pane
    if (sender.Mode ==
Microsoft.UI.Xaml.Controls.TwoPaneViewMode.SinglePane)
    {
        // Add the details content to Pane1.
        Pane1StackPanel.Children.Add(DetailsContent);
```

```
    }
    // Dual pane.
    else
    {
        // Put details content in Pane2.
        Pane2Root.Children.Add(DetailsContent);

        // If also in Wide mode, set Wide visual state
        // to constrain the width of the image to 2*.
        if (sender.Mode == Microsoft.UI.Xaml.Controls.TwoPaneViewMode.Wide)
        {
            Windows.UI.Xaml.VisualStateManager.GoToState(this, "Wide",
true);
        }
    }
}
```

Dos and don'ts

- Do use the two-pane view whenever you can so that your app can take advantage of dual-screens and large screens.
- Don't put a two-pane view inside another two-pane view.

Related articles

- Layout overview
- Dual-screen development
- Introduction to dual-screen devices

Progress controls

Article • 02/26/2025

A progress control provides feedback to the user that a long-running operation is underway. It can mean that the user cannot interact with the app when the progress indicator is visible, and can also indicate how long the wait time might be, depending on the indicator used.

Types of progress

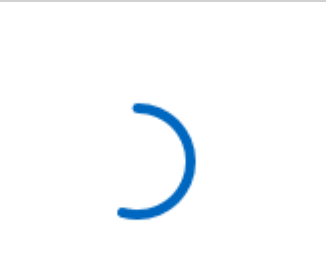
There are two controls to show the user that an operation is underway – either through a ProgressBar or through a ProgressRing. Both ProgressBar and ProgressRing have two states that communicate if the user can interact with the application or not.

- The *determinate* state for both ProgressBar and ProgressRing shows the percentage completed of a task. This should be used during an operation whose duration is known, but its progress should not block the user's interaction with the app.
- The *indeterminate* state for ProgressBar shows that an operation is underway, does not block user interaction with the app, and its completion time is unknown.
- The *indeterminate* state for ProgressRing shows that an operation is underway, blocks user interaction with the app, and its completion time is unknown.

Additionally, a progress control is read only and not interactive. Meaning that the user cannot invoke or use these controls directly.

Expand table

Control	Display
Indeterminate ProgressBar	
Determinate ProgressBar	

Control	Display
Indeterminate ProgressRing	
Determinate ProgressRing	

Is this the right control?

It's not always obvious what control or what state (determinate vs indeterminate) to use when trying to show something is happening. Sometimes a task is obvious enough that it doesn't require a progress control at all – and sometimes even if a progress control is used, a line of text is still necessary in order to explain to the user what operation is underway.

ProgressBar

- **Does the control have a defined duration or predictable end?**

  Use a determinate ProgressBar then, and update the percentage or value accordingly.

- **Can the user continue without having to monitor the operation's progress?**

  When a ProgressBar is in use, interaction is non-modal, typically meaning that the user is not blocked by that operation's completion, and can continue to use the app in other ways until that aspect has completed.

- **Keywords**

  If your operation falls around these keywords, or if you're showing text that alongside the progress operation that matches these keywords; consider using a ProgressBar:
  - *Loading...*

- *Retrieving*
- *Working...*

ProgressRing

- **Will the operation cause the user to wait to continue?**

  If an operation requires all (or a large portion of) interaction with the app to wait until it has been completed, then the indeterminate ProgressRing is the better choice.
  - **Does the control have a defined duration or predictable end?**

  Use a determinate ProgressRing if you want the visual to be a ring instead of a bar, and update the percentage or value accordingly.

- **Is the app waiting for the user to complete a task?**

  If so, use an indeterminate ProgressRing, as they're meant to indicate an unknown wait time for the user.

- **Keywords**

  If your operation falls around these keywords, or if you're showing text alongside the progress operation that matches these keywords; consider using a ProgressRing:
  - *Refreshing*
  - *Signing in...*
  - *Connecting...*

No progress indication necessary

- **Does the user need to know that something is happening?**

  For example, if the app is downloading something in the background and the user didn't initiate the download, the user doesn't necessarily need to know about it.

- **Is the operation a background activity that doesn't block user activity and is of minimal (but still some) interest to the user?**

  Use text when your app is performing tasks that don't have to be visible all the time, but you still need to show the status.

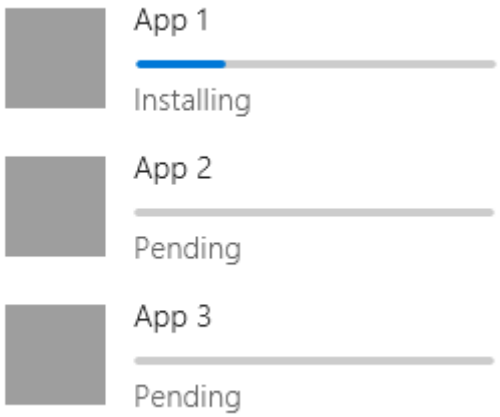
- **Does the user only care about the completion of the operation?**

> Sometimes it's best to show a notice only when the operation is completed, or give a visual that the operation has been completed immediately, and run the finishing touches in the background.

Progress controls best practices

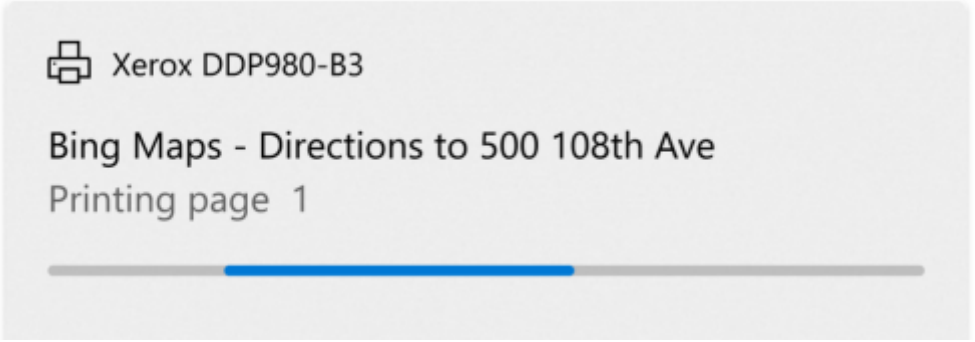
Sometimes it's best to see some visual representations of when and where to use these different progress controls:

ProgressBar - Determinate



The first example is the determinate ProgressBar. When the duration of the operation is known, when installing, downloading, setting up, etc; a determinate ProgressBar is best.

ProgressBar - Indeterminate

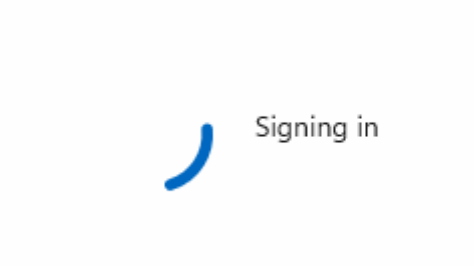


When it is not known how long the operation will take, use an indeterminate ProgressBar. Indeterminate ProgressBars are also good when filling a virtualized list, and creating a smooth visual transition between an indeterminate to determinate ProgressBar.

- **Is the operation in a virtualized collection?**

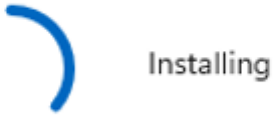
If so, do not put a progress indicator on each list item as they appear. Instead, use a ProgressBar and place it at the top of the collection of items being loaded in, to show that the items are being fetched.

ProgressRing - Indeterminate



The indeterminate ProgressRing is used when any further user interaction with the app is halted, or the app is waiting for the user's input to continue. The "signing in..." example above is a perfect scenario for the ProgressRing, the user cannot continue using the app until the sign is has completed.

ProgressRing - Determinate



When the duration of the operation is known and the ring visual is desired, when installing, downloading, setting up, etc; a determinate ProgressRing is best.

Create a progress control

✔ **Important APIs:** ProgressBar class, IsIndeterminate property, ProgressRing class, IsActive property

Open the WinUI 3 Gallery app and see the **ProgressBar** or **ProgressRing**.

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Both progress controls are rather simple; but some visual features of the controls are not obvious to customize.

Sizing the ProgressRing

The ProgressRing can be sized as large as you want, but can only be as small as 20x20epx. In order to resize a ProgressRing, you must set its height and width. If only height or width are set, the control will assume minimum sizing (20x20epx) – conversely if the height and width are set to two different sizes, the smaller of the sizes will be assumed. To ensure your ProgressRing is correct for your needs, set both the height and the width to the same value:

XAML

```
<ProgressRing Height="100" Width="100"/>
```

To make your ProgressRing visible, and animate, you must set the IsActive property to true:

XAML

```
<ProgressRing IsActive="True" Height="100" Width="100"/>
```

C#

```
progressRing.IsActive = true;
```

Coloring the progress controls

By default, the main coloring of the progress controls is set to the accent color of the system. To override this brush simply change the foreground property on either control.

XAML

```
<ProgressRing IsActive="True" Height="100" Width="100" Foreground="Blue"/>
<ProgressBar Width="100" Foreground="Green"/>
```

Changing the foreground color for the ProgressRing will change the fill color of the ring. The foreground property for the ProgressBar will change the fill color of the bar – to alter the unfilled portion of the bar, simply override the background property.

Showing a wait cursor

Sometimes it's best to just show a brief wait cursor, when the app or operation needs time to think, and you need to indicate to the user that the app or area where the wait cursor is visible should not be interacted with until the wait cursor has disappeared.

```
C#

Window.Current.CoreWindow.PointerCursor = new
Windows.UI.Core.CoreCursor(Windows.UI.Core.CoreCursorType.Wait, 10);
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The progress controls for UWP apps are included as part of WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in both the Windows.UI.Xaml.Controls and Microsoft.UI.Xaml.Controls namespaces.

- ✓ **UWP APIs:** ProgressBar class, IsIndeterminate property, ProgressRing class, IsActive property
- ✓ **WinUI 2 Apis:** ProgressBar class, IsIndeterminate property, ProgressRing class, IsActive property
- ✓ Open the WinUI 2 Gallery app and see the ProgressBar or ProgressRing. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles, templates, and features for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:ProgressBar />
<muxc:ProgressRing />
```

Related articles

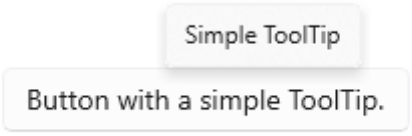
- ProgressBar class
- ProgressRing class

Tooltips

Article • 02/26/2025

A tooltip is a popup that contains additional information about another control or object. Tooltips display automatically when the user moves focus to, presses and holds, or hovers the pointer over the associated control. The tooltip disappears when the user moves focus from, stops pressing on, or stops hovering the pointer over the associated control (unless the pointer is moving towards the tooltip).

> **Note**
>
> Starting with Windows 11 version 21H2, a tooltip can also be dismissed by pressing the CTRL key.



Is this the right control?

Use a tooltip to reveal more info about a control before asking the user to perform an action. Tooltips should be used sparingly, and only when they are adding distinct value for the user who is trying to complete a task. One rule of thumb is that if the information is available elsewhere in the same experience, you do not need a tooltip. A valuable tooltip will clarify an unclear action.

When should you use a tooltip? To decide, consider these questions:

- **Should info become visible based on pointer hover?** If not, use another control. Display tips only as the result of user interaction, never display them on their own.
- **Does a control have a text label?** If not, use a tooltip to provide the label. It is a good UX design practice to label most controls inline and for these you don't need tooltips. Toolbar controls and command buttons showing only icons need tooltips.
- **Does an object benefit from a description or further info?** If so, use a tooltip. But the text must be supplemental — that is, not essential to the primary tasks. If it is essential, put it directly in the UI so that users don't have to discover or hunt for it.

- **Is the supplemental info an error, warning, or status?** If so, use another UI element, such as a flyout.
- **Do users need to interact with the tip?** If so, use another control. Users can't interact with tips because moving the mouse makes them disappear.
- **Do users need to print the supplemental info?** If so, use another control.
- **Will users find the tips annoying or distracting?** If so, consider using another solution — including doing nothing at all. If you do use tips where they might be distracting, allow users to turn them off.

Recommendations

- Use tooltips sparingly (or not at all). Tooltips are an interruption. A tooltip can be as distracting as a pop-up, so don't use them unless they add significant value.
- Keep the tooltip text concise. Tooltips are perfect for short sentences and sentence fragments. Large blocks of text can be overwhelming and the tooltip may time out before the user has finished reading.
- Create helpful, supplemental tooltip text. Tooltip text must be informative. Don't make it obvious or just repeat what is already on the screen. Because tooltip text isn't always visible, it should be supplemental info that users don't have to read. Communicate important info using self-explanatory control labels or in-place supplemental text.
- Use images when appropriate. Sometimes it's better to use an image in a tooltip. For example, when the user hovers over a hyperlink, you can use a tooltip to show a preview of the linked page.
- Keyboard accelerators are displayed in tooltips by default. If you add your own tooltip, make sure that it includes information about the keyboard accelerators which are available.
- Don't use a tooltip to display text already visible in the UI. For example, don't put a tooltip on a button that shows the same text of the button.
- Don't put interactive controls inside the tooltip.
- Don't put images that look like they are interactive inside the tooltip.

Create a tooltip

✓ **Important APIs:** ToolTip class, ToolTipService class

Open the WinUI 3 Gallery app and see the ToolTip in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

A ToolTip must be assigned to another UI element that is its owner. The ToolTipService class provides static methods to display a ToolTip.

In XAML, use the **ToolTipService.Tooltip** attached property to assign the ToolTip to an owner.

XAML

```
<Button Content="New" ToolTipService.ToolTip="Create a new document"/>
```

In code, use the ToolTipService.SetToolTip method to assign the ToolTip to an owner.

XAML

```
<Button x:Name="submitButton" Content="New"/>
```

C#

```
ToolTip toolTip = new ToolTip();
toolTip.Content = "Create a new document";
ToolTipService.SetToolTip(submitButton, toolTip);
```

Content

You can use any object as the Content of a ToolTip. Here's an example of using an Image in a ToolTip.

XAML

```
<TextBlock Text="store logo">
    <ToolTipService.ToolTip>
        <Image Source="Assets/StoreLogo.png"/>
    </ToolTipService.ToolTip>
</TextBlock>
```

Placement

By default, a ToolTip is displayed centered above the pointer. The placement is not constrained by the app window, so the ToolTip might be displayed partially or completely outside of the app window bounds.

For broad adjustments, use the Placement property or **ToolTipService.Placement** attached property to specify whether the ToolTip should draw above, below, left, or right of the pointer. You can set the VerticalOffset or HorizontalOffset properties to change the distance between the pointer and the ToolTip. Only one of the two offset values will influence the final position - VerticalOffset when Placement is Top or Bottom, HorizontalOffset when Placement is Left or Right.

XAML

```
<!-- An Image with an offset ToolTip. -->
<Image Source="Assets/StoreLogo.png">
    <ToolTipService.ToolTip>
        <ToolTip Content="Offset ToolTip."
                 Placement="Right"
                 HorizontalOffset="20"/>
    </ToolTipService.ToolTip>
</Image>
```

If a ToolTip obscures the content it is referring to, you can adjust its placement precisely using the **PlacementRect** property. PlacementRect anchors the ToolTip's position and also serves as an area that ToolTip will not occlude, provided there's sufficient screen space to draw ToolTip outside this area. You can specify the origin of the rectangle relative to the ToolTip's owner, and the height and width of the exclusion area. The Placement property will define if ToolTip should draw above, below, left, or right of the PlacementRect.

XAML

```
<!-- An Image with a non-occluding ToolTip. -->
<Image Source="Assets/StoreLogo.png" Height="64" Width="96">
    <ToolTipService.ToolTip>
        <ToolTip Content="Non-occluding ToolTip."
                 PlacementRect="0,0,96,64"/>
    </ToolTipService.ToolTip>
</Image>
```

UWP and WinUI 2

ⓘ Important

The information and examples in this article are optimized for apps that use the **Windows App SDK** and **WinUI 3**, but are generally applicable to UWP apps that use **WinUI 2**. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** ToolTip class, ToolTipService class
- ✓ Open the WinUI 2 Gallery app and see the ToolTip in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

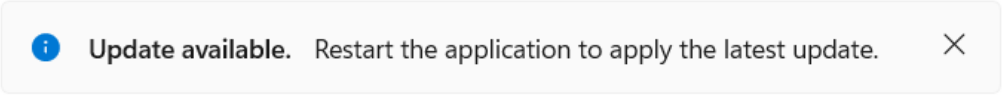
Related articles

- ToolTip class

InfoBar

Article • 02/26/2025

The InfoBar control is for displaying app-wide status messages to users that are highly visible yet non-intrusive. There are built-in Severity levels to easily indicate the type of message shown as well as the option to include your own call to action or hyperlink button. Since the InfoBar is inline with other UI content the option is there for the control to always be visible or dismissed by the user.



Is this the right control?

Use an InfoBar control when a user should be informed of, acknowledge, or take action on a changed application state. By default the notification will remain in the content area until closed by the user but will not necessarily break user flow.

An InfoBar will take up space in your layout and behave like any other child elements. It will not cover up other content or float on top of it.

Do not use an InfoBar control to confirm or respond directly to a user action that doesn't change the state of the app, for time-sensitive alerts, or for non-essential messages.

Remarks

Use an InfoBar that is closed by the user or when the status is resolved for scenarios that **directly** impact app perception or experience.

Here are some examples:

- Internet connectivity lost
- Error while saving a document when triggered automatically, not related to specific user action
- No microphone plugged in when attempting to record

- The subscription to the application is expired

Use an InfoBar that is closed by the user for scenarios that **indirectly** impact app perception or experience

Here are some examples:

- A call has begun recording
- Update applied with link to 'Release Notes'
- The terms of service have been updated and require acknowledgement
- An app-wide backup has successfully, asynchronously completed
- The subscription to the application is close to expiring

When should a different control be used?

There are some scenarios where a ContentDialog, Flyout, or TeachingTip may be more appropriate to use.

- For scenarios where a persistent notification is not needed, e.g. displaying information in context of a specific UI element, a Flyout is a better option.
- For scenarios where the application is confirming a user action, showing information the user ***must*** read, use a ContentDialog.
  - Additionally, if a status change to the app is so severe that it needs to block all further ability for the user to interact with the app, use a ContentDialog.
- For scenarios where a notification is a transient teaching moment, a TeachingTip is a better option.

For more info about choosing the right notification control, see the Dialogs and Flyouts article.

Recommendations

Enter and Exit Usability

Flashing content

The InfoBar should not appear and disappear from view rapidly to prevent flashing on the screen. Avoid flashing visuals for people with photosensitivities and to improve the usability of your application.

For InfoBars that automatically enter and exit the view via an app status condition, we recommend you include logic in your application to prevent content from appearing or

disappearing rapidly or multiple times in a row. However, in general, this control should be used for long-lived status messages.

Updating the InfoBar

Once the control is open, any changes made to the various properties like updating the message or Severity will not raise a notification event. If you would like to inform users that use screen readers of the InfoBar's updated content we recommend you close and re-open the control to trigger the event.

Inline messages offsetting content

For InfoBars that are inline with other UI content, keep in mind how the rest of the page will responsively react to the addition of the element.

InfoBars with a substantial height could dramatically alter the layout of the other elements on the page. If the InfoBar appears or disappears rapidly, especially in succession, the user may be confused with the changing visual state.

Color and Icon

When customizing the color and icon outside of the preset Severity levels, keep in mind user expectations for the connotations from the set of standard icons and colors.

Additionally, the preset Severity colors have already been designed for theme changes, high-contrast mode, color confusion accessibility, and contrast with foreground colors. We recommend to use these colors when possible and to include custom logic in your application to adapt to the various color states and accessibility features.

Please view the UX guidance for Standard Icons and Color to ensure your message is communicated clearly and accessible to users.

Severity

Avoid setting the Severity property for a notification that does not match the information communicated in the Title, Message, or custom content.

The accompanying information should aim to communicate the following to use that Severity.

- Error: An error or problem that has occurred.
- Warning: A condition that might cause a problem in the future.

- Success: A long-running and/or background task has completed.
- Default: General information that requires the user's attention.

Icons and color should not be the only UI components signifying meaning for your notification. Text in the notification's Title and/or Message should be included to display information.

Message

Text in your notification will not be a constant length in all languages. For the Title and Message property this may impact whether your notification will expand to a second line. We recommend you avoid positioning based on message length or other UI elements set to a specific language.

The notification will follow standard mirroring behavior when localized to/from languages that are right to left (RTL) or left to right (LTR). The icon will only mirror if there is directionality.

Please view the guidance for Adjust layout and fonts, and support RTL for more information about text localization in your notification.

Create an InfoBar

✓ **Important APIs:** InfoBar class

Open the WinUI 3 Gallery app and see the InfoBar in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

The XAML below describes an inline InfoBar with the default styling for an informational notification. An info bar can be placed like any other element and will follow base layout behavior. For example, in a vertical StackPanel, the InfoBar will horizontally expand to fill the available width.

By default, the InfoBar will not be visible. Set the IsOpen property to true in the XAML or code behind to display the InfoBar.

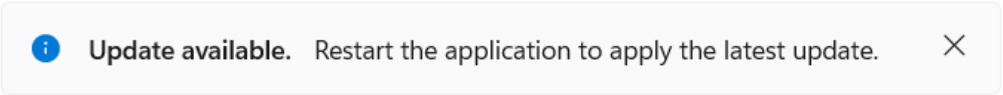
XAML

```
<InfoBar x:Name="UpdateAvailableNotification"
    Title="Update available."
```

```
    Message="Restart the application to apply the latest update.">
</InfoBar>
```

C#

```
public MainPage()
{
    this.InitializeComponent();

    if(IsUpdateAvailable())
    {
        UpdateAvailableNotification.IsOpen = true;
    }
}
```

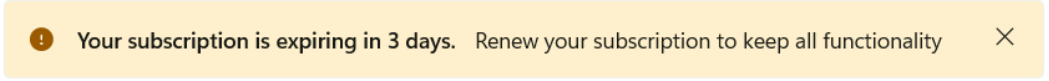


Using pre-defined severity levels

The type of the info bar can be set via the Severity property to automatically set a consistent status color, icon, and assistive technology settings dependent on the criticality of the notification. If no Severity is set, the default informational styling is applied.

XAML

```
<InfoBar x:Name="SubscriptionExpiringNotification"
    Severity="Warning"
    Title="Your subscription is expiring in 3 days."
    Message="Renew your subscription to keep all functionality" />
```



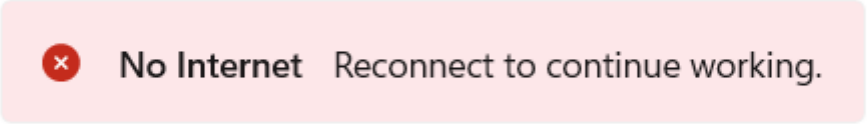
Programmatic close in InfoBar

An InfoBar can be closed by the user via the close button or programmatically. If the notification is required to be in view until the status is resolved and you would like to remove the ability for the user to close the info bar, you can set the IsClosable property to false.

The default value of this property is true, in which case the close button is present and takes the form of an 'X'.

XAML

```
<InfoBar x:Name="NoInternetNotification"
    Severity="Error"
    Title="No Internet"
    Message="Reconnect to continue working."
    IsClosable="False" />
```



Customization: Background color and icon

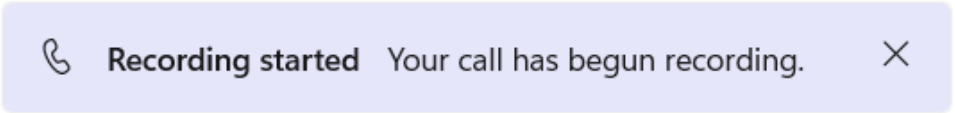
Outside of the pre-defined severity levels, the Background and IconSource properties can be set to customize the icon and background color. The InfoBar will retain the assistive technology settings of the severity defined, or default if none was defined.

A custom background color can be set via the standard Background property and will override the color set by Severity. Please keep in mind content readability and accessibility when setting your own color.

A custom icon can be set via the IconSource property. By default, an icon will be visible (assuming the control isn't collapsed). This icon can be removed by setting the IsIconVisible property to false. For custom icons, the recommended icon size is 20px.

XAML

```
<InfoBar x:Name="CallRecordingNotification"
    Title="Recording started"
    Message="Your call has begun recording."
    Background="{StaticResource LavenderBackgroundBrush}">
    <InfoBar.IconSource>
        <SymbolIconSource Symbol="Phone" />
    </InfoBar.IconSource>
</InfoBar>
```

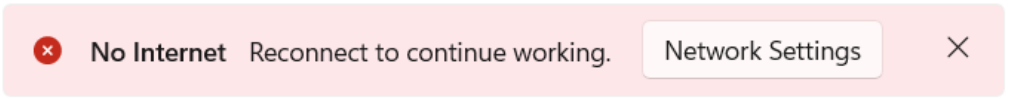


Add an action button

An additional action button can be added by defining your own button that inherits ButtonBase and setting it in the ActionButton property. Custom styling will be applied to action buttons of type Button and HyperlinkButton for consistency and accessibility. Aside from the ActionButton property, additional action buttons can be added via custom content and will appear below the message.

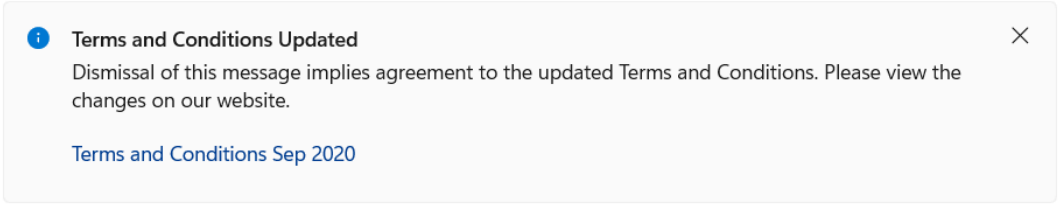
XAML

```
<InfoBar x:Name="NoInternetNotification"
    Severity="Error"
    Title="No Internet"
    Message="Reconnect to continue working.">
    <InfoBar.ActionButton>
        <Button Content="Network Settings"
Click="InternetInfoBarButton_Click" />
    </InfoBar.ActionButton>
</InfoBar>
```



XAML

```
<InfoBar x:Name="TermsAndConditionsUpdatedNotification"
    Title="Terms and Conditions Updated"
    Message="Dismissal of this message implies agreement to the updated
Terms and Conditions. Please view the changes on our website.">
    <InfoBar.ActionButton>
        <HyperlinkButton Content="Terms and Conditions Sep 2020"
            NavigateUri="https://www.example.com"
            Click="UpdateInfoBarHyperlinkButton_Click" />
    </InfoBar.ActionButton>
</InfoBar>
```



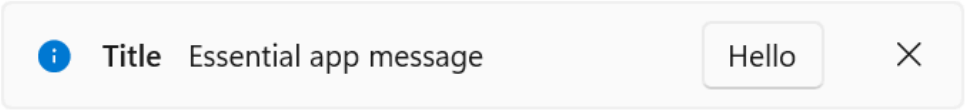
Right-aligned action button

By default, the action button is left-aligned in the available space directly to the right of the Message. Optionally, you can set HorizontalAlignment to "right" in your button to align this button to be directly to the left of the close button in the single line, horizontal, layout.

XAML

```
<InfoBar x:Name="RightAlignedButtonNotification"
    Title="Title"
    Message="Essential app message">
    <InfoBar.ActionButton>
        <Button Content="Hello" HorizontalAlignment="Right"/>
```

```
    </InfoBar.ActionButton>
</InfoBar>
```

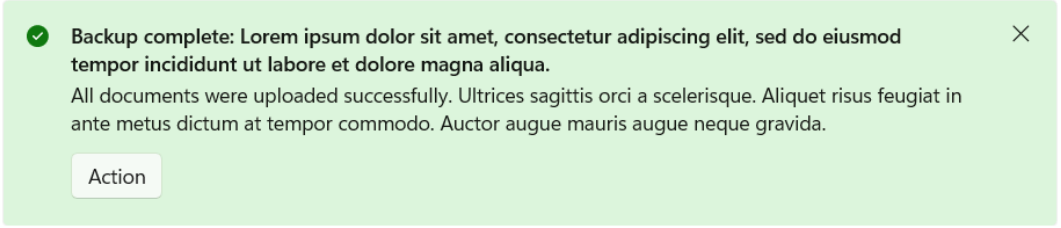


Content wrapping

If the InfoBar content, excluding custom content, is unable to fit on a single horizontal line they will be laid out vertically. The Title, Message, and ActionButton — if present — will each appear on separate lines.

XAML

```
<InfoBar x:Name="BackupCompleteNotification"
    Severity="Success"
    Title="Backup complete: Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua."
    Message="All documents were uploaded successfully. Ultrices sagittis
orci a scelerisque. Aliquet risus feugiat in ante metus dictum at tempor
commodo. Auctor augue mauris augue neque gravida.">
    <InfoBar.ActionButton>
        <Button Content="Action"
            Click="BackupInfoBarButton_Click" />
    </InfoBar.ActionButton>
</InfoBar>
```

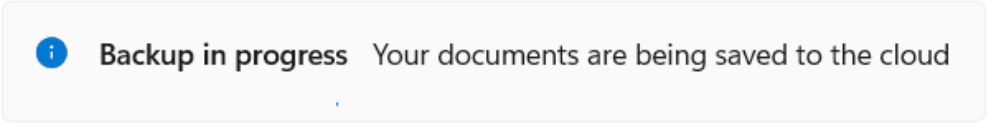


Custom content

XAML content can be added to an InfoBar using the Content property. It will appear in its own line below the rest of the control content. The InfoBar will expand to fit the content defined.

XAML

```
<InfoBar x:Name="BackupInProgressNotification"
    Title="Backup in progress"
    Message="Your documents are being saved to the cloud"
    IsClosable="False">
    <InfoBar.Content>
        <ProgressBar IsIndeterminate="True" Margin="0,0,0,6"
MaxWidth="200"/>
    </InfoBar.Content>
</InfoBar>
```



Lightweight styling

You can modify the default Style and ControlTemplate to give the control a unique appearance. See the Control Style and Template section of the InfoBar API docs for a list of the available theme resources.

For more info, see the Light-weight styling section of the Styling controls article.

For example, the following causes the background color for all informational InfoBars on a page to be blue:

XAML

```
<Page.Resources>
    <x:SolidColorBrush x:Key="InfoBarInformationalSeverityBackgroundBrush"
Color="LightBlue"></x:SolidColorBrush>
</Page.Resources>
```

Canceling close

The Closing event can be used to cancel and/or defer the close of an InfoBar. This can be used to keep the InfoBar open or allow time for a custom action to occur. When the closing of an InfoBar is canceled, IsOpen will go back to true. A programmatic close can also be canceled.

XAML

```
<InfoBar x:Name="UpdateAvailable"
    Title="Update Available"
    Message="Please close this tip to apply required security updates to
this application"
    Closing="InfoBar_Closing">
</InfoBar>
```

C#

```
public void InfoBar_Closing(InfoBar sender, InfoBarClosingEventArgs args)
{
    if (args.Reason == InfoBarCloseReason.CloseButton)
    {
        if (!ApplyUpdates())
        {
            // could not apply updates - do not close
            args.Cancel = true;
        }
    }
}
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

The InfoBar for UWP apps requires WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in the Microsoft.UI.Xaml.Controls

namespace.

- ✓ **WinUI 2 Apis:** InfoBar class
- ✓ Open the WinUI 2 Gallery app and see InfoBar in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:InfoBar />
```

Related articles

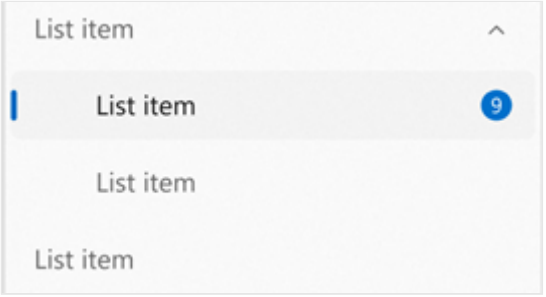
- Dialogs and flyouts

Info badge

Article • 02/26/2025

Badging is a non-intrusive and intuitive way to display notifications or bring focus to an area within an app - whether that be for notifications, indicating new content, or showing an alert. An info badge is a small piece of UI that can be added into an app and customized to display a number, icon, or a simple dot.

The info badge is built into the XAML navigation view, but can also be placed as a standalone element in the XAML tree, allowing you to place an info badge into any control or piece of UI of your choosing. When you use an info badge somewhere other than navigation view, you are responsible for programmatically determining when to show and dismiss the info badge, and where to place the info badge.



Is this the right control?

An info badge should be used when you want to bring the user's focus to a certain area of your app in an unintrusive way. When an info badge appears, it is meant to bring focus to an area and then let the user get back into their flow, giving them the choice of whether or not to look into the details of why the info badge appeared. Info badges should only represent messages that are dismissible and non-permanent – an info badge should have specific rules as to when it can appear, disappear, and change.

Examples of appropriate info badge usage:

- To indicate new messages have arrived.
- To indicate new articles are available to read.
- To indicate that there are new options available on a page.
- To indicate that there might be an issue with an item on a certain page that does not block the app from functioning.

When should a different control be used?

An info badge should not be used to display critical errors or convey highly important messages that need immediate action. Info badges should not be used in cases where they need to be interacted with immediately to continue using the app.

Examples of inappropriate info badge usage:

- To indicate an urgent matter on a page within the app that needs to be addressed before continuing to use the app. For this scenario, use a content dialog.
- Appearing in an app with no way for the user to dismiss the info badge. For a persistent alert like this, use an info bar.
- Using the info badge as a permanent way of bringing the user's focus to an area, without a way for the user to dismiss the info badge.
- Using an info badge as a regular icon or image in your app. Instead, use an appropriate image or icon (see IconElement and IconSource).

Types of info badges

There are three styles of info badge that you can choose from - *dot*, *icon*, and *numeric*, as shown in order below.

Dot info badge

The dot info badge is a simple ellipse with a diameter of 4px. It has no border, and is not meant to hold text or anything else inside of it.

You should use the dot info badge for general scenarios in which you want to guide the user's focus towards the info badge – for example, to indicate new content or updates are available.

Icon info badge

The icon info badge is an ellipse with a diameter of 16px that holds an icon inside of it. The info badge has an IconSource property that provides flexibility for the types of supported icons.

You should use the icon info badge to send a quick message along with getting the user's attention – for example, to alert the user that something non-blocking has gone wrong, an extra important update is available, or that something specific in the app is currently enabled (such as a countdown timer going).

If you'd like to use a BitmapIconSource for the `IconSource` of your info badge, you are responsible for ensuring that the bitmap fits inside of the info badge (either by changing the size of the icon, or changing the size of the info badge).

Numeric info badge

The numeric info badge is the same shape and size as the icon info badge, but it holds a number inside of it, determined by the Value property. Numbers must be whole integers and must be greater than or equal to zero. The width of the info badge will automatically expand as the number being displayed grows to multiple digits, with a smooth animation.

You should use the numeric info badge to show that there are a specific number of items that need attention – for example, new emails or messages.

Preset info badge styles

To help support the most common scenarios in which info badges are used, the control includes built-in preset info badge styles. While you can customize your info badge to use any color/icon/number combination that you want, these built-in presets are a quick option to make sure that your info badge is compliant with accessibility guidelines and is proportional in terms of icon and number sizing.

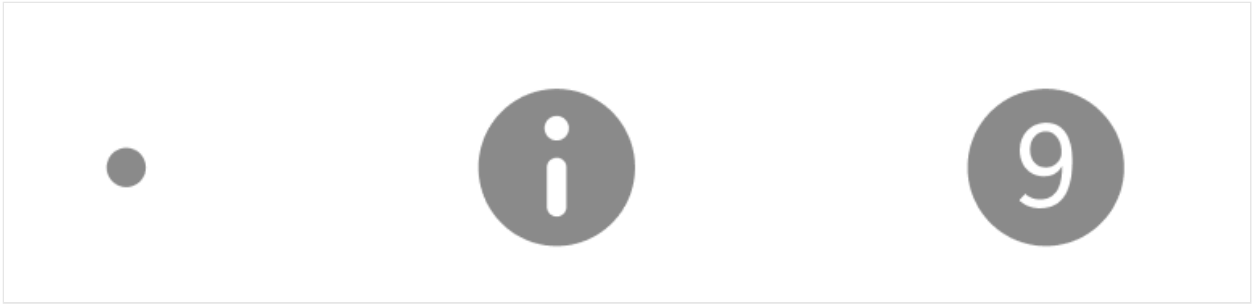
The following style presets are available for info badges:

Attention

- `AttentionDotInfoBadgeStyle`
- `AttentionIconInfoBadgeStyle`
- `AttentionValueInfoBadgeStyle`

Informational

- `InformationalDotInfoBadgeStyle`
- `InformationalIconInfoBadgeStyle`
- `InformationalValueInfoBadgeStyle`



Success

- `SuccessDotInfoBadgeStyle`
- `SuccessIconInfoBadgeStyle`
- `SuccessValueInfoBadgeStyle`

Caution

- `CautionDotInfoBadgeStyle`
- `CautionIconInfoBadgeStyle`
- `CautionValueInfoBadgeStyle`

Critical

- `CriticalDotInfoBadgeStyle`

- `CriticalIconInfoBadgeStyle`
- `CriticalValueInfoBadgeStyle`

If a style is set on an info badge and a conflicting property is also set, the property will overwrite the conflicting part of the style, but non-conflicting style elements will stay applied.

For example, if you apply the `CriticalIconInfoBadgeStyle` to an info badge, but also set `InfoBadge.Value = "1"`, you would end up with an info badge that has the "Critical" background color but displays the number 1 inside of it, rather than displaying the preset icon.

This example creates an info badge that takes on the color and icon of the *Attention Icon* preset style.

XAML

```
<InfoBadge Style="{ThemeResource AttentionIconInfoBadgeStyle}"/>
```

Accessibility

The info badge control does not have any screen reader functionality or user interface automation (UIA) built-in to it on its own, as the control is not focusable and cannot be interacted with.

If you're using an info badge inside of a navigation view, the navigation view provides built-in screen reader and assistive technology support. When you're tabbing through a navigation view and you land on a navigation view item with an info badge on it, the screen reader will announce that there is an info badge on this item. If the info badge in question is numeric, the screen reader will announce the info badge's value as well.

If you are using an info badge outside of a navigation view, we recommend the following to ensure your app is fully accessible:

- The parent element of the info badge should be focusable and accessible by tab.
- The parent element announces the info badge to screen readers.
- The app sends a UIA notification when the info badge appears for the first time.
- The app sends a UIA notification when an info badge disappears from the UI.
- The app sends a UIA notification when a significant change has occurred with an existing info badge.
    - The definition of "significant change" is up to you as the individual developer. Examples of this can include: an info badge switching between different types, an info badge changing color to represent its status, or an info badge's value exceeding a certain significant number.

To control what the parent element announces to screen readers, you can use attached properties of the AutomationProperties class. For an info badge, it's recommended that you set either the `AutomationProperties.FullDescription` or `AutomationProperties.ItemStatus` attached properties on the parent element.

To send UIA notifications upon the info badge's appearance or dismissal, you can use the AutomationPeer.RaiseAutomationEvent method.

The info badge comes at a default size that meets accessibility requirements. You can customize many aspects of the info badge including its height/width/color, etc., but it's important that the default info badge adheres to our accessibility guidelines for size and color.

Create an InfoBadge

> **ⓘ Important**
>
> Some information relates to prerelease product that may be substantially modified before it's released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

✓ **Important APIs:** InfoBadge class, IconSource property, Value property

Open the WinUI 3 Gallery app and see the InfoBadge in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source

> code on GitHub

You can create an InfoBadge in XAML or in code. The kind of InfoBadge you create is determined by which properties you set.

Dot

To create a dot InfoBadge, use a default InfoBadge control with no properties set.

XAML

```
<InfoBadge />
```

Icon

To create an icon InfoBadge, set the IconSource property.

XAML

```
<InfoBadge x:Name="SyncStatusInfoBadge">
    <InfoBadge.IconSource>
        <SymbolIconSource Symbol="Sync"/>
    </InfoBadge.IconSource>
</InfoBadge>
```

Numeric

To create a numeric InfoBadge, set the Value property.

XAML

```
<InfoBadge x:Name="EmailInfoBadge" Value="{x:Bind numUnreadMail}"/>
```

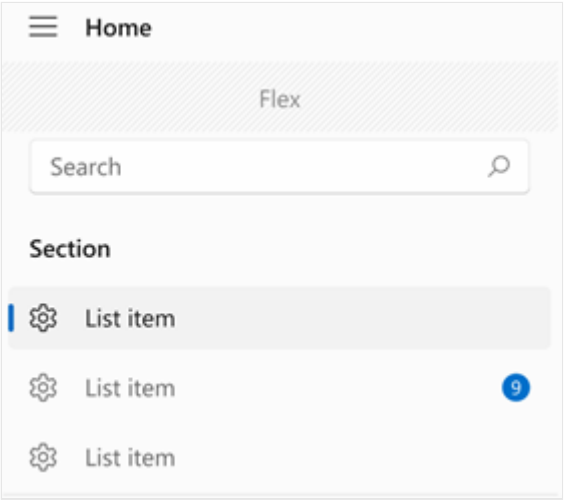
In most scenarios, you'll bind the `Value` property of the InfoBadge to a changing integer value in your app's backend so you can easily increment/decrement and show/hide the InfoBadge based on that specific value.

> ⓘ **Note**
>
> If both the `Icon` and `Value` properties are set, the `Value` property takes precedence and the InfoBadge appears as a numeric InfoBadge.

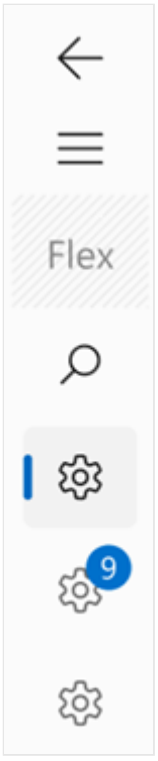
Using an InfoBadge in NavigationView

If you're using a NavigationView in your app, we recommend that you use an InfoBadge in the NavigationView to show app-wide notifications and alerts. To place the InfoBadge on a NavigationViewItem, assign the InfoBadge object to the `NavigationViewItem.InfoBadge` property.

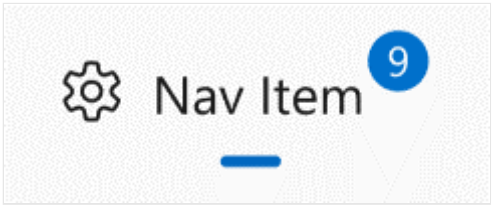
In Left-Expanded mode, the InfoBadge appears right-aligned to the edge of the NavigationViewItem.



In Left-Compact mode, the InfoBadge appears overlayed on the top right corner of the icon.



In Top mode, the InfoBadge is aligned to the upper right hand corner of the overall item.



We recommend that you not use different types of InfoBadges in one NavigationView, such as attaching a numeric InfoBadge to one NavigationViewItem and a dot InfoBadge to another NavigationViewItem in the same NavigationView.

Example: Incrementing a numeric InfoBadge in a NavigationView

This example simulates how an email app could use an InfoBadge in a NavigationView to display the number of new emails in the inbox, and increment the number shown in the InfoBadge when a new message is received.

XAML

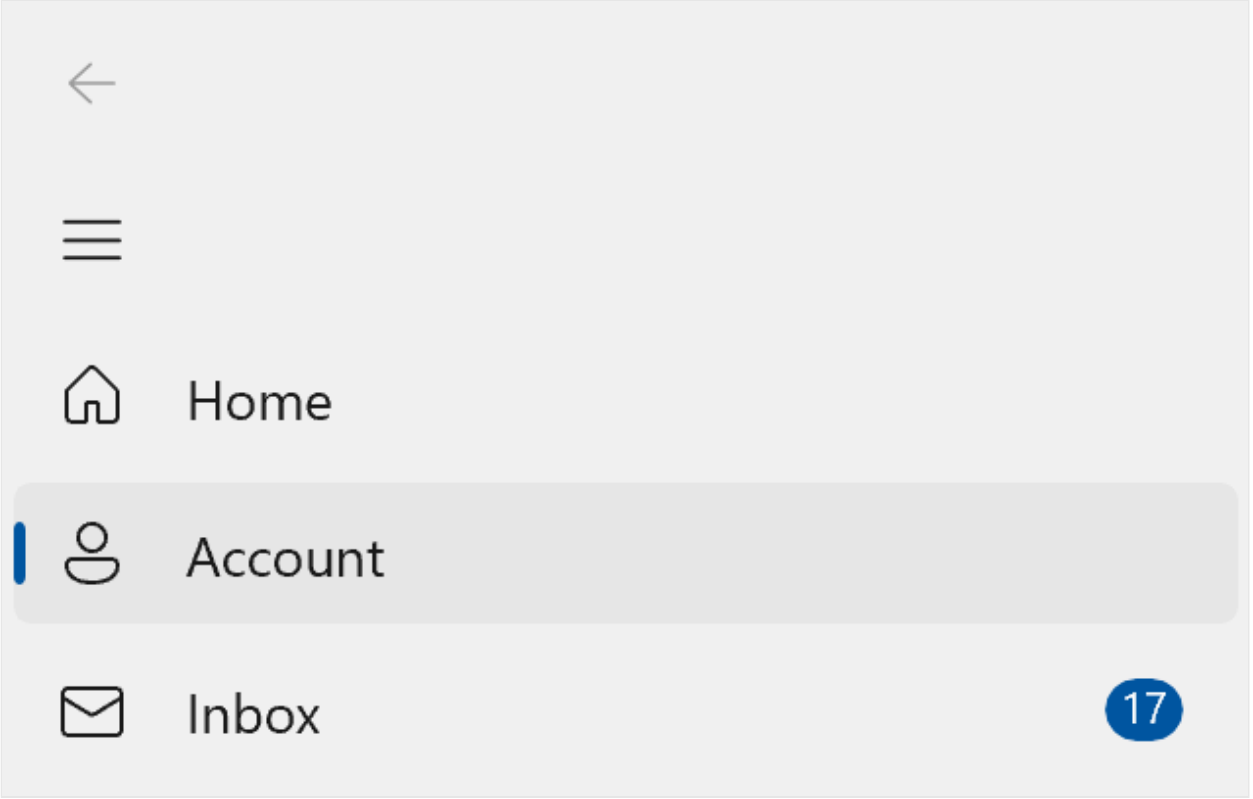
```
<NavigationView SelectionChanged="NavigationView_SelectionChanged">
    <NavigationView.MenuItems>
        <NavigationViewItem Content="Home" Icon="Home"/>
        <NavigationViewItem Content="Account" Icon="Contact"/>
        <NavigationViewItem x:Name="InboxPage" Content="Inbox" Icon="Mail">
            <NavigationViewItem.InfoBadge>
```

```
                <InfoBadge x:Name="bg1"
                                Value="{x:Bind mailBox.NewMailCount,
Mode=OneWay}"
                                Visibility="{x:Bind mailBox.HasNewMail,
Mode=OneWay}"/>
            </NavigationViewItem.InfoBadge>
        </NavigationViewItem>
    </NavigationView.MenuItems>
    <Frame x:Name="contentFrame" />
</NavigationView>
```

C#

```
public sealed partial class MainWindow : Window
{
    MailBox mailBox = new MailBox();

    public MainWindow()
    {
        this.InitializeComponent();

    }

    private void NavigationView_SelectionChanged(NavigationView sender,
                                NavigationViewSelectionChangedEventArgs args)
    {
        if (args.SelectedItem == InboxPage)
        {
            mailBox.ResetNewMailCount();
        }
        else
        {
            mailBox.CheckMail();
        }
    }
}

public class MailBox : DependencyObject
{
    DispatcherQueueTimer timer;

    // Dependency Properties for binding.
    public int NewMailCount
    {
        get { return (int)GetValue(NewMailCountProperty); }
        set { SetValue(NewMailCountProperty, value); }
    }
    public static readonly DependencyProperty NewMailCountProperty =
        DependencyProperty.Register("NewMailCount", typeof(int),
typeof(MailBox), new PropertyMetadata(0));

    public bool HasNewMail
    {
```

```
        get { return (bool)GetValue(HasNewMailProperty); }
        set { SetValue(HasNewMailProperty, value); }
    }
    public static readonly DependencyProperty HasNewMailProperty =
        DependencyProperty.Register("HasNewMail", typeof(bool),
typeof(MailBox), new PropertyMetadata(false));

    public MailBox()
    {
        timer = this.DispatcherQueue.CreateTimer();
        timer.Interval = new TimeSpan(15000000);
        timer.Tick += (s, e) =>
        {
            NewMailCount++;
            if (HasNewMail == false) { HasNewMail = true; }
        };
        timer.Start();
    }

    public void ResetNewMailCount()
    {
        NewMailCount = 0;
        HasNewMail = false;
        timer.Stop();
    }

    public void CheckMail()
    {
        timer.Start();
    }
}
```

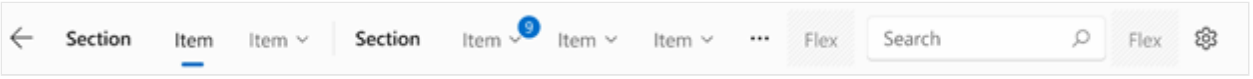


Hierarchy in NavigationView

If you have a hierarchical NavigationView, with NavigationViewItems nested in other NavigationViewItems, parent items will follow the same design/placement patterns as described above.

The parent NavigationViewItem and child NavigationViewItems will each have their own InfoBadge property. You can bind the value of the parent's InfoBadge to factors that determine the children's InfoBadge values, such as showing the sum of the children's numeric InfoBadges on the parent's InfoBadge.

This image shows a hierarchical NavigationView with its PaneDisplayMode set to Top, where the top-level (parent) item shows a numeric InfoBadge. The app has set the parent item InfoBadge to represent what's being displayed in the child items' InfoBadges, as the child items are currently not expanded (and therefore not visible).



Using an InfoBadge in another control

You might want to show alerts or notifications on elements within your app other than NavigationView. You might have a ListViewItem that needs special attention, or a menu item that displays a notification. In these cases, you can integrate InfoBadge directly into your UI with other controls.

InfoBadge is a UIElement and therefore cannot be used as a shared resource.

To do this, use InfoBadge as you would any other control – simply add the InfoBadge markup where you'd like it to appear. Since InfoBadge inherits from Control, it has all the built-in positioning properties, such as margin, alignment, padding, and more, which you can use to position your InfoBadge exactly where you want it.

If you place an InfoBadge inside of another control, such as a Button or a ListViewItem, it will likely get cropped if you position it to extend beyond the bounding box of the parent control. If your InfoBadge is inside of another control, it should not be positioned past the corners of the control's overall bounding box.

Example: Placing an InfoBadge inside another control

Here's a Button that has an InfoBadge placed in its upper right hand corner, with the badge layered on top of the content. This example can be applied to many controls

other than Button as well – it simply shows how to place, position, and show an InfoBadge inside of another WinUI control.

XAML

```
<Button Width="200" Height="60" Padding="4"
        HorizontalContentAlignment="Stretch"
VerticalContentAlignment="Stretch">
    <Grid>
        <SymbolIcon Symbol="Sync"/>
        <InfoBadge x:Name="buttonInfoBadge"
                        Background="#C42B1C"
                        HorizontalAlignment="Right"
                        VerticalAlignment="Top"
                        Width="16" Height="16">
            <InfoBadge.IconSource>
                <FontIconSource Glyph="&#xEA6A;"/>
            </InfoBadge.IconSource>
        </InfoBadge>
    </Grid>
</Button>
```

Managing an InfoBadge

An InfoBadge typically displays a transient alert, so it's common to show or hide it, and change it's style periodically while your app is running.

Showing and hiding an InfoBadge

You can use either the Visibility property or Opacity property to show and hide an InfoBadge based on user actions, program logic, counters, etc.

As with other UIElements, setting `Visibility.Collapsed` will make the InfoBadge not take space in your layout, so it might cause other elements to move around when it's shown and hidden.

If elements being repositioned is a concern, you can use the `Opacity` property to show and hide the InfoBadge. `Opacity` is set to `1.0` by default; you can set it to `0` to hide the InfoBadge. When you use the `Opacity` property, InfoBadge will still take up space in the layout even if it is currently hidden.

Change the InfoBadge style

You can change the icon or number displayed in an InfoBadge while it is being shown. Decrementing or incrementing a numeric InfoBadge based on user action can be achieved by changing the value of `InfoBadge.Value`. Changing the icon of an InfoBadge can be achieved by setting `InfoBadge.IconSource` to a new `IconSource` object. When changing icons, ensure that the new icon is the same size as the old icon to avoid a jarring visual effect.

Default behavior

If neither `InfoBadge.Value` nor `InfoBadge.IconSource` are set, the InfoBadge defaults to showing a dot (specifically if `Value` is set to `-1` and `IconSource` is set to `null`, which are the default values). If both the `Value` and `IconSource` properties are set, the InfoBadge will honor the `Value` property and display a number value.

You can also change the InfoBadge's type while it is being shown. To change the type of InfoBadge, be sure that the current type's corresponding property (`Value` or `IconSource`) is set to its default value (`-1` or `null`), and set the new type's property equal to an appropriate value. To change the type of InfoBadge from numeric or icon to a dot type InfoBadge, make sure that `InfoBadge.Value` is set to `-1` and `InfoBadge.IconSource` is set to `null`.

Depending on how you've positioned your InfoBadge, be aware that this may cause items to shift as the size and shape of the InfoBadge may change.

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

The InfoBadge for UWP apps requires WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in the Microsoft.UI.Xaml.Controls namespace.

- ✓ **WinUI 2 Apis:** InfoBadge class, IconSource property, Value property
- ✓ Open the WinUI 2 Gallery app and see InfoBadge in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:InfoBadge/>
```

Related articles

- NavigationView

Text controls

Article • 06/09/2022

Text controls consist of text input boxes, password boxes, auto-suggest boxes, and text blocks. The XAML framework provides several controls for rendering, entering, and editing text, and a set of properties for formatting the text.

- The controls for displaying read-only text are TextBlock and RichTextBlock.
- The controls for text entry and editing are: TextBox, RichEditBox, AutoSuggestBox, and PasswordBox.

> **Important APIs**: TextBlock class, RichTextBlock class, TextBox class, RichEditBox class, AutoSuggestBox class, PasswordBox class

Is this the right control?

The text control you use depends on your scenario. Use this info to pick the right text control to use in your app.

Render read-only text

Use a **TextBlock** to display most read-only text in your app. You can use it to display single-line or multi-line text, inline hyperlinks, and text with formatting like bold, italic, or underlined.

TextBlock is typically easier to use and provides better text rendering performance than RichTextBlock, so it's preferred for most app UI text. You can easily access and use text from a TextBlock in your app by getting the value of the Text property.

It also provides many of the same formatting options for customizing how your text is rendered. Although you can put line breaks in the text, TextBlock is designed to display a single paragraph and doesn't support text indentation.

Use a **RichTextBlock** when you need support for multiple paragraphs, multi-column text or other complex text layouts, or inline UI elements like images. RichTextBlock provides several features for advanced text layout.

The content property of RichTextBlock is the Blocks property, which supports paragraph based text via the Paragraph element. It doesn't have a **Text** property that you can use to easily access the control's text content in your app.

Text input

Use a **TextBox** control to let a user enter and edit unformatted text, such as in a form. You can use the Text property to get and set the text in a TextBox.

You can make a TextBox read-only, but this should be a temporary, conditional state. If the text is never editable, consider using a TextBlock instead.

Use a **PasswordBox** control to collect a password or other private data, such as a Social Security number. A password box is a text input box that conceals the characters typed in it for the sake of privacy. A password box looks like a text input box, except that it renders bullets in place of the text that has been entered. The bullet character can be customized.

Use an **AutoSuggestBox** control to show the user a list of suggestions to choose from as they type. An auto-suggest box is a text entry box that triggers a list of basic search suggestions. Suggested terms can draw from a combination of popular search terms and historical user-entered terms.

You should also use an AutoSuggestBox control to implement a search box.

Use a **RichEditBox** to display and edit text files. You don't use a RichEditBox to get user input into your app the way you use other standard text input boxes. Rather, you use it to work with text files that are separate from your app. You typically save text entered into a RichEditBox to a .rtf file.

Is text input the best option?

There are many ways you can get user input in your app. These questions will help answer whether one of the standard text input boxes or another control is the best fit for getting user input.

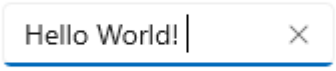
- **Is it practical to efficiently enumerate all valid values?** If so, consider using one of the selection controls, such as a check box, drop-down list, list box, radio button, slider, toggle switch, date picker, or time picker.
- **Is there a fairly small set of valid values?** If so, consider a drop-down list or a list box, especially if the values are more than a few characters long.
- **Is the valid data completely unconstrained? Or is the valid data only constrained by format (constrained length or character types)?** If so, use a text input control. You can limit the number of characters that can be entered, and you can validate the format in your app code.
- **Does the value represent a data type that has a specialized common control?** If so, use the appropriate control instead of a text input control. For example, use a DatePicker instead of a text input control to accept a date entry.

- If the data is strictly numeric:
  - **Is the value being entered approximate and/or relative to another quantity on the same page?** If so, use a slider.
  - **Would the user benefit from instant feedback on the effect of setting changes?** If so, use a slider, possibly with an accompanying control.
  - **Is the value entered likely to be adjusted after the result is observed, such as with volume or screen brightness?** If so, use a slider.

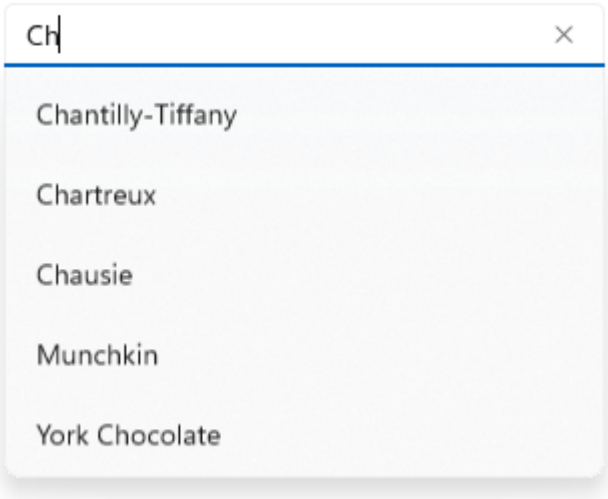
Examples

WinUI 2 Gallery
If you have the **WinUI 2 Gallery** app installed, click here to open the app and see the text controls in action. • Get the WinUI 2 Gallery app (Microsoft Store) • Get the source code (GitHub)

Text box



Auto suggest box



Password box

Create a text control

See these articles for info and examples specific to each text control.

- AutoSuggestBox
- PasswordBox
- RichEditBox
- RichTextBlock
- TextBlock
- TextBox

Font and style guidelines

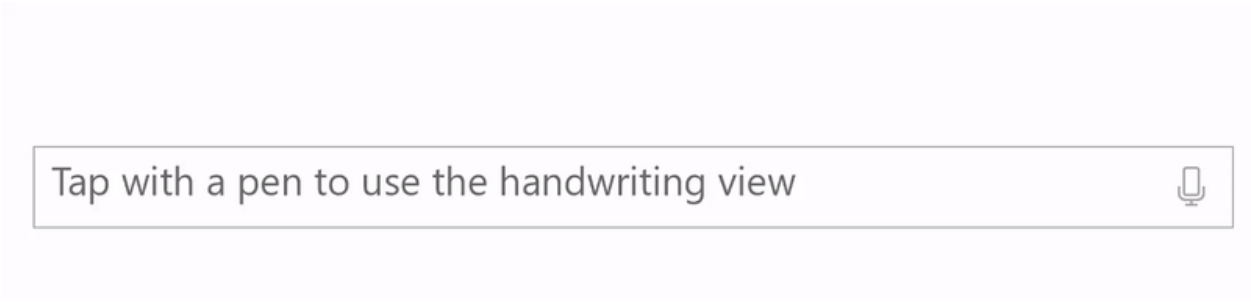
See these articles for font guidelines:

- Typography guidelines
- Segoe MDL2 icon list and guidelines

Pen input

Applies to: TextBox, RichEditBox, AutoSuggestBox

Starting with Windows 10, version 1803, XAML text input boxes feature embedded support for pen input using Windows Ink. When a user taps into a text input box using a Windows pen, the text box transforms to let the user write directly into it with a pen, rather than opening a separate input panel.



For more info, see Text input with the handwriting view.

Choose the right keyboard for your text control

Applies to: TextBox, PasswordBox RichEditBox

To help users to enter data using the touch keyboard, or Soft Input Panel (SIP), you can set the input scope of the text control to match the kind of data the user is expected to enter.

> Tip This info applies only to the SIP. It does not apply to hardware keyboards or the On-Screen Keyboard available in the Windows Ease of Access options.

The touch keyboard can be used for text entry when your app runs on a device with a touch screen. The touch keyboard is invoked when the user taps on an editable input field, such as a TextBox or RichEditBox. You can make it much faster and easier for users to enter data in your app by setting the input scope of the text control to match the kind of data you expect the user to enter. The input scope provides a hint to the system about the type of text input expected by the control so the system can provide a specialized touch keyboard layout for the input type.

For example, if a text box is used only to enter a 4-digit PIN, set the InputScope property to **Number**. This tells the system to show the number keypad layout, which makes it easier for the user to enter the PIN.

> Important
> The input scope does not cause any input validation to be performed, and does not prevent the user from providing any input through a hardware keyboard or other input device. You are still responsible for validating the input in your code as needed.

For more info, see Use input scope to change the touch keyboard.

Color fonts

Applies to: TextBlock, RichTextBlock, TextBox, RichEditBox

Windows has the ability for fonts to include multiple colored layers for each glyph. For example, the Segoe UI Emoji font defines color versions of the Emoticon and other Emoji characters.

The standard and rich text controls support display color fonts. By default, the **IsColorFontEnabled** property is **true** and fonts with these additional layers are rendered in color. The default color font on the system is Segoe UI Emoji and the controls will fall back to this font to display the glyphs in color.

XAML

```
<TextBlock FontSize="30">Hello ☺⛄☂♨⛅</TextBlock>
```

The rendered text looks like this:

For more info, see the IsColorFontEnabled property.

Guidelines for line and paragraph separators

Applies to: TextBlock, RichTextBlock, multi-line TextBox, RichEditBox

Use the line separator character (0x2028) and the paragraph separator character (0x2029) to divide plain text. A new line is begun after each line separator. A new paragraph is begun after each paragraph separator.

It isn't necessary to start the first line or paragraph in a file with these characters or to end the last line or paragraph with them; doing so indicates that there is an empty line

or paragraph in that location.

Your app can use the line separator to indicate an unconditional end of line. However, line separators do not correspond to the separate carriage return and linefeed characters, or to a combination of these characters. Line separators must be processed separately from carriage return and linefeed characters.

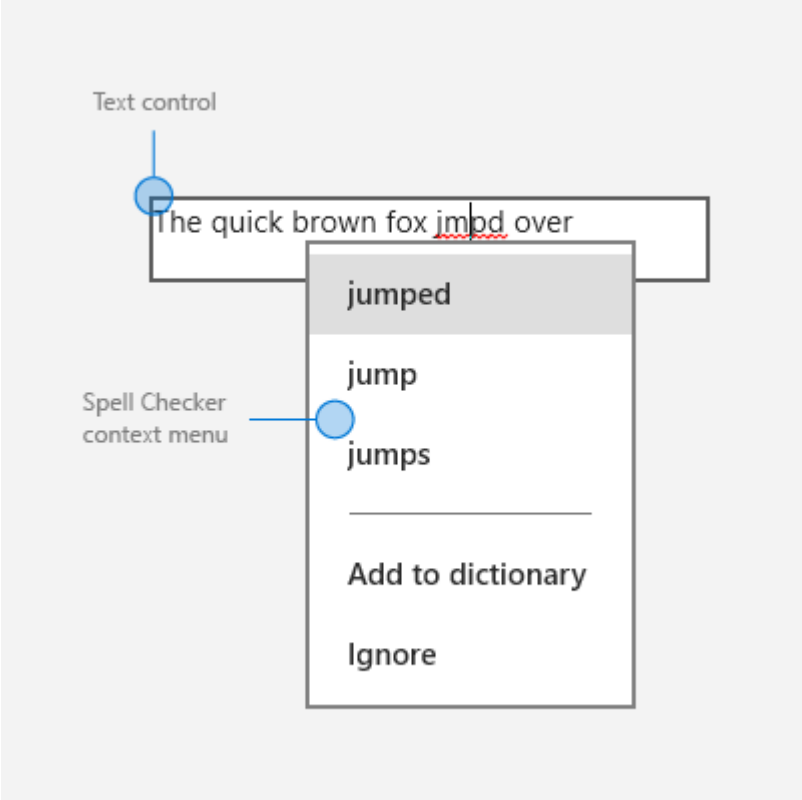
Your app can insert a paragraph separator between paragraphs of text. Use of this separator allows creation of plain text files that can be formatted with different line widths on different operating systems. The target system can ignore any line separators and break paragraphs only at the paragraph separators.

Guidelines for spell checking

Applies to: TextBox, RichEditBox

During text entry and editing, spell checking informs the user that a word is misspelled by highlighting it with a red squiggle and provides a way for the user to correct the misspelling.

Here's an example of the built-in spell checker:



Use spell checking with text input controls for these two purposes:

- **To auto-correct misspellings**

The spell checking engine automatically corrects misspelled words when it's confident about the correction. For example, the engine automatically changes "teh" to "the."

- **To show alternate spellings**

  When the spell checking engine is not confident about the corrections, it adds a red line under the misspelled word and displays the alternates in a context menu when you tap or right-click the word.

- Use spell checking to help users as they enter words or sentences into text input controls. Spell checking works with touch, mouse, and keyboard inputs.

- Don't use spell checking when a word is not likely to be in the dictionary or if users wouldn't value spell checking. For example, don't turn it on if the text box is intended to capture a telephone number or name.

- Don't disable spell checking just because the current spell checking engine doesn't support your app language. When the spell checker doesn't support a language, it doesn't do anything, so there's no harm in leaving the option on. Also, some users might use an Input Method Editor (IME) to enter another language into your app, and that language might be supported. For example, when building a Japanese language app, even though the spell checking engine might not currently recognize that language, don't turn spell checking off. The user may switch to an English IME and type English into the app; if spell checking is enabled, the English will get spell checked.

For TextBox and RichEditBox controls, spell checking is turned on by default. You can turn it off by setting the **IsSpellCheckEnabled** property to **false**.

Related articles

For designers

- Typography guidelines
- Segoe MDL2 icon list and guidelines
- Adding search

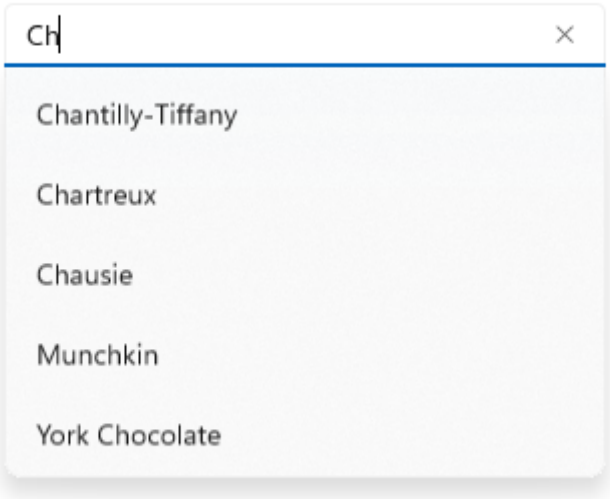
For developers (XAML)

- TextBox class
- Windows.UI.Xaml.Controls PasswordBox class
- String.Length property

Auto-suggest box

Article • 02/26/2025

Use an AutoSuggestBox to provide a list of suggestions for a user to select from as they type.



Is this the right control?

If you'd like a simple, customizable control that allows text search with a list of suggestions, then choose an auto-suggest box.

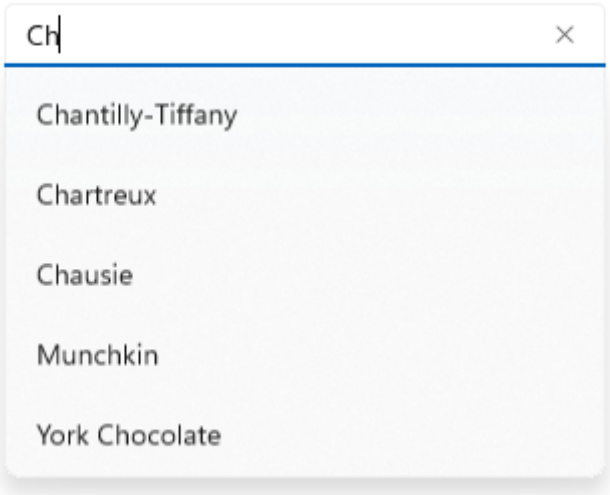
For more info about choosing the right text control, see the Text controls article.

Anatomy

The entry point for the auto-suggest box consists of an optional header and a text box with optional hint text:

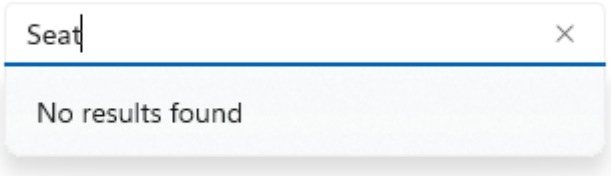
The auto-suggest results list populates automatically once the user starts to enter text. The results list can appear above or below the text entry box. A "clear all" button

appears:



Recommendations

- When using the auto-suggest box to perform searches and no search results exist for the entered text, display a single-line "No results" message as the result so that users know their search request executed:



Create an auto-suggest box

✓ **Important APIs:** AutoSuggestBox class, TextChanged event, SuggestionChose event, QuerySubmitted event

Open the WinUI 3 Gallery app and see the AutoSuggestBox in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

To use an AutoSuggestBox, you need to respond to 3 user actions.

- Text changed - When the user enters text, update the suggestion list.
- Suggestion chosen - When the user chooses a suggestion in the suggestion list, update the text box.
- Query submitted - When the user submits a query, show the query results.

Text changed

The TextChanged event occurs whenever the content of the text box is updated. Use the event args Reason property to determine whether the change was due to user input. If the change reason is **UserInput**, filter your data based on the input. Then, set the filtered data as the ItemsSource of the AutoSuggestBox to update the suggestion list.

To control how items are displayed in the suggestion list, you can use DisplayMemberPath or ItemTemplate.

- To display the text of a single property of your data item, set the DisplayMemberPath property to choose which property from your object to display in the suggestion list.
- To define a custom look for each item in the list, use the ItemTemplate property.

Suggestion chosen

When a user navigates through the suggestion list using the keyboard, you need to update the text in the text box to match.

You can set the TextMemberPath property to choose which property from your data object to display in the text box. If you specify a TextMemberPath, the text box is updated automatically. You should typically specify the same value for DisplayMemberPath and TextMemberPath so the text is the same in the suggestion list and the text box.

If you need to show more than a simple property, handle the SuggestionChosen event to populate the text box with custom text based on the selected item.

Query submitted

Handle the QuerySubmitted event to perform a query action appropriate to your app and show the result to the user.

The QuerySubmitted event occurs when a user commits a query string. The user can commit a query in one of these ways:

- While the focus is in the text box, press Enter or click the query icon. The event args ChosenSuggestion property is **null**.
- While the focus is in the suggestion list, press Enter, click, or tap an item. The event args ChosenSuggestion property contains the item that was selected from the list.

In all cases, the event args QueryText property contains the text from the text box.

Here is a simple AutoSuggestBox with the required event handlers.

XAML

```
<AutoSuggestBox PlaceholderText="Search" QueryIcon="Find" Width="200"
                TextChanged="AutoSuggestBox_TextChanged"
                QuerySubmitted="AutoSuggestBox_QuerySubmitted"
                SuggestionChosen="AutoSuggestBox_SuggestionChosen"/>
```

C#

```
private void AutoSuggestBox_TextChanged(AutoSuggestBox sender,
AutoSuggestBoxTextChangedEventArgs args)
{
    // Only get results when it was a user typing,
    // otherwise assume the value got filled in by TextMemberPath
    // or the handler for SuggestionChosen.
    if (args.Reason == AutoSuggestionBoxTextChangeReason.UserInput)
    {
        //Set the ItemsSource to be your filtered dataset
        //sender.ItemsSource = dataset;
    }
}


private void AutoSuggestBox_SuggestionChosen(AutoSuggestBox sender,
AutoSuggestBoxSuggestionChosenEventArgs args)
{
    // Set sender.Text. You can use args.SelectedItem to build your text
string.
}


private void AutoSuggestBox_QuerySubmitted(AutoSuggestBox sender,
AutoSuggestBoxQuerySubmittedEventArgs args)
{
    if (args.ChosenSuggestion != null)
    {
        // User selected an item from the suggestion list, take an action on
it here.
    }
    else
    {
        // Use args.QueryText to determine what to do.
```

```
    }
}
```

Use AutoSuggestBox for search

Use an AutoSuggestBox to provide a list of suggestions for a user to select from as they type.

By default, the text entry box doesn't have a query button shown. You can set the QueryIcon property to add a button with the specified icon on the right side of the text box. For example, to make the AutoSuggestBox look like a typical search box, add a 'find' icon, like this.

XAML

```
<AutoSuggestBox QueryIcon="Find"/>
```

Here's an AutoSuggestBox with a 'find' icon.

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** AutoSuggestBox class, TextChanged event, SuggestionChose event, QuerySubmitted event

- ✓ Open the WinUI 2 Gallery app and see the AutoSuggestBox in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related articles

- Text controls
- Spell checking
- TextBox class
- Windows.UI.Xaml.Controls PasswordBox class
- String.Length property

Text block

Article • 02/05/2025

Text block is the primary control for displaying read-only text in apps. You can use it to display single-line or multi-line text, inline hyperlinks, and text with formatting like bold, italic, or underlined.

Is this the right control?

A text block is typically easier to use and provides better text rendering performance than a rich text block, so it's preferred for most app UI text. You can easily access and use text from a text block in your app by getting the value of the Text property. It also provides many of the same formatting options for customizing how your text is rendered.

Although you can put line breaks in the text, text block is designed to display a single paragraph and doesn't support text indentation. Use a **RichTextBlock** when you need support for multiple paragraphs, multi-column text or other complex text layouts, or inline UI elements like images.

For more info about choosing the right text control, see the Text controls article.

Create a text block

✓ **Important APIs:** TextBlock class, Text property, Inlines property

Open the WinUI 3 Gallery app and see the TextBlock in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Here's how to define a simple TextBlock control and set its `Text` property to a string.

XAML

```
<TextBlock Text="Hello, world!" />
```

C#

```
TextBlock textBlock1 = new TextBlock();
textBlock1.Text = "Hello, world!";
```

Content model

There are two properties you can use to add content to a TextBlock: Text and Inlines.

The most common way to display text is to set the `Text` property to a string value, as shown in the previous example.

You can also add content by placing inline flow content elements in the `Inlines` property, like this. (`Inlines` is the default content property of a TextBlock, so you don't need to explicitly add it in XAML.)

XAML

```
<TextBlock>Text can be <Bold>bold</Bold>, <Underline>underlined</Underline>,
    <Italic>italic</Italic>, or a <Bold><Italic>combination</Italic></Bold>.
</TextBlock>
```

Elements derived from the Inline class, such as Bold, Italic, Run, Span, and LineBreak, enable different formatting for different parts of the text. For more info, see the Formatting text section. The inline Hyperlink element lets you add a hyperlink to your text. However, using `Inlines` also disables fast path text rendering, which is discussed in the next section.

Performance considerations

Whenever possible, XAML uses a more efficient code path to layout text. This fast path both decreases overall memory use and greatly reduces the CPU time to do text measuring and arranging. This fast path applies only to TextBlock, so it should be preferred when possible over RichTextBlock.

Certain conditions require TextBlock to fall back to a more feature-rich and CPU intensive code path for text rendering. To keep text rendering on the fast path, be sure to follow these guidelines when setting the properties listed here.

- Text: The most important condition is that the fast path is used only when you set text by explicitly setting the `Text` property, either in XAML or in code (as shown in the previous examples). Setting the text via TextBlock's `Inlines` collection (such as

`<TextBlock>Inline text</TextBlock>`) will disable the fast path, due to the potential complexity of multiple formats.

- CharacterSpacing: Only the default value of 0 is fast path.
- TextTrimming: Only the `None`, `CharacterEllipsis`, and `WordEllipsis` values are fast path. The `Clip` value disables the fast path.

> ⓘ Note
>
> **UWP Only:** Prior to Windows 10, version 1607, additional properties also affect the fast path. If your app is run on an earlier version of Windows, these conditions will cause your text to render on the slow path. For more info about versions, see Version adaptive code.
>
> - Typography: Only the default values for the various `Typography` properties are fast path.
> - LineStackingStrategy: If LineHeight is not 0, the `BaselineToBaseline` and `MaxHeight` values disable the fast path.
> - IsTextSelectionEnabled: Only `false` is fast path. Setting this property to `true` disables the fast path.

You can set the DebugSettings.IsTextPerformanceVisualizationEnabled property to `true` during debugging to determine whether text is using fast path rendering. When this property is set to `true`, the text that is on the fast path displays in a bright green color.

You typically set debug settings in the OnLaunched method override in the code-behind page for `App.xaml`, like this.

C#

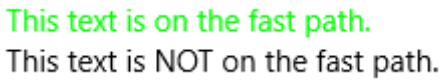
```
protected override void OnLaunched(LaunchActivatedEventArgs e)
{
#if DEBUG
    if (System.Diagnostics.Debugger.IsAttached)
    {
        this.DebugSettings.IsTextPerformanceVisualizationEnabled = true;
    }
#endif

// ...

}
```

In this example, the first TextBlock is rendered using the fast path, while the second is not.

XAML

```
<StackPanel>
    <TextBlock Text="This text is on the fast path."/>
    <TextBlock>This text is NOT on the fast path.</TextBlock>
<StackPanel/>
```

When you run this XAML in debug mode with IsTextPerformanceVisualizationEnabled set to `true`, the result looks like this.



> **⊗ Caution**
>
> The color of text that is not on the fast path is not changed. If you have text in your app with its color specified as bright green, it is still displayed in bright green when it's on the slower rendering path. Be careful to not confuse text that is set to green in the app with text that is on the fast path and green because of the debug settings.

Formatting text

Although the Text property stores plain text, you can apply various formatting options to the TextBlock control to customize how the text is rendered in your app. You can set standard control properties like FontFamily, FontSize, FontStyle, Foreground, and CharacterSpacing to change the look of the text. You can also use inline text elements and Typography attached properties to format your text. These options affect only how the TextBlock displays the text locally, so if you copy and paste the text into a rich text control, for example, no formatting is applied.

> **ⓘ Note**
>
> Remember, as noted in the previous section, inline text elements and non-default typography values are not rendered on the fast path.

Inline elements

The Microsoft.UI.Xaml.Documents namespace provides a variety of inline text elements that you can use to format your text, such as Bold, Italic, Run, Span, and LineBreak.

You can display a series of strings in a TextBlock, where each string has different formatting. You can do this by using a `Run` element to display each string with its formatting and by separating each `Run` element with a `LineBreak` element.

Here's how to define several differently formatted text strings in a TextBlock by using `Run` objects separated with a `LineBreak`.

XAML

```
<TextBlock FontFamily="Segoe UI" Width="400" Text="Sample text formatting
runs">
    <LineBreak/>
    <Run Foreground="Gray" FontFamily="Segoe UI Light" FontSize="24">
        Segoe UI Light 24
    </Run>
    <LineBreak/>
    <Run Foreground="Teal" FontFamily="Georgia" FontSize="18"
FontStyle="Italic">
        Georgia Italic 18
    </Run>
    <LineBreak/>
    <Run Foreground="Black" FontFamily="Arial" FontSize="14"
FontWeight="Bold">
        Arial Bold 14
    </Run>
</TextBlock>
```

Here's the result.

Typography

The attached properties of the Typography class provide access to a set of Microsoft OpenType typography properties. You can set these attached properties either on the TextBlock, or on individual inline text elements. These examples show both.

XAML

```
<TextBlock Text="Hello, world!"
           Typography.Capitals="SmallCaps"
```

```
            Typography.StylisticSet4="True"/>
```

C#

```
TextBlock textBlock1 = new TextBlock();
textBlock1.Text = "Hello, world!";
Typography.SetCapitals(textBlock1, FontCapitals.SmallCaps);
Typography.SetStylisticSet4(textBlock1, true);
```

XAML

```
<TextBlock>12 x <Run Typography.Fraction="Slashed">1/3</Run> = 4.
</TextBlock>
```

UWP and WinUI 2

> **ⓘ Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** TextBlock class, Text property, Inlines property
- ✓ Open the WinUI 2 Gallery app and see the TextBlock in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles, templates, and features for all controls.

Related articles

- Text controls
- TextBox class

- PasswordBox class
- String.Length property

Rich text block

Article • 02/26/2025

Rich text blocks provide several features for advanced text layout that you can use when you need support for paragraphs, inline UI elements, or complex text layouts.

Is this the right control?

Use a **RichTextBlock** when you need support for multiple paragraphs, multi-column or other complex text layouts, or inline UI elements like images.

Use a **TextBlock** to display most read-only text in your app. You can use it to display single-line or multi-line text, inline hyperlinks, and text with formatting like bold, italic, or underlined. TextBlock provides a simpler content model, so it's typically easier to use, and it can provide better text rendering performance than RichTextBlock. It's preferred for most app UI text. Although you can put line breaks in the text, TextBlock is designed to display a single paragraph and doesn't support text indentation.

For more info about choosing the right text control, see the Text controls article.

Recommendations

See Typography and Guidelines for fonts.

Create a rich text block

✓ **Important APIs:** RichTextBlock class, RichTextBlockOverflow class, Paragraph class, Typography class

Open the WinUI 3 Gallery app and see the RichTextBlock in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

The content property of RichTextBlock is the Blocks property, which supports paragraph based text via the Paragraph element. It doesn't have a **Text** property that you can use to easily access the control's text content in your app. However, RichTextBlock provides several unique features that TextBlock doesn't provide.

RichTextBlock supports:

- Multiple paragraphs. Set the indentation for paragraphs by setting the TextIndent property.
- Inline UI elements. Use an InlineUIContainer to display UI elements, such as images, inline with your text.
- Overflow containers. Use RichTextBlockOverflow elements to create multi-column text layouts.

Paragraphs

You use Paragraph elements to define the blocks of text to display within a RichTextBlock control. Every RichTextBlock should include at least one Paragraph.

You can set the indent amount for all paragraphs in a RichTextBlock by setting the RichTextBlock.TextIndent property. You can override this setting for specific paragraphs in a RichTextBlock by setting the Paragraph.TextIndent property to a different value.

XAML

```
<RichTextBlock TextIndent="12">
  <Paragraph TextIndent="24">First paragraph.</Paragraph>
  <Paragraph>Second paragraph.</Paragraph>
  <Paragraph>Third paragraph. <Bold>With an inline.</Bold></Paragraph>
</RichTextBlock>
```

Inline UI elements

The InlineUIContainer class lets you embed any UIElement inline with your text. A common scenario is to place an Image inline with your text, but you can also use interactive elements, like a Button or CheckBox.

If you want to embed more than one element inline in the same position, consider using a panel as the single InlineUIContainer child, and then place the multiple elements within that panel.

This example shows how to use an InlineUIContainer to insert an image into a RichTextBlock.

XAML

```
<RichTextBlock>
    <Paragraph>
        <Italic>This is an inline image.</Italic>
```

```
        <InlineUIContainer>
            <Image Source="Assets/Square44x44Logo.png" Height="30"
Width="30"/>
        </InlineUIContainer>
        Mauris auctor tincidunt auctor.
    </Paragraph>
</RichTextBlock>
```

Overflow containers

You can use a RichTextBlock with RichTextBlockOverflow elements to create multi-column or other advanced page layouts. The content for a RichTextBlockOverflow element always comes from a RichTextBlock element. You link RichTextBlockOverflow elements by setting them as the OverflowContentTarget of a RichTextBlock or another RichTextBlockOverflow.

Here's a simple example that creates a two column layout. See the Examples section for a more complex example.

XAML

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>
    <RichTextBlock Grid.Column="0"
                   OverflowContentTarget="{Binding
ElementName=overflowContainer}" >
        <Paragraph>
            Proin ac metus at quam luctus ultricies.
        </Paragraph>
    </RichTextBlock>
    <RichTextBlockOverflow x:Name="overflowContainer" Grid.Column="1"/>
</Grid>
```

Formatting text

Although the RichTextBlock stores plain text, you can apply various formatting options to customize how the text is rendered in your app. You can set standard control properties like FontFamily, FontSize, FontStyle, Foreground, and CharacterSpacing to change the look of the text. You can also use inline text elements and Typography attached properties to format your text. These options affect only how the RichTextBlock displays the text locally, so if you copy and paste the text into a rich text control, for example, no formatting is applied.

Inline elements

The Microsoft.UI.Xaml.Documents namespace provides a variety of inline text elements that you can use to format your text, such as Bold, Italic, Run, Span, and LineBreak. A typical way to apply formatting to sections of text is to place the text in a Run or Span element, and then set properties on that element.

Here's a Paragraph with the first phrase shown in bold, blue, 16pt text.

XAML

```
<Paragraph>
    <Bold><Span Foreground="DarkSlateBlue" FontSize="16">Lorem ipsum dolor
sit amet</Span></Bold>
    , consectetur adipiscing elit.
</Paragraph>
```

Typography

The attached properties of the Typography class provide access to a set of Microsoft OpenType typography properties. You can set these attached properties either on the RichTextBlock, or on individual inline text elements, as shown here.

XAML

```
<RichTextBlock Typography.StylisticSet4="True">
    <Paragraph>
        <Span Typography.Capitals="SmallCaps">Lorem ipsum dolor sit
amet</Span>
        , consectetur adipiscing elit.
    </Paragraph>
</RichTextBlock>
```

UWP and WinUI 2

ⓘ Important

The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** RichTextBlock class, RichTextBlockOverflow class, Paragraph class, Typography class
- ✓ Open the WinUI 2 Gallery app and see the RichTextBox in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related articles

Text controls

For designers

- Guidelines for spell checking
- Adding search
- Guidelines for text input

For developers (XAML)

- TextBox class
- PasswordBox class

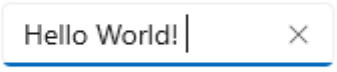
For developers (other)

- String.Length property

Text box

Article • 02/26/2025

The TextBox control lets a user type text into an app. It's typically used to capture a single line of text, but can be configured to capture multiple lines of text. The text displays on the screen in a simple, uniform, plaintext format.



TextBox has a number of features that can simplify text entry. It comes with a familiar, built-in context menu with support for copying and pasting text. The "clear all" button lets a user quickly delete all text that has been entered. It also has spell checking capabilities built in and enabled by default.

Is this the right control?

Use a **TextBox** control to let a user enter and edit unformatted text, such as in a form. You can use the Text property to get and set the text in a TextBox.

You can make a TextBox read-only, but this should be a temporary, conditional state. If the text is never editable, consider using a TextBlock instead.

Use a PasswordBox control to collect a password or other private data, such as a Social Security number. A password box looks like a text input box, except that it renders bullets in place of the text that has been entered.

Use an AutoSuggestBox control to let the user enter search terms or to show the user a list of suggestions to choose from as they type.

Use a RichEditBox to display and edit rich text files.

For more info about choosing the right text control, see the Text controls article.

Recommendations

- Use a label or placeholder text if the purpose of the text box isn't clear. A label is visible whether or not the text input box has a value. Placeholder text is displayed

inside the text input box and disappears once a value has been entered.
- Give the text box an appropriate width for the range of values that can be entered. Word length varies between languages, so take localization into account if you want your app to be world-ready.
- A text input box is typically single-line (`TextWrap = "NoWrap"`). When users need to enter or edit a long string, set the text input box to multi-line (`TextWrap = "Wrap"`).
- Generally, a text input box is used for editable text. But you can make a text input box read-only so that its content can be read, selected, and copied, but not edited.
- If you need to reduce clutter in a view, consider making a set of text input boxes appear only when a controlling checkbox is checked. You can also bind the enabled state of a text input box to a control such as a checkbox.
- Consider how you want a text input box to behave when it contains a value and the user taps it. The default behavior is appropriate for editing the value rather than replacing it; the insertion point is placed between words and nothing is selected. If replacing is the most common use case for a given text input box, you can select all the text in the field whenever the control receives focus, and typing replaces the selection.

Single-line input boxes

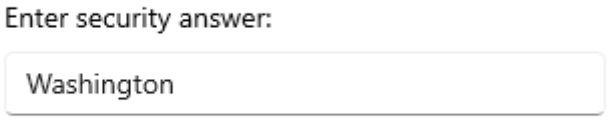
- Use several single-line text boxes to capture many small pieces of text information. If the text boxes are related in nature, group those together.

- Make the size of single-line text boxes slightly wider than the longest anticipated input. If doing so makes the control too wide, separate it into two controls. For example, you could split a single address input into "Address line 1" and "Address line 2".

- Set a maximum length for characters that can be entered. If the backing data source doesn't allow a long input string, limit the input and use a validation popup to let users know when they reach the limit.

- Use single-line text input controls to gather small pieces of text from users.

  The following example shows a single-line text box to capture an answer to a security question. The answer is expected to be short, and so a single-line text box is appropriate here.

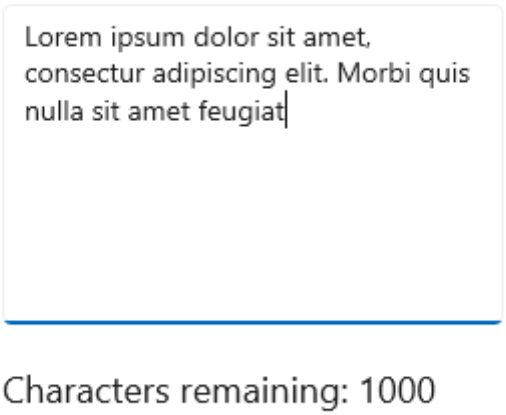


- Use a set of short, fixed-sized, single-line text input controls to enter data with a specific format.

- Use a single-line, unconstrained text input control to enter or edit strings, combined with a command button that helps users select valid values.

Multi-line text input controls

- When you create a rich text box, provide styling buttons and implement their actions.
- Use a font that's consistent with the style of your app.
- Make the height of the text control tall enough to accommodate typical entries.
- When capturing long spans of text with a maximum character or word count, use a plain text box and provide a live-running counter to show the user how many characters or words they have left before they reach the limit. You'll need to create the counter yourself; place it below the text box and dynamically update it as the user enters each character or word.



- Don't let your text input controls grow in height while users type.
- Don't use a multi-line text box when users only need a single line.
- Don't use a rich text control if a plain text control is adequate.

Create a text box

✔ **Important APIs:** TextBox class, Text property

Open the WinUI 3 Gallery app and see the TextBox in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Here's the XAML for a simple text box with a header and placeholder text.

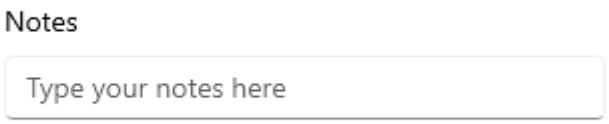
XAML

```
<TextBox Width="500" Header="Notes" PlaceholderText="Type your notes here"/>
```

C#

```
TextBox textBox = new TextBox();
textBox.Width = 300;
textBox.Header = "Notes";
textBox.PlaceholderText = "Type your notes here";
// Add the TextBox to the visual tree.
rootGrid.Children.Add(textBox);
```

Here's the text box that results from this XAML.



Use a text box for data input in a form

It's common to use a text box to accept data input on a form, and use the Text property to get the complete text string from the text box. You typically use an event like a submit button click to access the Text property, but you can handle the TextChanged or TextChanging event if you need to do something when the text changes.

This example shows how to get and set the current content of a text box.

XAML

```
<TextBox name="SampleTextBox" Text="Sample Text"/>
```

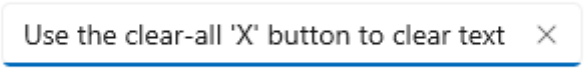
C#

```
string sampleText = SampleTextBox.Text;
...
SampleTextBox.Text = "Sample text retrieved";
```

You can add a Header (or label) and PlaceholderText (or watermark) to the text box to give the user an indication of what the text box is for. To customize the look of the header, you can set the HeaderTemplate property instead of Header. *For design info, see Guidelines for labels*.

You can restrict the number of characters the user can type by setting the MaxLength property. However, MaxLength does not restrict the length of pasted text. Use the Paste event to modify pasted text if this is important for your app.

The text box includes a clear all button ("X") that appears when text is entered in the box. When a user clicks the "X", the text in the text box is cleared. It looks like this.



The clear all button is shown only for editable, single-line text boxes that contain text and have focus.

The clear all button is not shown in any of these cases:

- **IsReadOnly** is **true**
- **AcceptsReturn** is **true**
- **TextWrap** has a value other than **NoWrap**

This example shows how to get and set the current content of a text box.

XAML

```
<TextBox name="SampleTextBox" Text="Sample Text"/>
```

C#

```
string sampleText = SampleTextBox.Text;
...
SampleTextBox.Text = "Sample text retrieved";
```

Make a text box read-only

You can make a text box read-only by setting the IsReadOnly property to **true**. You typically toggle this property in your app code based on conditions in your app. If need text that is always read-only, consider using a TextBlock instead.

You can make a TextBox read-only by setting the IsReadOnly property to true. For example, you might have a TextBox for a user to enter comments that is enabled only under certain conditions. You can make the TextBox read-only until the conditions are met. If you need only to display text, consider using a TextBlock or RichTextBlock instead.

A read-only text box looks the same as a read/write text box, so it might be confusing to a user. A user can select and copy text. IsEnabled

Enable multi-line input

There are two properties that you can use to control whether the text box displays text on more than one line. You typically set both properties to make a multi-line text box.

- To let the text box allow and display the newline or return characters, set the AcceptsReturn property to **true**.
- To enable text wrapping, set the TextWrapping property to **Wrap**. This causes the text to wrap when it reaches the edge of the text box, independent of line separator characters.

> **Note**
>
> TextBox and RichEditBox don't support the **WrapWholeWords** value for their TextWrapping properties. If you try to use WrapWholeWords as a value for TextBox.TextWrapping or RichEditBox.TextWrapping an invalid argument exception is thrown.

A multi-line text box will continue to grow vertically as text is entered unless it's constrained by its Height or MaxHeight property, or by a parent container. You should test that a multi-line text box doesn't grow beyond its visible area, and constrain its growth if it does. We recommend that you always specify an appropriate height for a multi-line text box, and not let it grow in height as the user types.

Scrolling using a scroll-wheel or touch is automatically enabled when needed. However, the vertical scrollbars are not visible by default. You can show the vertical scrollbars by setting the ScrollViewer.VerticalScrollBarVisibility to **Auto** on the embedded ScrollViewer, as shown here.

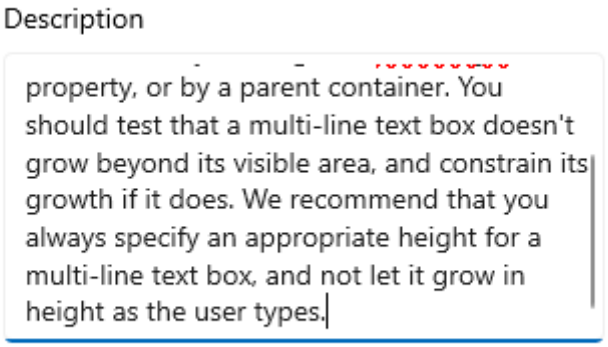
XAML

```
<TextBox AcceptsReturn="True" TextWrapping="Wrap"
         MaxHeight="172" Width="300" Header="Description"
         ScrollViewer.VerticalScrollBarVisibility="Auto"/>
```

C#

```
TextBox textBox = new TextBox();
textBox.AcceptsReturn = true;
textBox.TextWrapping = TextWrapping.Wrap;
textBox.MaxHeight = 172;
textBox.Width = 300;
textBox.Header = "Description";
ScrollViewer.SetVerticalScrollBarVisibility(textBox,
ScrollBarVisibility.Auto);
```

Here's what the text box looks like after text is added.



Format the text display

Use the TextAlignment property to align text within a text box. To align the text box within the layout of the page, use the HorizontalAlignment and VerticalAlignment properties.

While the text box supports only unformatted text, you can customize how the text is displayed in the text box to match your branding. You can set standard Control properties like FontFamily, FontSize, FontStyle, Background, Foreground, and CharacterSpacing to change the look of the text. These properties affect only how the text box displays the text locally, so if you were to copy and paste the text into a rich text control, for example, no formatting would be applied.

This example shows a read-only text box with several properties set to customize the appearance of the text.

XAML

```
<TextBox Text="Sample Text" IsReadOnly="True"
         FontFamily="Verdana" FontSize="24"
         FontWeight="Bold" FontStyle="Italic"
         CharacterSpacing="200" Width="300"
         Foreground="Blue" Background="Beige"/>
```

C#

```
TextBox textBox = new TextBox();
textBox.Text = "Sample Text";
textBox.IsReadOnly = true;
textBox.FontFamily = new FontFamily("Verdana");
textBox.FontSize = 24;
textBox.FontWeight = Windows.UI.Text.FontWeights.Bold;
```

```
textBox.FontStyle = Windows.UI.Text.FontStyle.Italic;
textBox.CharacterSpacing = 200;
textBox.Width = 300;
textBox.Background = new SolidColorBrush(Windows.UI.Colors.Beige);
textBox.Foreground = new SolidColorBrush(Windows.UI.Colors.Blue);
// Add the TextBox to the visual tree.
rootGrid.Children.Add(textBox);
```

The resulting text box looks like this.

Modify the context menu

By default, the commands shown in the text box context menu depend on the state of the text box. For example, the following commands can be shown when the text box is editable.

Expand table

Command	Shown when...
Copy	text is selected.
Cut	text is selected.
Paste	the clipboard contains text.
Select all	the TextBox contains text.
Undo	text has been changed.

To modify the commands shown in the context menu, handle the ContextMenuOpening event. For an example of this, see the **Customizing RichEditBox's CommandBarFlyout - adding 'Share'** example in the WinUI 2 Gallery. For design info, see Guidelines for context menus.

Select, copy, and paste

You can get or set the selected text in a text box using the SelectedText property. Use the SelectionStart and SelectionLength properties, and the Select and SelectAll methods,

to manipulate the text selection. Handle the SelectionChanged event to do something when the user selects or de-selects text. You can change the color used to highlight the selected text by setting the SelectionHighlightColor property.

TextBox supports copy and paste by default. You can provide custom handling of the Paste event on editable text controls in your app. For example, you might remove the line breaks from a multi-line address when pasting it into a single-line search box. Or, you might check the length of the pasted text and warn the user if it exceeds the maximum length that can be saved to a database. For more info and examples, see the Paste event.

Here, we have an example of these properties and methods in use. When you select text in the first text box, the selected text is displayed in the second text box, which is read-only. The values of the SelectionLength and SelectionStart properties are shown in two text blocks. This is done using the SelectionChanged event.

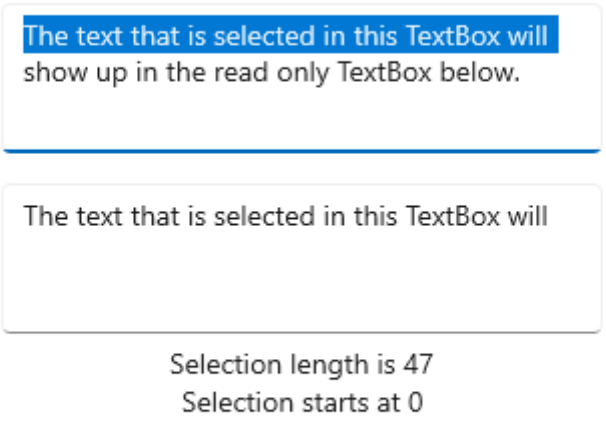
XAML

```
<StackPanel>
   <TextBox x:Name="textBox1" Height="75" Width="300" Margin="10"
         Text="The text that is selected in this TextBox will show up in the
read only TextBox below."
         TextWrapping="Wrap" AcceptsReturn="True"
         SelectionChanged="TextBox1_SelectionChanged" />
   <TextBox x:Name="textBox2" Height="75" Width="300" Margin="5"
         TextWrapping="Wrap" AcceptsReturn="True" IsReadOnly="True"/>
   <TextBlock x:Name="label1" HorizontalAlignment="Center"/>
   <TextBlock x:Name="label2" HorizontalAlignment="Center"/>
</StackPanel>
```

C#

```
private void TextBox1_SelectionChanged(object sender, RoutedEventArgs e)
{
    textBox2.Text = textBox1.SelectedText;
    label1.Text = "Selection length is " +
textBox1.SelectionLength.ToString();
    label2.Text = "Selection starts at " +
textBox1.SelectionStart.ToString();
}
```

Here's the result of this code.



Choose the right keyboard for your text control

To help users to enter data using the touch keyboard, or Soft Input Panel (SIP), you can set the input scope of the text control to match the kind of data the user is expected to enter.

The touch keyboard can be used for text entry when your app runs on a device with a touch screen. The touch keyboard is invoked when the user taps on an editable input field, such as a TextBox or RichEditBox. You can make it much faster and easier for users to enter data in your app by setting the input scope of the text control to match the kind of data you expect the user to enter. The input scope provides a hint to the system about the type of text input expected by the control so the system can provide a specialized touch keyboard layout for the input type.

For example, if a text box is used only to enter a 4-digit PIN, set the InputScope property to **Number**. This tells the system to show the number keypad layout, which makes it easier for the user to enter the PIN.

> **Important** The input scope does not cause any input validation to be performed, and does not prevent the user from providing any input through a hardware keyboard or other input device. You are still responsible for validating the input in your code as needed.

Other properties that affect the touch keyboard are IsSpellCheckEnabled, IsTextPredictionEnabled, and PreventKeyboardDisplayOnProgrammaticFocus. (IsSpellCheckEnabled also affects the TextBox when a hardware keyboard is used.)

For more info and examples, see Use input scope to change the touch keyboard and the property documentation.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** TextBox class, Text property
- ✓ Open the WinUI 2 Gallery app and see the TextBox in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related articles

- Text controls
- Guidelines for spell checking
- Adding search
- Guidelines for text input
- TextBox class
- PasswordBox class
- String.Length property

Rich edit box

Article • 02/26/2025

You can use a RichEditBox control to enter and edit rich text documents that contain formatted text, hyperlinks, and images. You can make a RichEditBox read-only by setting its IsReadOnly property to **true**.

Is this the right control?

Use a **RichEditBox** to display and edit text files. You don't use a RichEditBox to get user input into you app the way you use other standard text input boxes. Rather, you use it to work with text files that are separate from your app. You typically save text entered into a RichEditBox to a .rtf file.

- If the primary purpose of the multi-line text box is for creating read-only documents (such as blog entries or the contents of an email message), and those documents require rich text, use a rich text block instead.
- When capturing text that will only be consumed and not redisplayed to users, use a plain text input control.
- For all other scenarios, use a plain text input control.

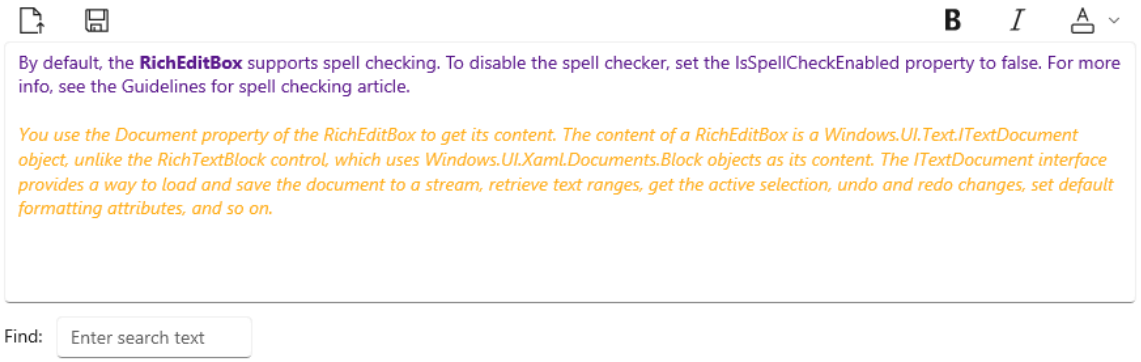
For more info about choosing the right text control, see the Text controls article.

Recommendations

- When you create a rich text box, provide styling buttons and implement their actions.
- Use a font that's consistent with the style of your app.
- Make the height of the text control tall enough to accommodate typical entries.
- Don't let your text input controls grow in height while users type.
- Don't use a multi-line text box when users only need a single line.
- Don't use a rich text control if a plain text control is adequate.

Examples

This rich edit box has a rich text document open in it. The formatting and file buttons aren't part of the rich edit box, but you should provide at least a minimal set of styling buttons and implement their actions.



Create a rich edit box

✔ **Important APIs:** RichEditBox class, Document property, IsReadOnly property, IsSpellCheckEnabled property

Open the WinUI 3 Gallery app and see the RichEditBox in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

By default, the RichEditBox supports spell checking. To disable the spell checker, set the IsSpellCheckEnabled property to **false**. For more info, see the Guidelines for spell checking article.

You use the Document property of the RichEditBox to get its content. The content of a RichEditBox is an ITextDocument object, unlike the RichTextBlock control, which uses Block objects as its content. The ITextDocument interface provides a way to load and save the document to a stream, retrieve text ranges, get the active selection, undo and redo changes, set default formatting attributes, and so on.

This example shows how to edit, load, and save a Rich Text Format (.rtf) file in a RichEditBox.

XAML

```
<RelativePanel Margin="20" HorizontalAlignment="Stretch">
    <RelativePanel.Resources>
        <Style TargetType="AppBarButton">
            <Setter Property="IsCompact" Value="True"/>
        </Style>
    </RelativePanel.Resources>
    <AppBarButton x:Name="openFileButton" Icon="OpenFile"
```

```
                Click="OpenButton_Click" ToolTipService.ToolTip="Open
file"/>
    <AppBarButton Icon="Save" Click="SaveButton_Click"
                  ToolTipService.ToolTip="Save file"
                  RelativePanel.RightOf="openFileButton" Margin="8,0,0,0"/>

    <AppBarButton Icon="Bold" Click="BoldButton_Click"
ToolTipService.ToolTip="Bold"
                  RelativePanel.LeftOf="italicButton" Margin="0,0,8,0"/>
    <AppBarButton x:Name="italicButton" Icon="Italic"
Click="ItalicButton_Click"
                  ToolTipService.ToolTip="Italic"
RelativePanel.LeftOf="underlineButton" Margin="0,0,8,0"/>
    <AppBarButton x:Name="underlineButton" Icon="Underline"
Click="UnderlineButton_Click"
                  ToolTipService.ToolTip="Underline"
RelativePanel.AlignRightWithPanel="True"/>

    <RichEditBox x:Name="editor" Height="200"
RelativePanel.Below="openFileButton"
                 RelativePanel.AlignLeftWithPanel="True"
RelativePanel.AlignRightWithPanel="True"/>
</RelativePanel>
```

C#

```
private async void OpenButton_Click(object sender, RoutedEventArgs e)
{
    // Open a text file.
    Windows.Storage.Pickers.FileOpenPicker open =
        new Windows.Storage.Pickers.FileOpenPicker();
    open.SuggestedStartLocation =
        Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
    open.FileTypeFilter.Add(".rtf");

    Windows.Storage.StorageFile file = await open.PickSingleFileAsync();

    if (file != null)
    {
        try
        {
            Windows.Storage.Streams.IRandomAccessStream randAccStream =
        await file.OpenAsync(Windows.Storage.FileAccessMode.Read);

            // Load the file into the Document property of the RichEditBox.

editor.Document.LoadFromStream(Windows.UI.Text.TextSetOptions.FormatRtf,
randAccStream);
        }
        catch (Exception)
        {
            ContentDialog errorDialog = new ContentDialog()
            {
```

```
                Title = "File open error",
                Content = "Sorry, I couldn't open the file.",
                PrimaryButtonText = "Ok"
            };

            await errorDialog.ShowAsync();
        }
    }
}

private async void SaveButton_Click(object sender, RoutedEventArgs e)
{
    Windows.Storage.Pickers.FileSavePicker savePicker = new
Windows.Storage.Pickers.FileSavePicker();
    savePicker.SuggestedStartLocation =
Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;

    // Dropdown of file types the user can save the file as
    savePicker.FileTypeChoices.Add("Rich Text", new List<string>() { ".rtf"
});

    // Default file name if the user does not type one in or select a file
to replace
    savePicker.SuggestedFileName = "New Document";

    Windows.Storage.StorageFile file = await savePicker.PickSaveFileAsync();
    if (file != null)
    {
        // Prevent updates to the remote version of the file until we
        // finish making changes and call CompleteUpdatesAsync.
        Windows.Storage.CachedFileManager.DeferUpdates(file);
        // write to file
        Windows.Storage.Streams.IRandomAccessStream randAccStream =
            await file.OpenAsync(Windows.Storage.FileAccessMode.ReadWrite);


editor.Document.SaveToStream(Windows.UI.Text.TextGetOptions.FormatRtf,
randAccStream);

        // Let Windows know that we're finished changing the file so the
        // other app can update the remote version of the file.
        Windows.Storage.Provider.FileUpdateStatus status = await
Windows.Storage.CachedFileManager.CompleteUpdatesAsync(file);
        if (status != Windows.Storage.Provider.FileUpdateStatus.Complete)
        {
            Windows.UI.Popups.MessageDialog errorBox =
                new Windows.UI.Popups.MessageDialog("File " + file.Name + "
couldn't be saved.");
            await errorBox.ShowAsync();
        }
    }
}

private void BoldButton_Click(object sender, RoutedEventArgs e)
{
```

```
    Windows.UI.Text.ITextSelection selectedText = editor.Document.Selection;
    if (selectedText != null)
    {
        Windows.UI.Text.ITextCharacterFormat charFormatting =
selectedText.CharacterFormat;
        charFormatting.Bold = Windows.UI.Text.FormatEffect.Toggle;
        selectedText.CharacterFormat = charFormatting;
    }
}

private void ItalicButton_Click(object sender, RoutedEventArgs e)
{
    Windows.UI.Text.ITextSelection selectedText = editor.Document.Selection;
    if (selectedText != null)
    {
        Windows.UI.Text.ITextCharacterFormat charFormatting =
selectedText.CharacterFormat;
        charFormatting.Italic = Windows.UI.Text.FormatEffect.Toggle;
        selectedText.CharacterFormat = charFormatting;
    }
}

private void UnderlineButton_Click(object sender, RoutedEventArgs e)
{
    Windows.UI.Text.ITextSelection selectedText = editor.Document.Selection;
    if (selectedText != null)
    {
        Windows.UI.Text.ITextCharacterFormat charFormatting =
selectedText.CharacterFormat;
        if (charFormatting.Underline == Windows.UI.Text.UnderlineType.None)
        {
            charFormatting.Underline = Windows.UI.Text.UnderlineType.Single;
        }
        else {
            charFormatting.Underline = Windows.UI.Text.UnderlineType.None;
        }
        selectedText.CharacterFormat = charFormatting;
    }
}
```

Choose the right keyboard for your text control

To help users to enter data using the touch keyboard, or Soft Input Panel (SIP), you can set the input scope of the text control to match the kind of data the user is expected to enter. The default keyboard layout is usually appropriate for working with rich text documents.

For more info about how to use input scopes, see Use input scope to change the touch keyboard.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** RichEditBox class, Document property, IsReadOnly property, IsSpellCheckEnabled property
- ✓ Open the WinUI 2 Gallery app and see the RichEditBox in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related articles

- Text controls
- Guidelines for spell checking
- Guidelines for text input
- TextBox class
- PasswordBox class

Password box

Article • 02/26/2025

A password box is a text input box that conceals the characters typed into it for the purpose of privacy. A password box looks like a text box, except that it renders placeholder characters in place of the text that has been entered. You can configure the placeholder character.

By default, the password box provides a way for the user to view their password by holding down a reveal button. You can disable the reveal button, or provide an alternate mechanism to reveal the password, such as a check box.

Is this the right control?

Use a **PasswordBox** control to collect a password or other private data, such as a Social Security number.

For more info about choosing the right text control, see the Text controls article.

Recommendations

- Use a label or placeholder text if the purpose of the password box isn't clear. A label is visible whether or not the text input box has a value. Placeholder text is displayed inside the text input box and disappears once a value has been entered.
- Give the password box an appropriate width for the range of values that can be entered. Word length varies between languages, so take localization into account if you want your app to be world-ready.
- Don't put another control right next to a password input box. The password box has a password reveal button for users to verify the passwords they have typed, and having another control right next to it might make users accidentally reveal their passwords when they try to interact with the other control. To prevent this from happening, put some spacing between the password in put box and the other control, or put the other control on the next line.

- Consider presenting two password boxes for account creation: one for the new password, and a second to confirm the new password.
- Only show a single password box for logins.
- When a password box is used to enter a PIN, consider providing an instant response as soon as the last number is entered instead of using a confirmation button.

Examples

The password box has several states, including these notable ones.

A password box at rest can show hint text so that the user knows its purpose:

When the user types in a password box, the default behavior is to show bullets that hide the text being entered:

Pressing the "reveal" button on the right gives a peek at the password text being entered:

Create a password box

✓ **Important APIs:** PasswordBox class, Password property, PasswordChar property, PasswordRevealMode property, PasswordChanged event

Open the WinUI 3 Gallery app and see PasswordBox in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Use the Password property to get or set the contents of the PasswordBox. You can do this in the handler for the PasswordChanged event to perform validation while the user enters the password. Or, you can use another event, like a button Click, to perform validation after the user completes the text entry.

Here's the XAML for a password box control that demonstrates the default look of the PasswordBox. When the user enters a password, you check to see if it's the literal value, "Password". If it is, you display a message to the user.

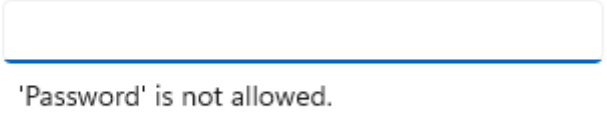
XAML

```
<StackPanel>
  <PasswordBox x:Name="passwordBox" Width="200" MaxLength="16"
               PasswordChanged="passwordBox_PasswordChanged"/>

  <TextBlock x:Name="statusText" Margin="10" HorizontalAlignment="Center" />
</StackPanel>
```

C#

```
private void passwordBox_PasswordChanged(object sender, RoutedEventArgs e)
{
    if (passwordBox.Password == "Password")
    {
        statusText.Text = "'Password' is not allowed as a password.";
    }
    else
    {
        statusText.Text = string.Empty;
    }
}
```

Here's the result when this code runs and the user enters "Password".



Password character

You can change the character used to mask the password by setting the PasswordChar property. Here, the default bullet is replaced with a pound sign.

XAML

```
<PasswordBox x:Name="passwordBox" Width="300" PasswordChar="#"/>
```

The result looks like this.

Headers and placeholder text

You can use the Header and PlaceholderText properties to provide context for the PasswordBox. This is especially useful when you have multiple boxes, such as on a form to change a password.

XAML

```
<PasswordBox x:Name="passwordBox" Width="200" Header="Password"
PlaceholderText="Enter your password"/>
```

Maximum length

Specify the maximum number of characters that the user can enter by setting the MaxLength property. There is no property to specify a minimum length, but you can check the password length, and perform any other validation, in your app code.

Password reveal mode

The PasswordBox has a built-in button that the user can press to display the password text. Here's the result of the user's action. When the user releases it, the password is automatically hidden again.

Peek mode

By default, the password reveal button (or "peek" button) is shown. The user must continuously press the button to view the password, so that a high level of security is maintained.

The value of the PasswordRevealMode property is not the only factor that determines whether a password reveal button is visible to the user. Other factors include whether the control is displayed above a minimum width, whether the PasswordBox has focus, and whether the text entry field contains at least one character. The password reveal button is shown only when the PasswordBox receives focus for the first time and a character is entered. If the PasswordBox loses focus and then regains focus, the reveal button is not shown again unless the password is cleared and character entry starts over.

Hidden and Visible modes

The other PasswordRevealMode enumeration values, **Hidden** and **Visible**, hide the password reveal button and let you programmatically manage whether the password is obscured.

To always obscure the password, set PasswordRevealMode to Hidden. Unless you need the password to be always obscured, you can provide a custom UI to let the user toggle the PasswordRevealMode between Hidden and Visible. For example, you can use a check box to toggle whether the password is obscured, as shown in the following example. You can also use other controls, like ToggleButton, to let the user switch modes.

This example shows how to use a CheckBox to let a user switch the reveal mode of a PasswordBox.

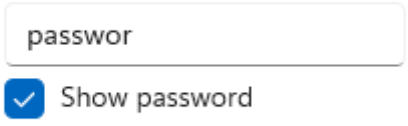
XAML

```
<StackPanel Width="200">
    <PasswordBox Name="passwordBox1"
                 PasswordRevealMode="Hidden"/>
    <CheckBox Name="revealModeCheckBox" Content="Show password"
              IsChecked="False"
              Checked="CheckBox_Changed" Unchecked="CheckBox_Changed"/>
</StackPanel>
```

C#

```
private void CheckBox_Changed(object sender, RoutedEventArgs e)
{
    if (revealModeCheckBox.IsChecked == true)
    {
        passwordBox1.PasswordRevealMode = PasswordRevealMode.Visible;
    }
    else
    {
        passwordBox1.PasswordRevealMode = PasswordRevealMode.Hidden;
    }
}
```

This PasswordBox looks like this.



Choose the right keyboard for your text control

To help users to enter data using the touch keyboard, or Soft Input Panel (SIP), you can set the input scope of the text control to match the kind of data the user is expected to enter. PasswordBox supports only the **Password** and **NumericPin** input scope values. Any other value is ignored.

For more info about how to use input scopes, see Use input scope to change the touch keyboard.

UWP and WinUI 2

> **Important**
>
> The information and examples in this article are optimized for apps that use the Windows App SDK and WinUI 3, but are generally applicable to UWP apps that use WinUI 2. See the UWP API reference for platform specific information and examples.
>
> This section contains information you need to use the control in a UWP or WinUI 2 app.

APIs for this control exist in the Windows.UI.Xaml.Controls namespace.

- ✓ **UWP APIs:** PasswordBox class, Password property, PasswordChar property, PasswordRevealMode property, PasswordChanged event
- ✓ Open the WinUI 2 Gallery app and see PasswordBox in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

We recommend using the latest WinUI 2 to get the most current styles and templates for all controls. WinUI 2.2 or later includes a new template for this control that uses rounded corners. For more info, see Corner radius.

Related articles

Text controls

- Guidelines for spell checking
- Adding search
- Guidelines for text input
- TextBox class
- class
- String.Length property

Number box

Article • 02/26/2025

Represents a control that can be used to display and edit numbers. This supports validation, increment stepping, and computing inline calculations of basic equations, such as multiplication, division, addition, and subtraction.

Is this the right control?

You can use a NumberBox control to capture and display mathematic input. If you need an editable text box that accepts more than numbers, use the TextBox control. If you need an editable text box that accepts passwords or other sensitive input, see PasswordBox. If you need a text box to enter search terms, see AutoSuggestBox. If you need to enter or edit formatted text, see RichEditBox.

Recommendations

- `Text` and `Value` make it easy to capture the value of a NumberBox as a String or as a Double without needing to convert the value between types. When programmatically altering the value of a NumberBox, it is recommended to do so through the `Value` property. `Value` will overwrite `Text` in initial set up. After the initial set up, changes to one will be propagated to the other, but consistently making programmatic changes through `Value` helps avoid any conceptual misunderstanding that NumberBox will accept non-numeric characters through `Text`.
- Use `Header` or `PlaceholderText` to inform users that NumberBox accepts only numerical characters as input. Spelled representation of numbers, such as "one", will not resolve to an accepted value.

Create a NumberBox

✔ **Important APIs:** NumberBox class

Open the WinUI 3 Gallery app and see NumberBox in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Here's the XAML for a basic NumberBox that demonstrates the default look. Use x:Bind to ensure the data displayed to the user remains in sync with the data stored in your app.

XAML

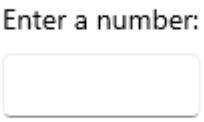
```
<NumberBox Value="{x:Bind Path=ViewModel.NumberBoxValue, Mode=TwoWay}" />
```

Labeling NumberBox

Use `Header` or `PlaceholderText` if the purpose of the NumberBox isn't clear. `Header` is visible whether or not the NumberBox has a value.

XAML

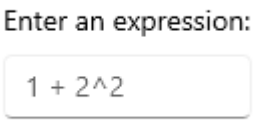
```
<NumberBox Header="Enter a number:"
    Value="{x:Bind Path=ViewModel.NumberBoxValue, Mode=TwoWay}" />
```



`PlaceholderText` is displayed inside the NumberBox and will only appear when `Value` is set to `NaN` or when the input is cleared by the user.

XAML

```
<NumberBox PlaceholderText="1+2^2"
    Value="{x:Bind Path=ViewModel.NumberBoxValue, Mode=TwoWay}" />
```



Enable calculation support

Setting the `AcceptsExpression` property to true enables NumberBox to evaluate basic inline expressions such as multiplication, division, addition, and subtraction using standard order of operations. Evaluation is triggered on loss of focus or when the user presses the "Enter" key. Once an expression is evaluated, the original form of the expression is not preserved.

XAML

```
<NumberBox Value="{x:Bind Path=ViewModel.NumberBoxValue, Mode=TwoWay}"
    AcceptsExpression="True" />
```

Increment and decrement stepping

Use the `SmallChange` property to configure how much the value inside a NumberBox is changed when the NumberBox is in focus and the user:

- scrolls
- presses the up arrow key
- presses the down arrow key

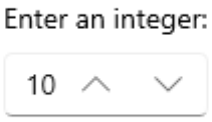
Use the `LargeChange` property to configure how much the value inside a NumberBox is changed when the NumberBox is in focus and the user press the PageUp or PageDown key.

Use the `SpinButtonPlacementMode` property to enable buttons that can be clicked to increment or decrement the value in the NumberBox by the amount specified by the `SmallChange` property. These buttons will be disabled if a Maximum or Minimum value would be surpassed with another step.

Set `SpinButtonPlacementMode` to `Inline` to enable the buttons to appear beside the control.

XAML

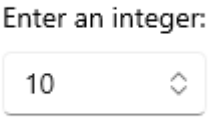
```
<NumberBox Value="{x:Bind Path=ViewModel.NumberBoxValue, Mode=TwoWay}"
    SmallChange="10"
    LargeChange="100"
    SpinButtonPlacementMode="Inline" />
```

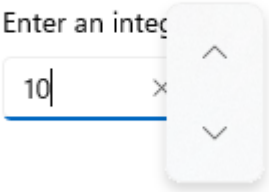


Set `SpinButtonPlacementMode` to `Compact` to enable the buttons to appear as a Flyout only when the NumberBox is in focus.

XAML

```
<NumberBox Value="{x:Bind Path=ViewModel.NumberBoxValue, Mode=TwoWay}"
    SmallChange="10"
    LargeChange="100"
    SpinButtonPlacementMode="Compact" />
```





Enabling input validation

Setting `ValidationMode` to `InvalidInputOverwritten` will enable NumberBox to overwrite invalid input that is not numerical nor legally formulaic with the last valid value when evaluation is triggered on loss of focus or a press of the "Enter" key.

XAML

```
<NumberBox Header="Quantity"
    Value="{x:Bind Path=ViewModel.NumberBoxValue, Mode=TwoWay}"
    ValidationMode="InvalidInputOverwritten" />
```

Setting `ValidationMode` to `Disabled` allows custom input validation to be configured.

With regard to decimal points and commas, the formatting used by a user will be replaced by the formatting configured for the NumberBox. An input validation error will not be triggered.

Formatting input

Number formatting can be used to format the value of a NumberBox by configuring an instance of a formatting class and assigning it to the `NumberFormatter` property. Decimal, currency, percent, and significant figures are few of the number formatting classes available. Note that rounding is also defined by number formatting properties.

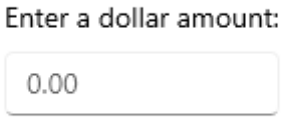
Here is an example of using DecimalFormatter to format a NumberBox's value to have one integer digit, two fraction digits, and round up to the nearest 0.25:

XAML

```
<NumberBox  x:Name="FormattedNumberBox"
    Value="{x:Bind Path=ViewModel.NumberBoxValue, Mode=TwoWay}" />
```

C#

```
private void SetNumberBoxNumberFormatter()
{
    IncrementNumberRounder rounder = new IncrementNumberRounder();
    rounder.Increment = 0.25;
    rounder.RoundingAlgorithm = RoundingAlgorithm.RoundUp;

    DecimalFormatter formatter = new DecimalFormatter();
    formatter.IntegerDigits = 1;
    formatter.FractionDigits = 2;
    formatter.NumberRounder = rounder;
    FormattedNumberBox.NumberFormatter = formatter;
}
```



With regard to decimal points and commas, the formatting used by a user will be replaced by the formatting configured for the NumberBox. An input validation error will not be triggered.

Remarks

Input Scope

`Number` will be used for the input scope. This input scope is intended for working with digits 0-9. This may be overwritten but alternative InputScope types will not be explicitly supported.

Not a Number

When a NumberBox is cleared of input, `Value` will be set to `NaN` to indicate no numerical value is present.

Expression evaluation

NumberBox uses infix notation to evaluate expressions. In order of precedence, the allowable operators are:

- ^
- */
- +-

Note that parentheses can be used to override precedence rules.

UWP and WinUI 2

> **ⓘ Important**

The information and examples in this article are optimized for apps that use the **Windows App SDK** and **WinUI 3**, but are generally applicable to UWP apps that use **WinUI 2**. See the UWP API reference for platform specific information and examples.

This section contains information you need to use the control in a UWP or WinUI 2 app.

The NumberBox for UWP apps requires WinUI 2. For more info, including installation instructions, see WinUI 2. APIs for this control exist in the Microsoft.UI.Xaml.Controls namespace.

- ✓ **WinUI 2 Apis:** NumberBox class
- ✓ Open the WinUI 2 Gallery app and see NumberBox in action. The **WinUI 2 Gallery** app includes interactive examples of most WinUI 2 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub.

To use the code in this article with WinUI 2, use an alias in XAML (we use `muxc`) to represent the Windows UI Library APIs that are included in your project. See Get Started with WinUI 2 for more info.

XAML

```
xmlns:muxc="using:Microsoft.UI.Xaml.Controls"

<muxc:NumberBox />
```

Labels

Article • 06/09/2022

A label is the name or title of a control or a group of related controls.

> **Important APIs**: Header property, TextBlock class

In XAML, many controls have a built-in Header property that you use to display the label. For controls that don't have a Header property, or to label groups of controls, you can use a TextBlock instead.

Recommendations

- Use a label to indicate to the user what they should enter into an adjacent control. You can also label a group of related controls, or display instructional text near a group of related controls.
- When labeling controls, write the label as a noun or a concise noun phrase, not as a sentence, and not as instructional text. Avoid colons or other punctuation.
- When you do have instructional text in a label, you can be more generous with text-string length and also use punctuation.

Get the sample code

- WinUI Gallery sample

Related topics

- Text controls
- TextBox.Header property

- PasswordBox.Header property
- ToggleSwitch.Header property
- DatePicker.Header property
- TimePicker.Header property
- Slider.Header property
- ComboBox.Header property
- RichEditBox.Header property
- TextBlock class

Content links in text controls

Article • 05/03/2022

Content links provide a way to embed rich data in your text controls, which lets a user find and use more information about a person or place without leaving the context of your app.

> **ⓘ Important**
>
> The Windows features that enable content links are not available in versions of Windows after Windows 10 version 1903. Content links for XAML text controls will not function in versions of Windows later than version 1903.

When the user prefixes an entry with the at (@) symbol in a RichEditBox, they're shown a list of people and/or place suggestions that matches the entry. Then, for example, when the user picks a place, a ContentLink for that place is inserted into the text. When the user invokes the content link from the RichEditBox, a flyout is shown with a map and additional info about the place.

> **Important APIs**: ContentLink class, ContentLinkInfo class, RichEditTextRange class

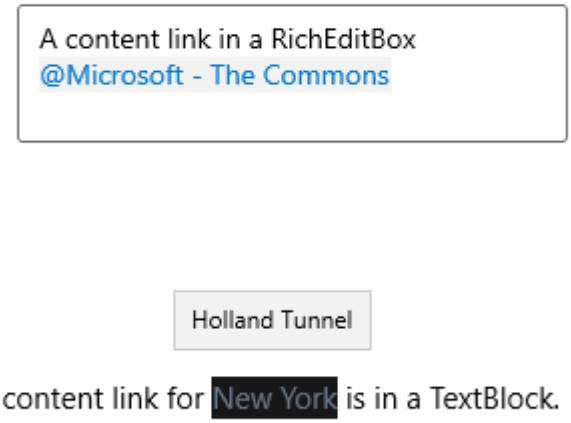
> **ⓘ Note**
>
> The APIs for content links are spread across the following namespaces: Windows.UI.Xaml.Controls, Windows.UI.Xaml.Documents, and Windows.UI.Text.

Content links in rich edit vs. text block controls

There are two distinct ways to use content links:

1. In a RichEditBox, the user can open a picker to add a content link by prefixing text with an @ symbol. The content link is stored as part the rich text content.
2. In a TextBlock or RichTextBlock, the content link is a text element with usage and behavior much like a Hyperlink.

Here's how content links look by default in a RichEditBox and in a TextBlock.



Differences in usage, rendering, and behavior are covered in detail in the following sections. This table gives a quick comparison of the main differences between a content link in a RichEditBox and a text block.

Expand table

Feature	RichEditBox	text block
Usage	ContentLinkInfo instance	ContentLink text element
Cursor	Determined by type of content link, can't be changed	Determined by Cursor property, **null** by default
ToolTip	Not rendered	Shows secondary text

Enable content links in a RichEditBox

The most common use of a content link is to let a user quickly add information by prefixing a person or place name with an ampersand (@) symbol in their text. When enabled in a RichEditBox, this opens a picker and lets the user insert a person from their contact list, or a nearby place, depending on what you've enabled.

The content link can be saved with the rich text content, and you can extract it to use in other parts of your app. For example, in an email app, you might extract the person info and use it to populate the To box with an email address.

> **Note**
>
> The content link picker is an app that's part of Windows, so it runs in a separate process from your app.

Content link providers

You enable content links in a RichEditBox by adding one or more content link providers to the RichEditBox.ContentLinkProviders collection. There are 2 content link providers built into the XAML framework.

- ContactContentLinkProvider – looks up contacts using the **People** app.
- PlaceContentLinkProvider – looks up places using the **Maps** app.

> ⓘ **Important**
>
> The default value for the RichEditBox.ContentLinkProviders property is **null**, not an empty collection. You need to explicitly create the **ContentLinkProviderCollection** before you add content link providers.

Here's how to add the content link providers in XAML.

XAML

```
<RichEditBox>
    <RichEditBox.ContentLinkProviders>
        <ContentLinkProviderCollection>
            <ContactContentLinkProvider/>
            <PlaceContentLinkProvider/>
        </ContentLinkProviderCollection>
    </RichEditBox.ContentLinkProviders>
</RichEditBox>
```

You can also add content link providers in a Style and apply it to multiple RichEditBoxes like this.

XAML

```
<Page.Resources>
    <Style TargetType="RichEditBox" x:Key="ContentLinkStyle">
        <Setter Property="ContentLinkProviders">
            <Setter.Value>
                <ContentLinkProviderCollection>
                    <PlaceContentLinkProvider/>
                    <ContactContentLinkProvider/>
                </ContentLinkProviderCollection>
            </Setter.Value>
        </Setter>
    </Style>
</Page.Resources>

<RichEditBox x:Name="RichEditBox01" Style="{StaticResource
```

```
ContentLinkStyle}" />
<RichEditBox x:Name="RichEditBox02" Style="{StaticResource
ContentLinkStyle}" />
```

Here's how to add content link providers in code.

```
C#

RichEditBox editor = new RichEditBox();
editor.ContentLinkProviders = new ContentLinkProviderCollection
{
    new ContactContentLinkProvider(),
    new PlaceContentLinkProvider()
};
```

Content link colors

The appearance of a content link is determined by its foreground, background, and icon. In a RichEditBox, you can set the ContentLinkForegroundColor and ContentLinkBackgroundColor properties to change the default colors.

You can't set the cursor. The cursor is rendered by the RichEditbox based on the type of content link - a Person cursor for a person link, or a Pin cursor for a place link.

The ContentLinkInfo object

When the user makes a selection from the people or places picker, the system creates a ContentLinkInfo object and adds it to the **ContentLinkInfo** property of the current RichEditTextRange.

The ContentLinkInfo object contains the information used to display, invoke, and manage the content link.

- **DisplayText** – This is the string that is shown when the content link is rendered. In a RichEditBox, the user can edit the text of a content link after it's created, which alters the value of this property.
- **SecondaryText** – This string is shown in the tooltip of a rendered content link.
  - In a Place content link created by the picker, it contains the address of the location, if available.
- **Uri** – The link to more information about the subject of the content link. This Uri can open an installed app or a website.
- **Id** - This is a read-only, per control, counter created by the RichEditBox control. It's used to track this ContentLinkInfo during actions such as delete or edit. If the

ContentLinkInfo is cut and paste back into the control, it will get a new Id. Id values are incremental.
- **LinkContentKind** – A string that describes the type of the content link. The built-in content types are *Places* and *Contacts*. The value is case sensitive.

Link content kind

There are several situations where the LinkContentKind is important.

- When a user copies a content link from a RichEditBox and pastes it into another RichEditBox, both controls must have a ContentLinkProvider for that content type. If not, the link is pasted as text.
- You can use LinkContentKind in a ContentLinkChanged event handler to determine what to do with a content link when you use it in other parts of your app. See the Example section for more info.
- The LinkContentKind influences how the system opens the Uri when the link is invoked. We'll see this in the discussion of Uri launching next.

Uri launching

The Uri property works much like the NavigateUri property of a Hyperlink. When a user clicks it, it launches the Uri in the default browser, or in the app that's registered for the particular protocol specified in the Uri value.

The specific behavior for the 2 built in kinds of link content are described here.

Places

The Places picker creates a ContentLinkInfo with a Uri root of https://maps.windows.com/. This link can be opened in 3 ways:

- If LinkContentKind = "Places", it opens an info card in a flyout. The flyout is similar to the content link picker flyout. It's part of Windows, and runs in a separate process from your app.
- If LinkContentKind is not "Places", it attempts to open the **Maps** app to the specified location. For example, this can happen if you've modified the LinkContentKind in the ContentLinkChanged event handler.
- If the Uri can't be opened in the Maps app, the map is opened in the default browser. This typically happens when the user's *Apps for websites* settings don't allow opening the Uri with the **Maps** app.

People

The People picker creates a ContentLinkInfo with a Uri that uses the **ms-people** protocol.

- If LinkContentKind = "People", it opens an info card in a flyout. The flyout is similar to the content link picker flyout. It's part of Windows, and runs in a separate process from your app.
- If LinkContentKind is not "People", it opens the **People** app. For example, this can happen if you've modified the LinkContentKind in the ContentLinkChanged event handler.

> **Tip**
>
> For more info about opening other apps and websites from your app, see the topics under **Launch an app with a Uri**.

Invoked

When the user invokes a content link, the ContentLinkInvoked event is raised before the default behavior of launching the Uri happens. You can handle this event to override or cancel the default behavior.

This example shows how you can override the default launching behavior for a Place content link. Instead of opening the map in a Place info card, Maps app, or default web browser, you mark the event as Handled and open the map in an in-app WebView control.

XAML

```
<RichEditBox x:Name="editor"
             ContentLinkInvoked="editor_ContentLinkInvoked">
    <RichEditBox.ContentLinkProviders>
        <ContentLinkProviderCollection>
            <PlaceContentLinkProvider/>
        </ContentLinkProviderCollection>
    </RichEditBox.ContentLinkProviders>
</RichEditBox>

<WebView x:Name="webView1"/>
```

C#

```
private void editor_ContentLinkInvoked(RichEditBox sender,
ContentLinkInvokedEventArgs args)
{
    if (args.ContentLinkInfo.LinkContentKind == "Places")
    {
        args.Handled = true;
        webView1.Navigate(args.ContentLinkInfo.Uri);
    }
}
```

ContentLinkChanged

You can use the ContentLinkChanged event to be notified when a content link is added, modified, or removed. This lets you extract the content link from the text and use it in other places in your app. This is shown later in the Examples section.

The properties of the ContentLinkChangedEventArgs are:

- **ChangedKind** - This ContentLinkChangeKind enum is the action within the ContentLink. For example, if the ContentLink is inserted, removed, or edited.
- **Info** - The ContentLinkInfo that was the target of the action.

This event is raised for each ContentLinkInfo action. For example, if the user copies and pastes several content links into the RichEditBox at once, this event is raised for each added item. Or if the user selects and deletes several content links at the same time, this event is raised for each deleted item.

This event is raised on the RichEditBox after the **TextChanging** event and before the **TextChanged** event.

Enumerate content links in a RichEditBox

You can use the RichEditTextRange.ContentLink property to get a content link from a rich edit document. The TextRangeUnit enumeration has the value ContentLink to specify the content link as a unit to use when navigating a text range.

This example shows how you can enumerate all the content links in a RichEditBox, and extract the people into a list.

XAML

```
<StackPanel Width="300">
    <RichEditBox x:Name="Editor" Height="200">
        <RichEditBox.ContentLinkProviders>
            <ContentLinkProviderCollection>
```

```
                <ContactContentLinkProvider/>
                <PlaceContentLinkProvider/>
            </ContentLinkProviderCollection>
        </RichEditBox.ContentLinkProviders>
    </RichEditBox>

    <Button Content="Get people" Click="Button_Click"/>

    <ListView x:Name="PeopleList" Header="People">
        <ListView.ItemTemplate>
            <DataTemplate x:DataType="ContentLinkInfo">
                <TextBlock>
                    <ContentLink Info="{x:Bind}" Background="Transparent"/>
                </TextBlock>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</StackPanel>
```

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    PeopleList.Items.Clear();
    RichEditTextRange textRange = Editor.Document.GetRange(0, 0) as
RichEditTextRange;

    do
    {
        // The Expand method expands the Range EndPosition
        // until it finds a ContentLink range.
        textRange.Expand(TextRangeUnit.ContentLink);
        if (textRange.ContentLinkInfo != null
            && textRange.ContentLinkInfo.LinkContentKind == "People")
        {
            PeopleList.Items.Add(textRange.ContentLinkInfo);
        }
    } while (textRange.MoveStart(TextRangeUnit.ContentLink, 1) > 0);
}
```

ContentLink text element

To use a content link in a TextBlock or RichTextBlock control, you use the ContentLink text element (from the Windows.UI.Xaml.Documents namespace).

Typical sources for a ContentLink in a text block are:

- A content link created by a picker that you extracted from a RichTextBlock control.
- A content link for a place that you create in your code.

For other situations, a Hyperlink text element is usually appropriate.

ContentLink appearance

The appearance of a content link is determined by its foreground, background, and cursor. In a text block, you can set the Foreground (from TextElement) and Background properties to change the default colors.

By default, the Hand cursor is shown when the user hovers over the content link. Unlike RichEditBlock, text block controls don't change the cursor automatically based on the link type. You can set the Cursor property to change the cursor based on link type or other factors.

Here's an example of a ContentLink used in a TextBlock. The ContentLinkInfo is created in code and assigned to the Info property of the ContentLink text element that's created in XAML.

XAML

```
<StackPanel>
    <TextBlock>
        <Span xml:space="preserve">
            <Run>This volcano erupted in 1980: </Run><ContentLink
x:Name="placeContentLink" Cursor="Pin"/>
            <LineBreak/>
        </Span>
    </TextBlock>

    <Button Content="Show" Click="Button_Click"/>
</StackPanel>
```

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var info = new Windows.UI.Text.ContentLinkInfo();
    info.DisplayText = "Mount St. Helens";
    info.SecondaryText = "Washington State";
    info.LinkContentKind = "Places";
    info.Uri = new Uri("https://maps.windows.com?cp=46.1912~-122.1944");
    placeContentLink.Info = info;
}
```

Tip

When you use a ContentLink in a text control with other text elements in XAML, place the content in a **Span** container and apply the `xml:space="preserve"` attribute to the Span to keep the white space between the ContentLink and other elements.

Examples

In this example, a user can enter a person or place content link into a RickTextBlock. You handle the ContentLinkChanged event to extract the content links and keep them up-to-date in either a people list or places list.

In the item templates for the lists, you use a TextBlock with a ContentLink text element to show the content link info. The ListView provides its own background for each item, so the ContentLink background is set to Transparent.

XAML

```
<StackPanel Orientation="Horizontal" Grid.Row="1">
    <RichEditBox x:Name="Editor"
                 ContentLinkChanged="Editor_ContentLinkChanged"
                 Margin="20" Width="300" Height="200"
                 VerticalAlignment="Top">
        <RichEditBox.ContentLinkProviders>
            <ContentLinkProviderCollection>
                <ContactContentLinkProvider/>
                <PlaceContentLinkProvider/>
            </ContentLinkProviderCollection>
        </RichEditBox.ContentLinkProviders>
    </RichEditBox>

    <ListView x:Name="PeopleList" Header="People">
        <ListView.ItemTemplate>
            <DataTemplate x:DataType="ContentLinkInfo">
                <TextBlock>
                    <ContentLink Info="{x:Bind}"
                                 Background="Transparent"
                                 Cursor="Person"/>
                </TextBlock>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>

    <ListView x:Name="PlacesList" Header="Places">
        <ListView.ItemTemplate>
            <DataTemplate x:DataType="ContentLinkInfo">
                <TextBlock>
                    <ContentLink Info="{x:Bind}"
                                 Background="Transparent"
                                 Cursor="Pin"/>
                </TextBlock>
```

```
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</StackPanel>
```

C#

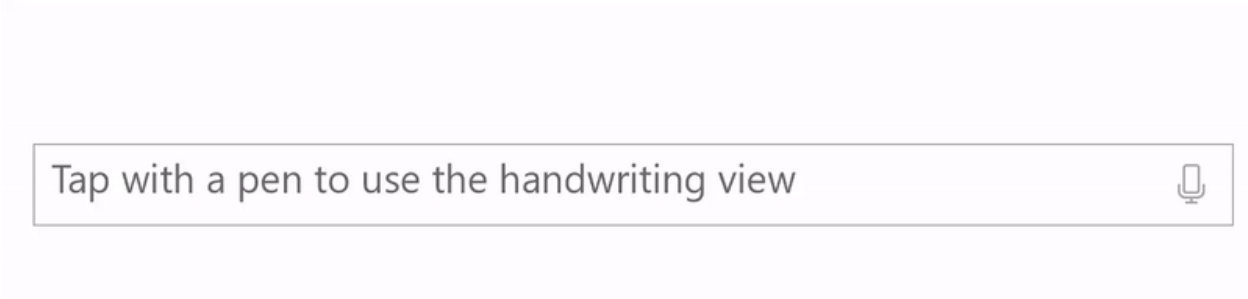
```
private void Editor_ContentLinkChanged(RichEditBox sender,
ContentLinkChangedEventArgs args)
{
    var info = args.ContentLinkInfo;
    var change = args.ChangeKind;
    ListViewBase list = null;

    // Determine whether to update the people or places list.
    if (info.LinkContentKind == "People")
    {
        list = PeopleList;
    }
    else if (info.LinkContentKind == "Places")
    {
        list = PlacesList;
    }

    // Determine whether to add, delete, or edit.
    if (change == ContentLinkChangeKind.Inserted)
    {
        Add();
    }
    else if (args.ChangeKind == ContentLinkChangeKind.Removed)
    {
        Remove();
    }
    else if (change == ContentLinkChangeKind.Edited)
    {
        Remove();
        Add();
    }

    // Add content link info to the list. It's bound to the
    // Info property of a ContentLink in XAML.
    void Add()
    {
        list.Items.Add(info);
    }

    // Use ContentLinkInfo.Id to find the item,
    // then remove it from the list using its index.
    void Remove()
    {
        var items = list.Items.Where(i => ((ContentLinkInfo)i).Id ==
info.Id);
        var item = items.FirstOrDefault();
```

```
        var idx = list.Items.IndexOf(item);

        list.Items.RemoveAt(idx);
    }
}
```

Text input with the handwriting view

Article • 03/24/2023

> **Note**
>
> Handwriting view is not supported by text controls in the **Windows App SDK**. This article applies to UWP apps only.



Customize the handwriting view (for ink to text input) that is built into UWP text input controls such as the TextBox, RichEditBox, and AutoSuggestBox.

Overview

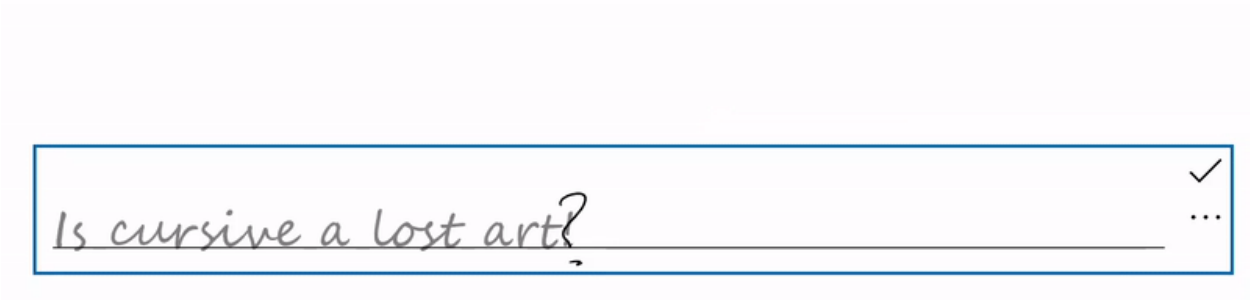
UWP text input controls support pen input using Windows Ink by transforming into a handwriting surface when a user taps into a text input box using a pen.

Text is recognized as the user writes anywhere in the handwriting surface while a candidate window shows the recognition results. The user can tap a result to choose it, or continue writing to accept the proposed candidate. The literal (letter-by-letter) recognition results are included in the candidate window, so recognition is not restricted to words in a dictionary. As the user writes, the accepted text input is converted to a script font that retains the feel of natural writing.

> **Note**
>
> The handwriting view is enabled by default, but you can disable it on a per-control basis and revert to the text input panel instead.

A user can edit their text using standard gestures and actions:

- *strike through* or *scratch out* - draw through to delete a word or part of a word
- *join* - draw an arc between words to delete the space between them
- *insert* - draw a caret symbol to insert a space
- *overwrite* - write over existing text to replace it



Disable the handwriting view

The built-in handwriting view is enabled by default.

You might want to disable the handwriting view if you already provide equivalent ink-to-text functionality in your application, or your text input experience relies on some kind of formatting or special character (such as a tab) not available through handwriting.

In this example, we disable the handwriting view by setting the IsHandwritingViewEnabled property of the TextBox control to false. All text controls that support the handwriting view support a similar property.

XAML

```
<TextBox Name="SampleTextBox"
    Height="50" Width="500"
    FontSize="36" FontFamily="Segoe UI"
    PlaceholderText="Try taping with your pen"
    IsHandwritingViewEnabled="False">
</TextBox>
```

Specify the alignment of the handwriting view

The handwriting view is located above the underlying text control and sized to accommodate the user's handwriting preferences (see **Settings -> Bluetooth & devices -> Pen & Windows Ink -> Handwriting -> Font size**). The view is also automatically aligned relative to the text control and its location within the app.

The application UI does not reflow to accommodate the larger control, which might occlude important UI.

The following snippet shows how to use the PlacementAlignment property of a TextBox HandwritingView to specify which anchor on the underlying text control is used to align the handwriting view.

XAML

```
<TextBox Name="SampleTextBox"
    Height="50" Width="500"
    FontSize="36" FontFamily="Segoe UI"
    PlaceholderText="Try taping with your pen">
        <TextBox.HandwritingView>
            <HandwritingView PlacementAlignment="TopLeft"/>
        </TextBox.HandwritingView>
</TextBox>
```

Disable auto-completion candidates

The text suggestion popup is enabled by default. It provides a list of top ink recognition candidates from which the user can select in case the primary candidate is incorrect.

If your application already provides robust, custom recognition functionality, you can use the AreCandidatesEnabled property to disable the built-in suggestions, as shown in the following example.

XAML

```
<TextBox Name="SampleTextBox"
    Height="50" Width="500"
    FontSize="36" FontFamily="Segoe UI"
    PlaceholderText="Try taping with your pen">
        <TextBox.HandwritingView>
            <HandwritingView AreCandidatesEnabled="False"/>
        </TextBox.HandwritingView>
</TextBox>
```

Use handwriting font preferences

A user can choose from a pre-defined collection of handwriting-based fonts to use when rendering text based on ink recognition (see **Settings -> Bluetooth & devices -> Pen & Windows Ink -> Handwriting -> Font**).

Your app can access this setting and use the selected font for the recognized text in the text control.

In this example, we listen for the TextChanged event of a TextBox and apply the user's selected font if the text change originated from the HandwritingView (or a default font, if not).

C#

```
private void SampleTextBox_TextChanged(object sender, TextChangedEventArgs
e)
{
    ((TextBox)sender).FontFamily =
        ((TextBox)sender).HandwritingView.IsOpen ?
            new FontFamily(PenAndInkSettings.GetDefault().FontFamilyName) :
            new FontFamily("Segoe UI");
}
```

Access the HandwritingView in composite controls

Composite controls that use either the TextBox or RichEditBox control (such as AutoSuggestBox), also support a HandwritingView.

To access the HandwritingView in a composite control, use the VisualTreeHelper API.

The following XAML snippet displays an AutoSuggestBox control.

XAML

```
<AutoSuggestBox Name="SampleAutoSuggestBox"
    Height="50" Width="500"
    PlaceholderText="Auto Suggest Example"
    FontSize="16" FontFamily="Segoe UI"
    Loaded="SampleAutoSuggestBox_Loaded">
</AutoSuggestBox>
```

In the corresponding code-behind, we show how to disable the HandwritingView on the AutoSuggestBox.

1. First, we handle the element's Loaded event and call a `FindInnerTextBox` function to start the visual tree traversal.

```
C#

private void SampleAutoSuggestBox_Loaded(object sender, RoutedEventArgs
e)
{
    if (FindInnerTextBox((AutoSuggestBox)sender))
        autoSuggestInnerTextBox.IsHandwritingViewEnabled = false;
}
```

2. In the `FindInnerTextBox` function, we iterate through the visual tree (starting at an AutoSuggestBox) by calling a `FindVisualChildByName` function.

```
csharp

private bool FindInnerTextBox(AutoSuggestBox autoSuggestBox)
{
    if (autoSuggestInnerTextBox == null)
    {
        // Cache textbox to avoid multiple tree traversals.
        autoSuggestInnerTextBox =
            (TextBox)FindVisualChildByName<TextBox>(autoSuggestBox);
    }
    return (autoSuggestInnerTextBox != null);
}
```

3. Finally, the `FindVisualChildByName` function iterates through the visual tree until the TextBox is retrieved.

```
C#

private FrameworkElement FindVisualChildByName<T>(DependencyObject obj)
{
    FrameworkElement element = null;
    int childrenCount =
        VisualTreeHelper.GetChildrenCount(obj);
    for (int i = 0; (i < childrenCount) && (element == null); i++)
    {
        FrameworkElement child =
            (FrameworkElement)VisualTreeHelper.GetChild(obj, i);
        if ((child.GetType()).Equals(typeof(T)) ||
(child.GetType().GetTypeInfo().IsSubclassOf(typeof(T))))
        {
            element = child;
        }
```

```
            else
            {
                element = FindVisualChildByName<T>(child);
            }
        }
        return (element);
    }
```

Reposition the HandwritingView

In some cases, you might need to ensure that the HandwritingView covers UI elements that it otherwise might not.

Here, we create a TextBox that supports dictation (implemented by placing a TextBox and a dictation button into a StackPanel).

As the StackPanel is now larger than the TextBox, the HandwritingView might not occlude all of the composite control.

To address this, set the PlacementTarget property of the HandwritingView to the UI element to which it should be aligned.

XAML

```
<StackPanel Name="DictationBox"
    Orientation="Horizontal"
    VerticalAlignment="Top"
    HorizontalAlignment="Left"
    BorderThickness="1" BorderBrush="DarkGray"
    Height="55" Width="500" Margin="50">
    <TextBox Name="DictationTextBox"
        Width="450" BorderThickness="0"
        FontSize="24" VerticalAlignment="Center">
        <TextBox.HandwritingView>
            <HandwritingView PlacementTarget="{Binding
ElementName=DictationBox}"/>
        </TextBox.HandwritingView>
    </TextBox>
```

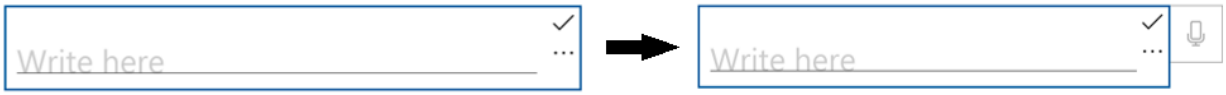
```
    <Button Name="DictationButton"
        Height="48" Width="48"
        FontSize="24"
        FontFamily="Segoe MDL2 Assets"
        Content="&#xE720;"
        Background="White" Foreground="DarkGray"
Tapped="DictationButton_Tapped" />
</StackPanel>
```

Resize the HandwritingView

You can also set the size of the HandwritingView, which can be useful when you need to ensure the view doesn't occlude important UI.

Like the previous example, we create a TextBox that supports dictation (implemented by placing a TextBox and a dictation button into a StackPanel).

In this case, we resize the HandwritingView to ensure that the dictation button is visible.



To do this, we bind the MaxWidth property of the HandwritingView to the width of the UI element that it should occlude.

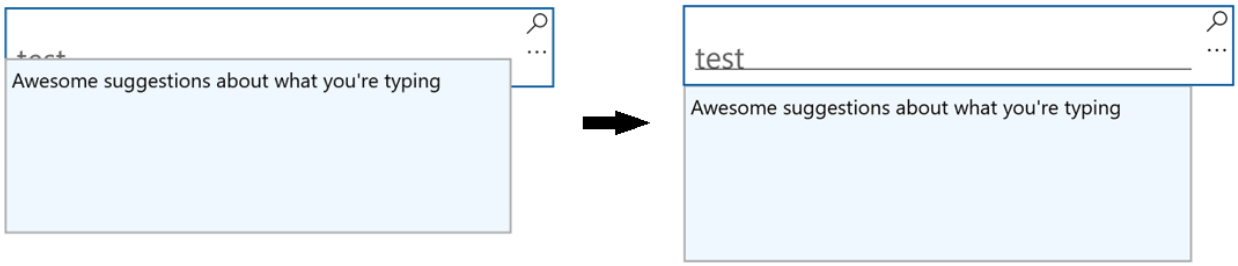
XAML

```
<StackPanel Name="DictationBox"
    Orientation="Horizontal"
    VerticalAlignment="Top"
    HorizontalAlignment="Left"
    BorderThickness="1"
    BorderBrush="DarkGray"
    Height="55" Width="500"
    Margin="50">
    <TextBox Name="DictationTextBox"
        Width="450"
        BorderThickness="0"
        FontSize="24"
        VerticalAlignment="Center">
        <TextBox.HandwritingView>
            <HandwritingView
                PlacementTarget="{Binding ElementName=DictationBox}"
```

```
                MaxWidth="{Binding ElementName=DictationTextBox,
Path=Width"/>
        </TextBox.HandwritingView>
    </TextBox>
    <Button Name="DictationButton"
        Height="48" Width="48"
        FontSize="24"
        FontFamily="Segoe MDL2 Assets"
        Content="&#xE720;"
        Background="White" Foreground="DarkGray"
        Tapped="DictationButton_Tapped" />
</StackPanel>
```

Reposition custom UI

If you have custom UI that appears in response to text input, such as an informational popup, you might need to reposition that UI so it doesn't occlude the handwriting view.



The following example shows how to listen for the Opened, Closed, and SizeChanged events of the HandwritingView to set the position of a Popup.

C#

```
private void Search_HandwritingViewOpened(
    HandwritingView sender, HandwritingPanelOpenedEventArgs args)
{
    UpdatePopupPositionForHandwritingView();
}

private void Search_HandwritingViewClosed(
    HandwritingView sender, HandwritingPanelClosedEventArgs args)
{
    UpdatePopupPositionForHandwritingView();
}

private void Search_HandwritingViewSizeChanged(
    object sender, SizeChangedEventArgs e)
{
    UpdatePopupPositionForHandwritingView();
}

private void UpdatePopupPositionForHandwritingView()
```

```
{
if (CustomSuggestionUI.IsOpen)
    CustomSuggestionUI.VerticalOffset = GetPopupVerticalOffset();
}

private double GetPopupVerticalOffset()
{
    if (SearchTextBox.HandwritingView.IsOpen)
        return (SearchTextBox.Margin.Top +
SearchTextBox.HandwritingView.ActualHeight);
    else
        return (SearchTextBox.Margin.Top + SearchTextBox.ActualHeight);
}
```

Re-template the HandwritingView control

As with all XAML framework controls, you can customize both the visual structure and visual behavior of a HandwritingView for your specific requirements.

To see a full example of creating a custom template check out the Create custom transport controls how-to or the Custom Edit Control sample.

Title bar

Article • 03/31/2025

The TitleBar control provides a simplified way to create a custom title bar for your app. The title bar is a fundamental component of a Windows app's user interface that identifies the app via its icon and title, houses the system caption buttons that let a user close, maximize, minimize, and restore the window, and lets a user drag the window around the screen.

You can use a custom title bar to better integrate the title bar area with your app UI. The title bar can be customized to match the app's visual style using Mica theming. It can include other relevant information, such as a document title or the current state (e.g., “Editing,” “Viewing,” etc.). It can also host other WinUI controls, like AutoSuggestBox and PersonPicture, providing a cohesive user experience for your app.

Is this the right control?

Use the TitleBar control when you want to integrate the title bar area with your app UI using customizations such as subtitles, Mica theming, and integrations with WinUI controls.

Anatomy

By default, the title bar shows only the system caption buttons. Other parts of the title bar are shown or hidden depending on associated property settings.

The title bar is divided into these areas:

- **Back button:** IsBackButtonEnabled, IsBackButtonVisible, BackRequested - A built-in back button for navigation.
- **Pane toggle button:** IsPaneToggleButtonVisible, PaneToggleRequested - This button is intended to be used in conjunction with the NavigationView control.
- **Left header:** LeftHeader
- **Icon:** IconSource
- **Title:** Title
- **Subtitle:** Subtitle
- **Content:** Content
- **Right header:** RightHeader
- **Min drag region:** This area is reserved next to the system caption buttons so that the user always has a place to grab the window for dragging.
- **System caption buttons:** These buttons are not part of the TitleBar control - it simply allocates space where the caption buttons appear, depending on RTL or LTR settings. Caption buttons and customizations are handled by the AppWindowTitleBar.

The layout is reversed when the FlowDirection is **RightToLeft**.

Create a title bar

✔ **Important APIs:** TitleBar class, Title property

Open the WinUI 3 Gallery app and see the TitleBar in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

This example creates a simple title bar that replaces the system title bar. It has a title, icon, and Mica theming.

XAML

```
<Window
    ...  >
    <Window.SystemBackdrop>
        <MicaBackdrop Kind="Base"/>
    </Window.SystemBackdrop>

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
```

```
        </Grid.RowDefinitions>

        <TitleBar x:Name="SimpleTitleBar"
                  Title="My App">
            <TitleBar.IconSource>
                <FontIconSource Glyph="&#xF4AA;"/>
            </TitleBar.IconSource>
        </TitleBar>

        <!-- App content -->
        <Frame x:Name="RootFrame" Grid.Row="1"/>
    </Grid>
</Window>
```

C#

```
public MainWindow()
{
    this.InitializeComponent();

    // Hides the default system title bar.
    ExtendsContentIntoTitleBar = true;
    // Replace system title bar with the WinUI TitleBar control.
    SetTitleBar(SimpleTitleBar);
}
```

Integration with NavigationView

The Navigation view has a built-in back button and pane toggle button. Fluent Design guidance recommends that these controls be placed in the title bar when a custom title bar is used.

This example demonstrates how to integrate the TitleBar control with a NavigationView control by hiding the back button and pane toggle button in the navigation view and using the corresponding buttons on the title bar instead.

XAML

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <TitleBar Title="My App"
              IsBackButtonVisible="True"
              IsBackButtonEnabled="{x:Bind RootFrame.CanGoBack,
Mode=OneWay}"
```

```
                BackRequested="TitleBar_BackRequested"
                IsPaneToggleButtonVisible="True"
                PaneToggleRequested="TitleBar_PaneToggleRequested">
    </TitleBar>

    <NavigationView x:Name="RootNavigationView" Grid.Row="1"
                    IsBackButtonVisible="Collapsed"
                    IsPaneToggleButtonVisible="False">
        <Frame x:Name="RootFrame" />
    </NavigationView>
</Grid>
```

C#

```
private void TitleBar_BackRequested(TitleBar sender, object args)
{
    if (RootFrame.CanGoBack)
    {
        RootFrame.GoBack();
    }
}

private void TitleBar_PaneToggleRequested(TitleBar sender, object args)
{
    RootNavigationView.IsPaneOpen = !RootNavigationView.IsPaneOpen;
}
```

UWP and WinUI 2

The TitleBar control is not available for UWP and WinUI 2. Instead, see Title bar customization (UWP apps).

Related articles

- Title bar (design)
- Title bar customization
- TitleBar class

Shell

Article • 08/08/2024

Learn how to use tiles, badges, toasts, and notifications to provide entry points into your app and keep users up-to-date.

Toast notifications

Adaptive and interactive toast notifications let you create flexible notifications with text, images, and buttons/inputs.

Badge notifications

A notification badge conveys summary or status information specific to your app.

Push notifications

Push notifications let third-party developers send toast, tile, badge, and raw updates from their own cloud service.

Taskbar

Ask your user to pin your app to the taskbar to make it easier to find.

Title bar

Extend your app content into the title bar and add custom content.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

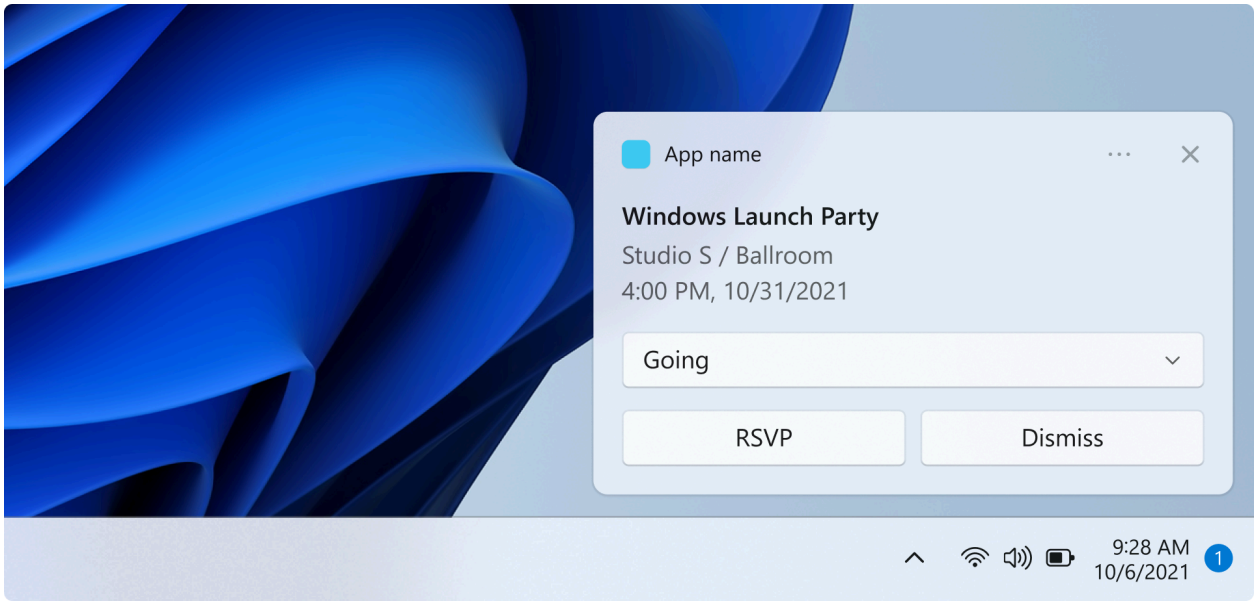
Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Toast notifications

Article • 11/26/2024



This section provides design and implementation guidance for toast notifications, which are messages that your app can construct and deliver to your user while they are not currently inside your app.

Toast notification UX and implementation guidance

- Toast notification UX guidance
- Send a local toast notification from C# apps
- Send a local toast notification from C++ UWP apps
- Send a local toast notification from Win32 C++ WRL apps
- Send a local toast notification from other types of unpackaged apps
- Toast content
- Toast content schema
- Schedule a toast notification

Additional toast notifications features

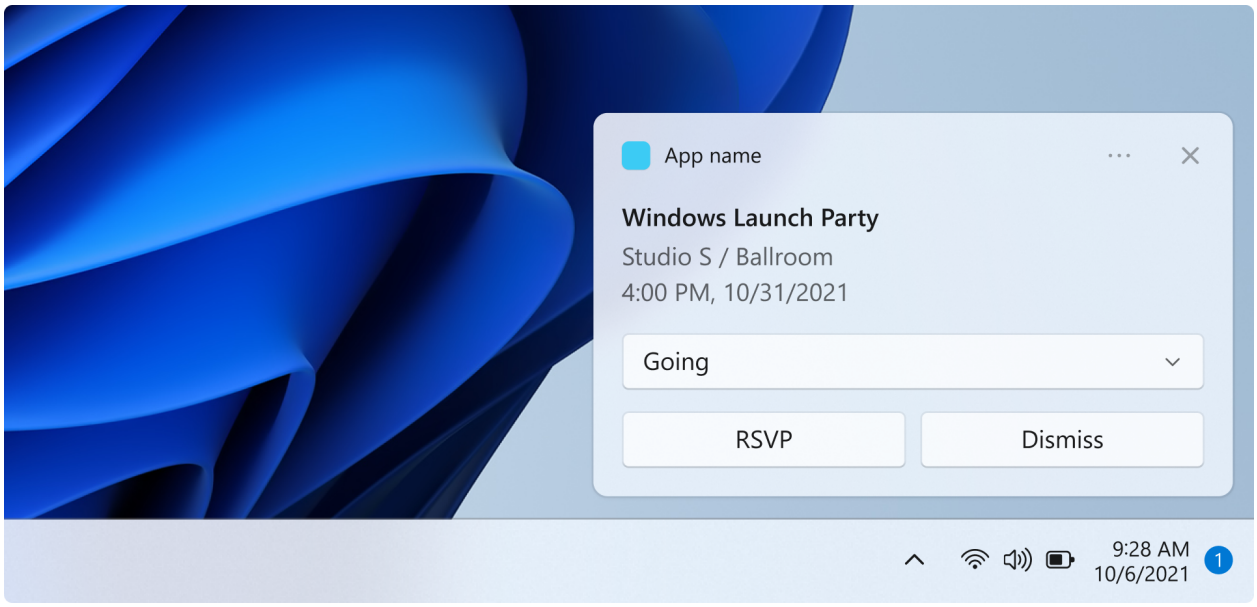
- Custom audio on toasts
- Toast progress bar and data binding
- Toast with pending update activation
- Custom timestamps on toasts
- Grouping toast notifications with collections

- Toast headers
- Notification listener: Access all notifications

Notifications design basics

Article • 06/03/2022

Drive user engagement with your app by providing personalized, actionable, and useful toast notifications. Toast notifications are the Windows notifications that appear in the lower right of the user's screen and the Notification Center (called Action Center in Windows 10). This article provides design guidance to consider as you plan and implement the toast notification experience for your app.

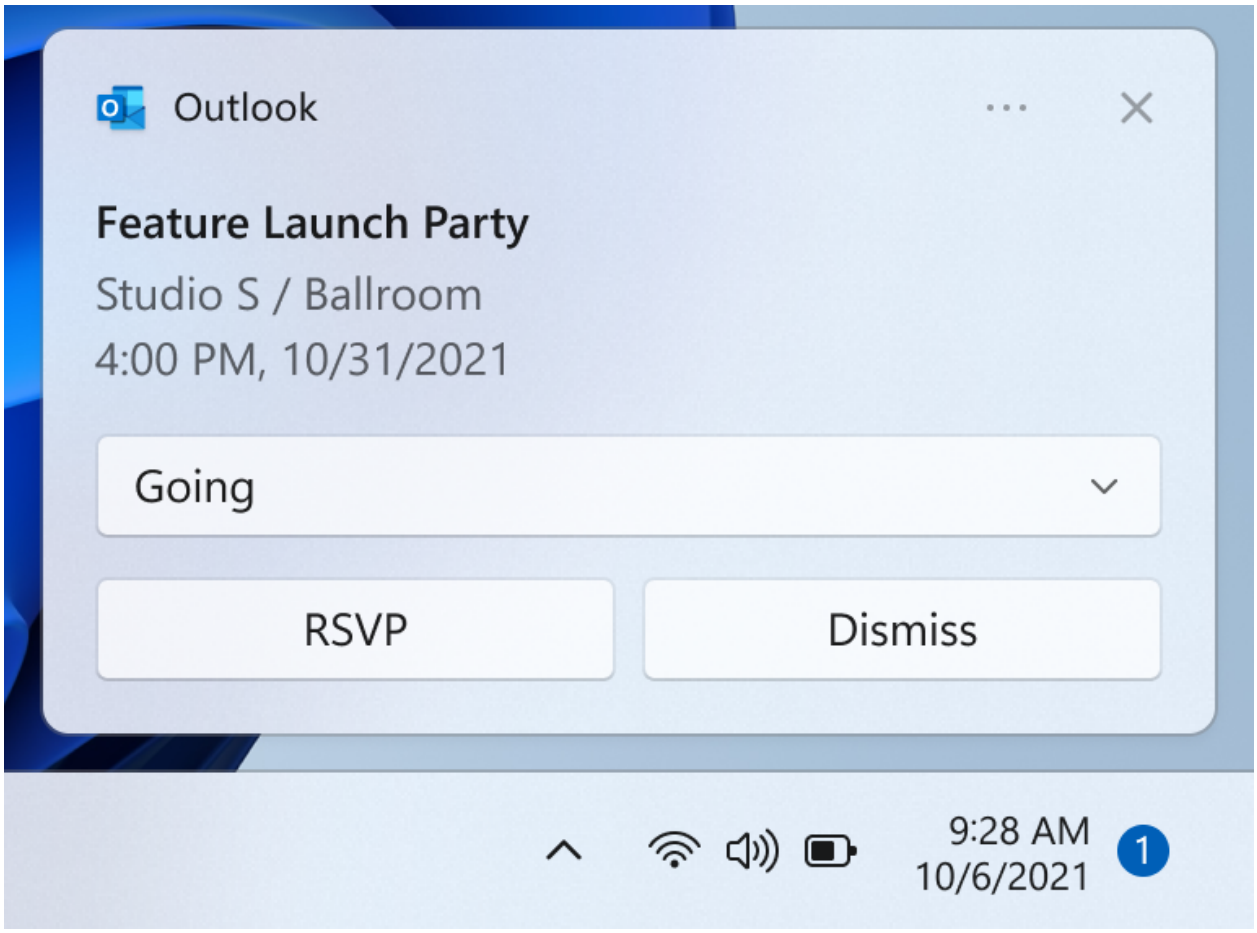


Notifications should be informative and valuable

Design your notifications to have clear intent so the user can quickly understand the purpose of the notification and stay in their flow. Consider the user scenarios your toast notifications are enabling and add elements to your notification that are appropriate for each scenario. Notifications should allow the user to save time, but shouldn't distract or create more work.

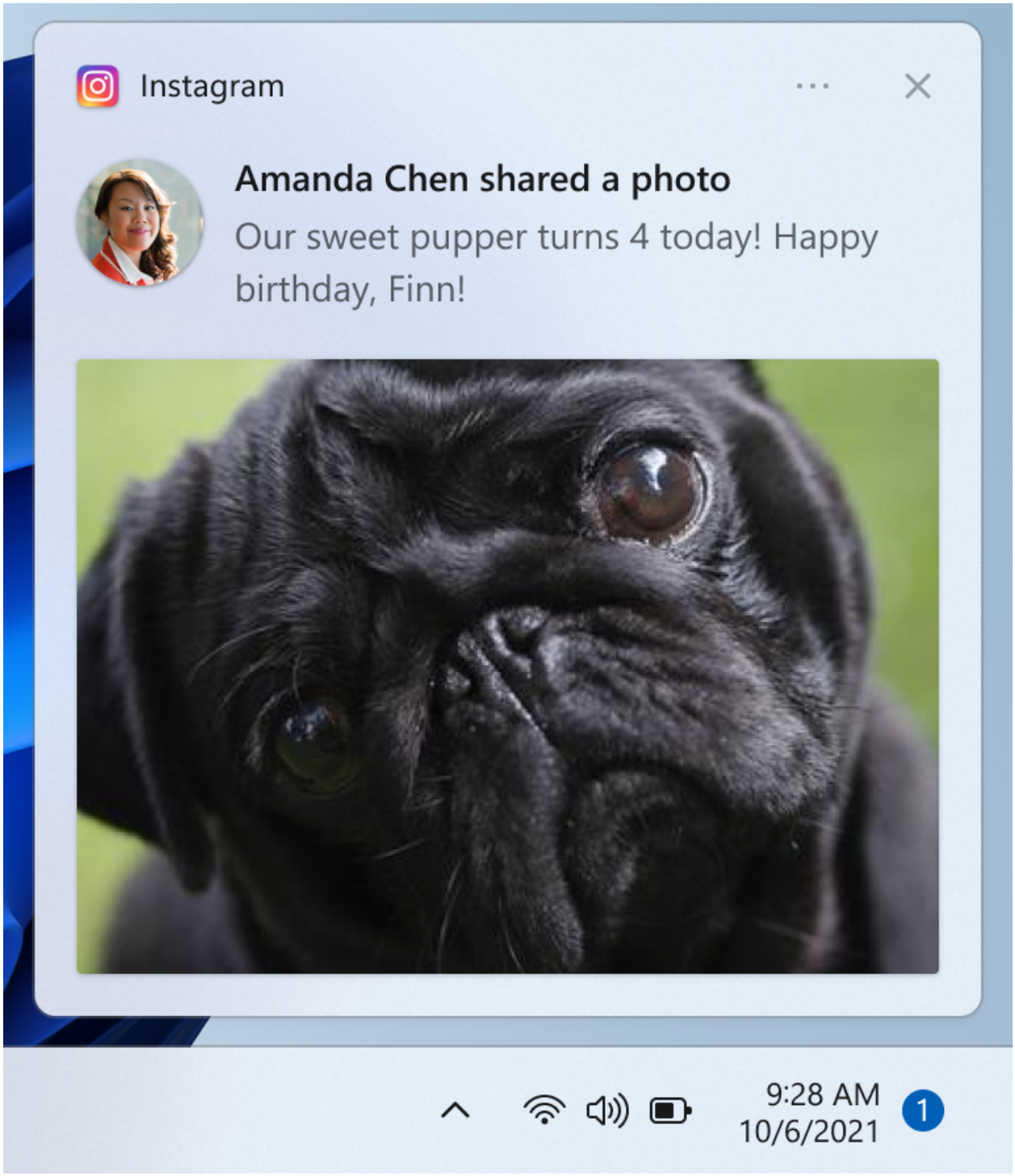
Choose the right notification UI elements

Consider which UI elements offer the most value to your user scenario. Is the purpose of the notification to allow the user to accomplish a task, or is it simply to get new information? The following section shows some example elements that you can use in your toast notifications.



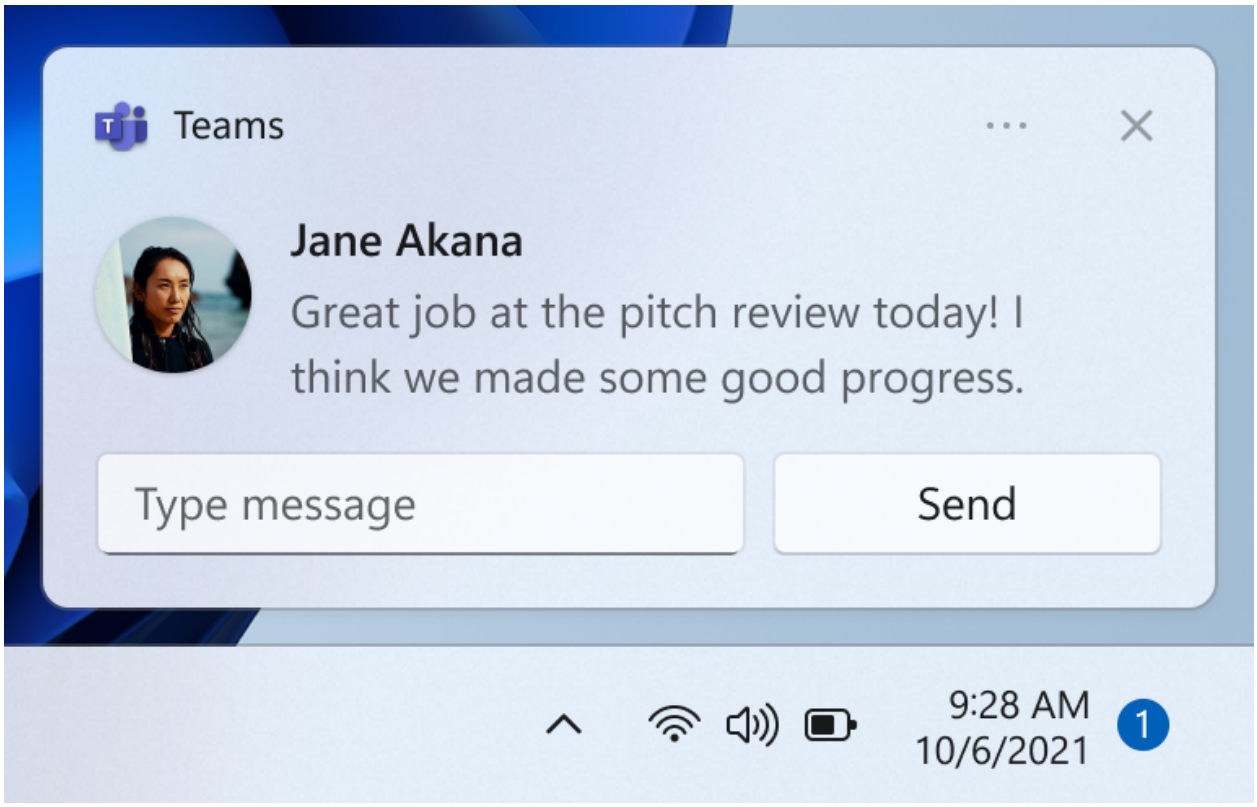
Buttons

Buttons trigger an immediate action. By adding buttons, the user can complete a task within the notification.



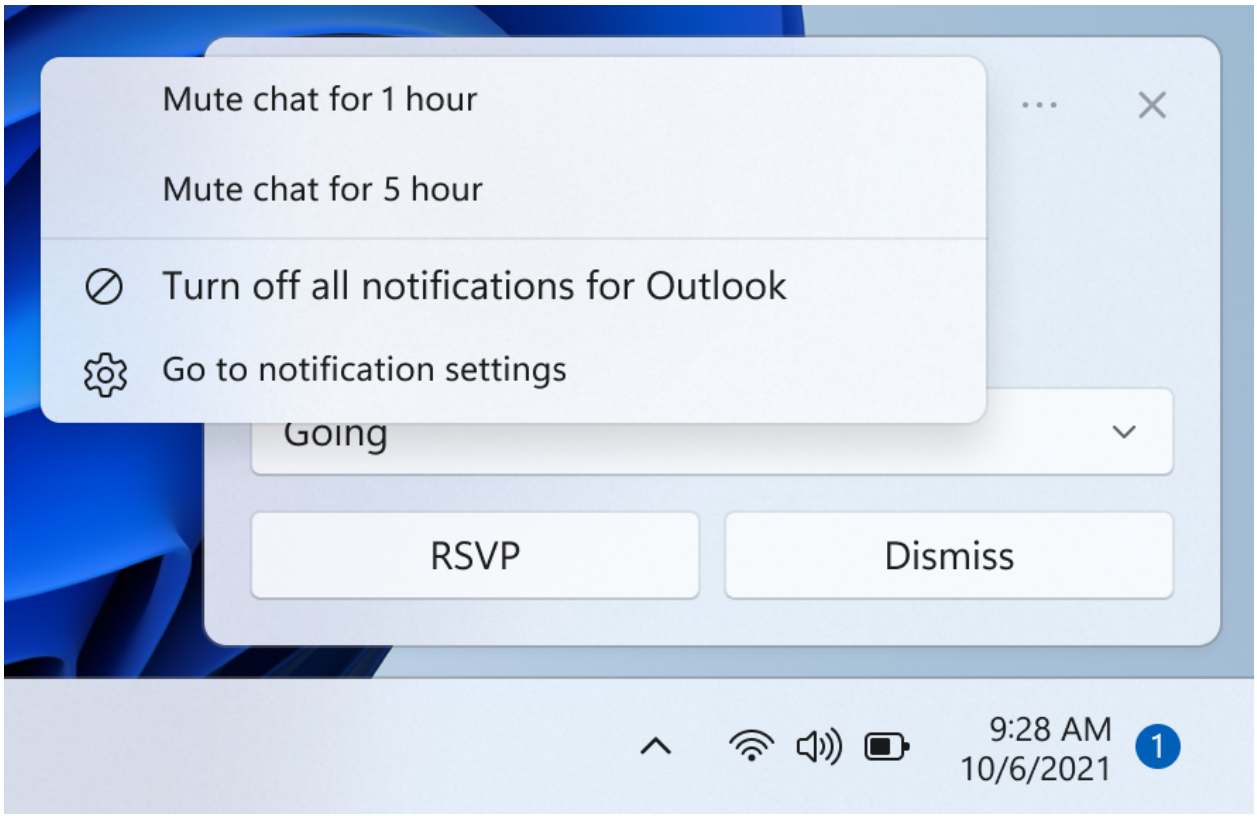
Images

Images can either be app logo override, inline, or hero. By adding images, the user can parse your content more quickly. Examples include adding an app logo override image depicting the sender's profile image to a chat or email notification.



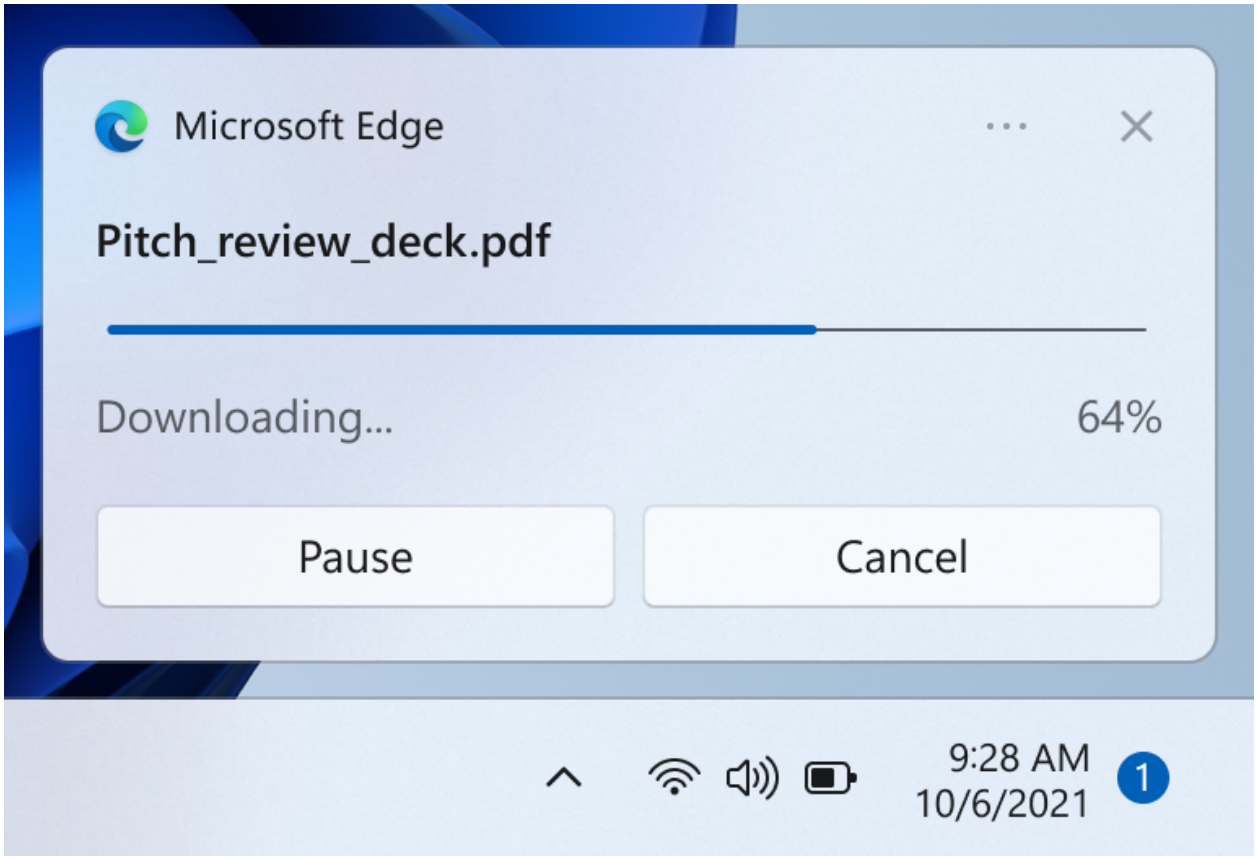
Quick replies

Quick reply boxes enable users to respond to messages without opening the app. Examples include adding quick reply boxes to a chat notification.



Context menu buttons

Context menu buttons give users more options on right-click on the notification or upon selection of the context menu icon. An example scenario is adding an option to mute a group chat for an hour.



Progress bars

Progress bars communicate status of long-running operations. By adding a progress bar, the user doesn't have to check back in your app for updates. Examples include communicating a file download or game update with a progress bar notification.

To see all of the UI elements that can be used in a notification, see Toast content.

Notifications should not be noisy

Users can easily be overloaded with too much information and get frustrated if they are being interrupted while they are trying to focus. Too many interruptions leads to users turning off this critical communication channel for your app.

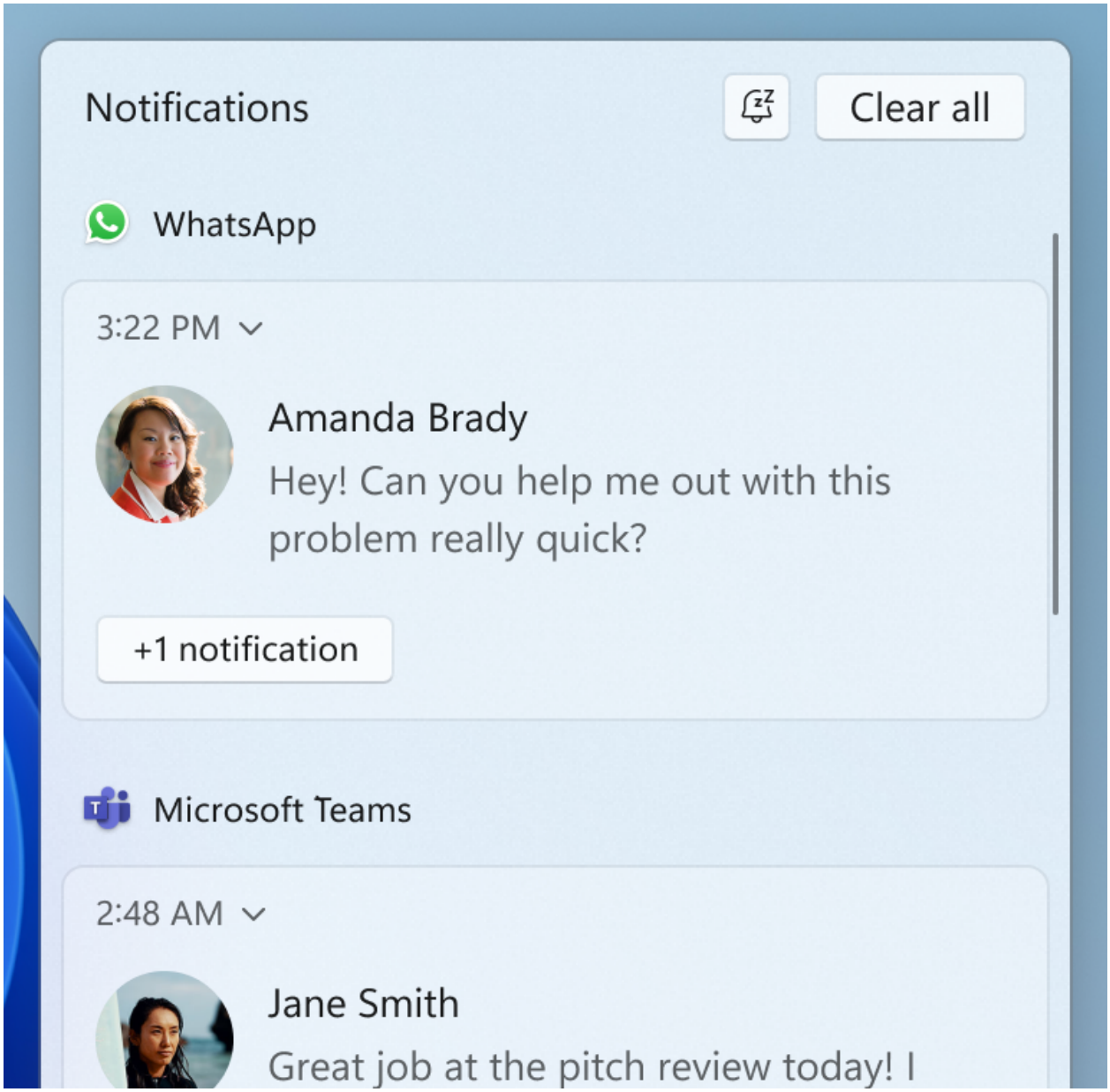
- One method of not interrupting the user but still communicating valuable information is by suppressing the notification, meaning the notification is automatically placed directly into the Notification Center without an alert.
- On Windows 11, Focus Sessions allow users to suppress extraneous notifications. You can use the FocusSessionManager API to detect if the user is in a Focus Session and adjust your app's notification behavior accordingly.

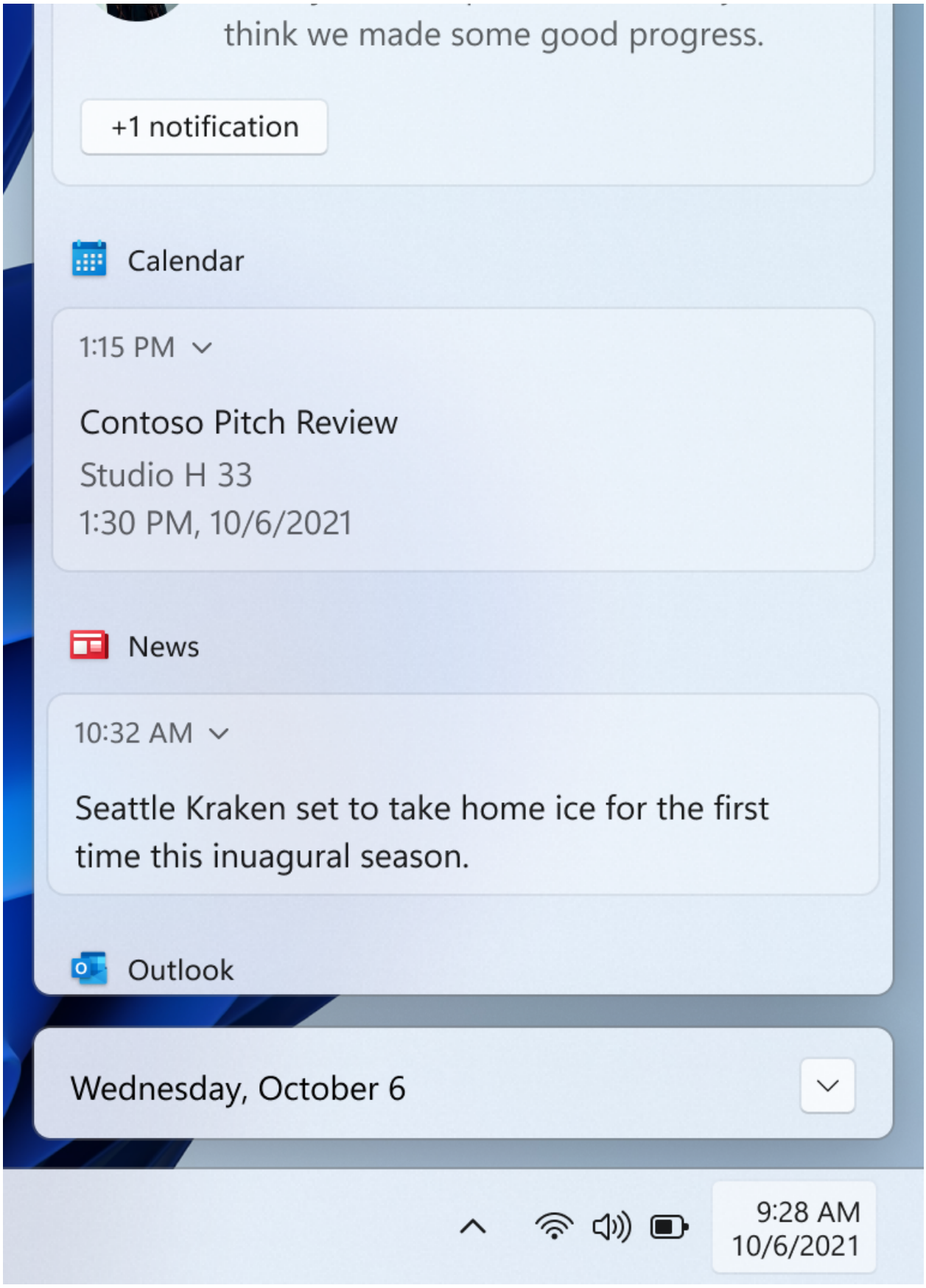
Respond to the user's intent

Just like the design of your notification UI, the behavior when the user clicks on the notification or on UI elements within the notification should be deliberate and consistent, and should engage the user in a clear task or scenario.

- If the user clicks on your notification to launch it, your app should launch in the notification's context.
- If the user clicks on a button in your notification, your app should launch in the context that is appropriate for the button's action
  - However, if the action is intended to be a background task, such as quick reply, your app should not launch.

Provide a consistent Notification Center experience





The Notification Center allows the user to view and respond to notifications after they have been displayed, or in the case of silent notifications, after they have been received but not displayed. It's important to keep Notification Center tidy by clearing out old notifications. You should also be sure to provide a consistent experience when removing notifications so the user feels confident they aren't missing information. Following

standard conventions for notification removal makes the notification experience predictable and intuitive for users.

The following are some example scenarios that illustrate strategies for clearing out old notifications from Notification Center:

- The user receives 10 new emails, and thus 10 new notifications. The user goes to the email app and views 10 emails, without interacting with the 10 notifications. After the user views the 10 emails, the email app removes the 10 notifications from the Notification Center. This also applies if the user has viewed the 10 emails elsewhere, such as their phone or website.
- The user receives 10 new chat messages from one user, and thus 10 new notifications. The user selects one of these chat notifications, which launches the chat. After the user views the chat, the app removes the 9 remaining notifications from the Notification Center.

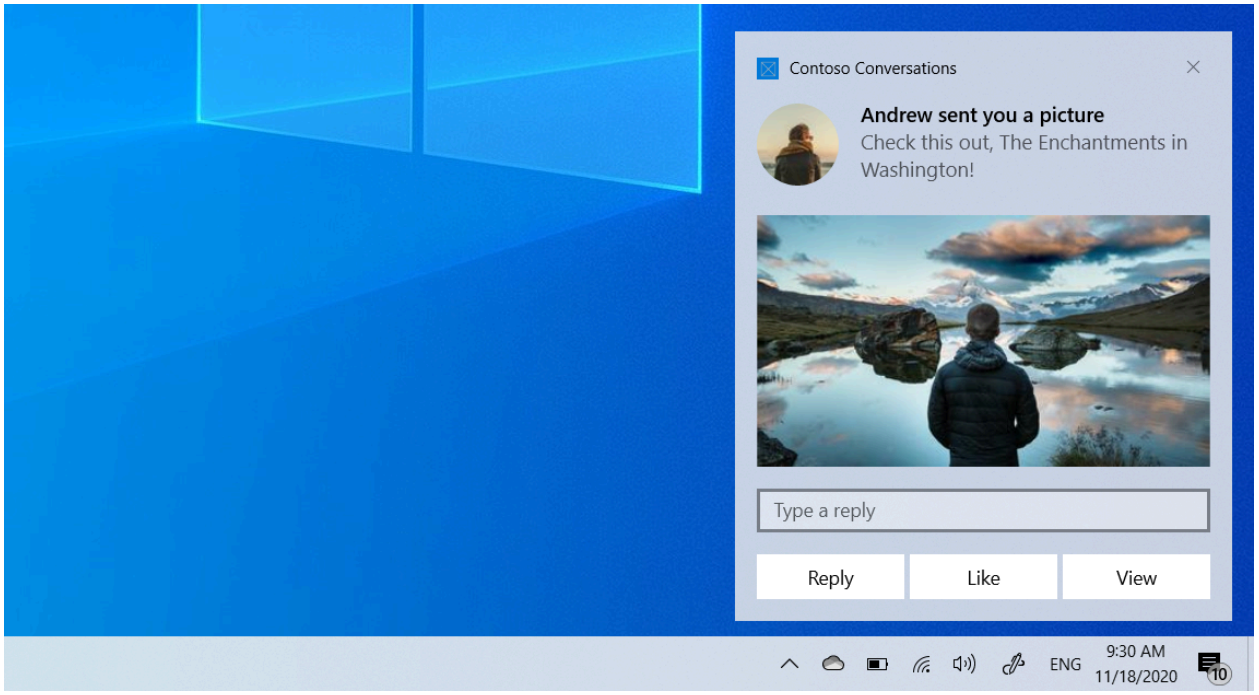
Related articles

Developer guidance for notifications

Send a local toast notification from a C# app

Article • 11/27/2024

A toast notification is a message that your app can construct and deliver to your user while they are not currently inside your app.



This quickstart walks you through the steps to create, deliver, and display a Windows 10 or Windows 11 toast notification using rich content and interactive actions. This quickstart uses local notifications, which are the simplest notification to implement. All types of apps (WPF, UWP, WinForms, console) can send notifications!

> **ⓘ Important**
>
> If you're writing a C++ app, please see the C++ UWP or C++ WRL documentation.

Step 1: Install NuGet package

Within your Visual Studio solution, right click your project, click "*Manage NuGet Packages...*" and search for and install the `Microsoft.Toolkit.Uwp.Notifications` NuGet package version 7.0 or greater.

> **ⓘ Important**

.NET Framework desktop apps that still use packages.config must migrate to PackageReference, otherwise the Windows SDKs won't be referenced correctly. In your project, right-click on "References", and click "Migrate packages.config to PackageReference".

.NET Core 3.0 WPF apps must update to .NET Core 3.1, otherwise the APIs will be absent.

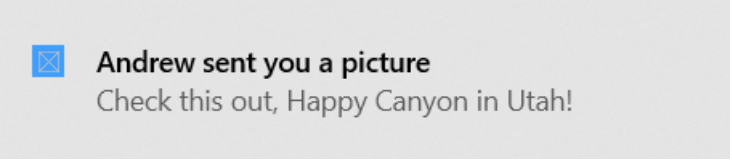
.NET apps must use one of the Windows TFMs, otherwise the toast sending and management APIs like `Show()` will be missing. Set your TFM to `net6.0-windows10.0.17763.0` or later.

Our code sample will use this package. This package allows you to create toast notifications without using XML, and also allows desktop apps to send toasts.

Step 2: Send a toast

In Windows 10 and Windows 11, your toast notification content is described using an adaptive language that allows great flexibility with how your notification looks. For more information, see the toast content documentation.

We'll start with a simple text-based notification. Construct the notification content (using the Notifications library), and show the notification! Note that the namespace is `Microsoft.Toolkit.Uwp.Notifications`.



C#

```
// Requires Microsoft.Toolkit.Uwp.Notifications NuGet package version 7.0 or greater
new ToastContentBuilder()
    .AddArgument("action", "viewConversation")
    .AddArgument("conversationId", 9813)
    .AddText("Andrew sent you a picture")
    .AddText("Check this out, The Enchantments in Washington!")
    .Show(); // Not seeing the Show() method? Make sure you have version 7.0, and if you're using .NET 6 (or later), then your TFM must be net6.0-windows10.0.17763.0 or greater
```

Try running this code and you should see the notification appear!

Step 3: Handling activation

After showing a notification, you likely need to handle the user clicking the notification (whether that means bringing up specific content after the user clicks it, opening your app in general, or performing an action when the user clicks the notification).

The steps for handling activation differ for UWP, and for packaged and unpackaged desktop apps.

UWP

When the user clicks your notification (or a button on the notification with foreground activation), your app's **App.xaml.cs OnActivated** will be invoked, and the arguments you added will be returned.

App.xaml.cs

C#

```
protected override void OnActivated(IActivatedEventArgs e)
{
    // Handle notification activation
    if (e is ToastNotificationActivatedEventArgs toastActivationArgs)
    {
        // Obtain the arguments from the notification
        ToastArguments args =
ToastArguments.Parse(toastActivationArgs.Argument);

        // Obtain any user input (text boxes, menu selections) from the
notification
        ValueSet userInput = toastActivationArgs.UserInput;

        // TODO: Show the corresponding content
    }
}
```

ⓘ Important

You must initialize your frame and activate your window just like your **OnLaunched** code. **OnLaunched is NOT called if the user clicks on your toast**, even if your app was closed and is launching for the first time. We often recommend combining **OnLaunched** and **OnActivated** into your own `OnLaunchedOrActivated` method since the same initialization needs to occur in both.

Step 4: Handling uninstallation

UWP

You don't need to do anything! When UWP apps are uninstalled, all notifications and any other related resources are automatically cleaned up.

Adding images

You can add rich content to notifications. We'll add an inline image and a profile (app logo override) image.

> **Note**
>
> Images can be used from the app's package, the app's local storage, or from the web. As of the Fall Creators Update, web images can be up to 3 MB on normal connections and 1 MB on metered connections. On devices not yet running the Fall Creators Update, web images must be no larger than 200 KB.

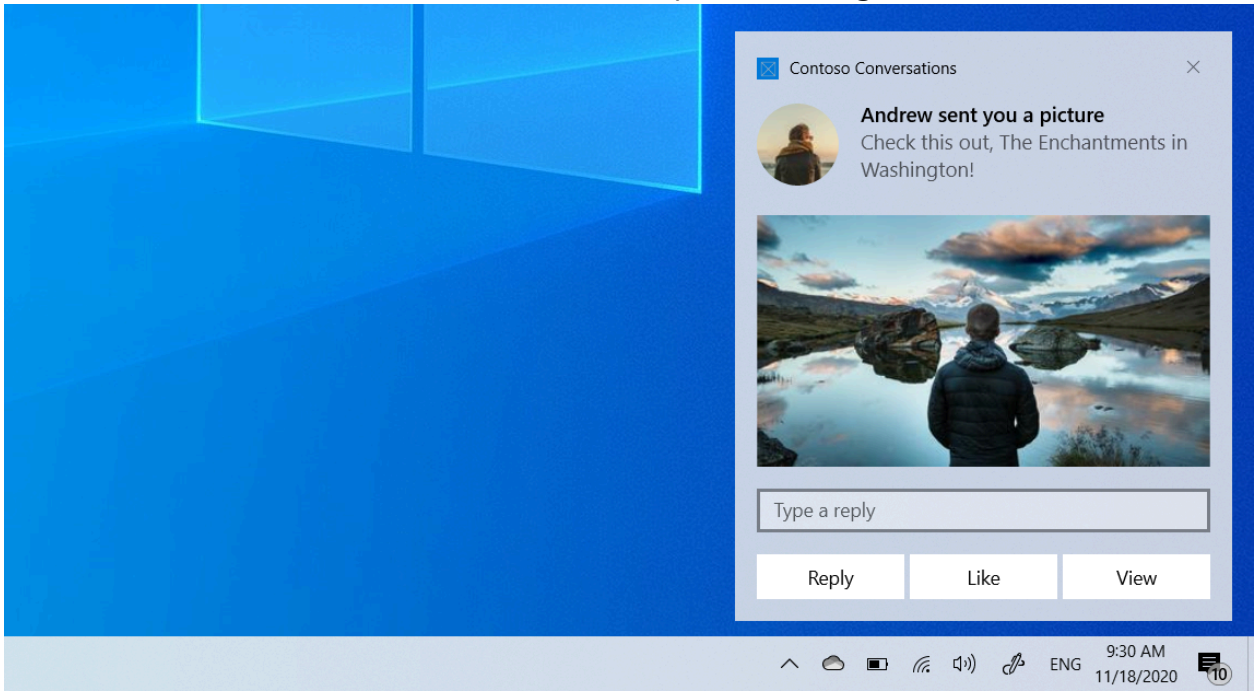
> **Important**
>
> Http images are supported only in packaged apps that have the internet capability in their manifest. Unpackaged apps don't support http images; you must download the image to your local app data, and reference it locally.

C#

```
// Construct the content and show the toast!
new ToastContentBuilder()
    ...

    // Inline image
    .AddInlineImage(new Uri("https://picsum.photos/360/202?image=883"))

    // Profile (app logo override) image
    .AddAppLogoOverride(new Uri("ms-appdata:///local/Andrew.jpg"),
ToastGenericAppLogoCrop.Circle)

    .Show();
```

Adding buttons and inputs

You can add buttons and inputs to make your notifications interactive. Buttons can launch your foreground app, a protocol, or your background task. We'll add a reply text box, a "Like" button, and a "View" button that opens the image.



C#

```
int conversationId = 384928;

// Construct the content
new ToastContentBuilder()
    .AddArgument("conversationId", conversationId)
    ...

    // Text box for replying
    .AddInputTextBox("tbReply", placeHolderContent: "Type a response")

    // Buttons
```

```
    .AddButton(new ToastButton()
        .SetContent("Reply")
        .AddArgument("action", "reply")
        .SetBackgroundActivation())

    .AddButton(new ToastButton()
        .SetContent("Like")
        .AddArgument("action", "like")
        .SetBackgroundActivation())

    .AddButton(new ToastButton()
        .SetContent("View")
        .AddArgument("action", "viewImage")
        .AddArgument("imageUrl", image.ToString()))

    .Show();
```

The activation of foreground buttons are handled in the same way as the main toast body (your App.xaml.cs OnActivated will be called).

Note that arguments added to the top-level toast (like conversation ID) will also be returned when the buttons are clicked, as long as buttons use the AddArgument API as seen above (if you custom assign arguments on a button, the top-level arguments won't be included).

Handling background activation

UWP

When you specify background activation on your toast (or on a button inside the toast), your background task will be executed instead of activating your foreground app.

For more information on background tasks, please see Support your app with background tasks.

If you are targeting build 14393 or later, you can use in-process background tasks, which greatly simplify things. Note that in-process background tasks will fail to run on older versions of Windows. We'll use an in-process background task in this code sample.

C#

```
const string taskName = "ToastBackgroundTask";

// If background task is already registered, do nothing
```

```
if (BackgroundTaskRegistration.AllTasks.Any(i =>
i.Value.Name.Equals(taskName)))
    return;

// Otherwise request access
BackgroundAccessStatus status = await
BackgroundExecutionManager.RequestAccessAsync();

// Create the background task
BackgroundTaskBuilder builder = new BackgroundTaskBuilder()
{
    Name = taskName
};

// Assign the toast action trigger
builder.SetTrigger(new ToastNotificationActionTrigger());

// And register the task
BackgroundTaskRegistration registration = builder.Register();
```

Then in your App.xaml.cs, override the OnBackgroundActivated method. You can then retrieve the pre-defined arguments and user input, similar to the foreground activation.

App.xaml.cs

C#

```
protected override async void
OnBackgroundActivated(BackgroundActivatedEventArgs args)
{
    var deferral = args.TaskInstance.GetDeferral();

    switch (args.TaskInstance.Task.Name)
    {
        case "ToastBackgroundTask":
            var details = args.TaskInstance.TriggerDetails as
ToastNotificationActionTriggerDetail;
            if (details != null)
            {
                ToastArguments arguments =
ToastArguments.Parse(details.Argument);
                var userInput = details.UserInput;

                // Perform tasks
            }
            break;
    }

    deferral.Complete();
}
```

Set an expiration time

In Windows 10, all toast notifications go in Action Center after they are dismissed or ignored by the user, so users can look at your notification after the popup is gone.

However, if the message in your notification is only relevant for a period of time, you should set an expiration time on the toast notification so the users do not see stale information from your app. For example, if a promotion is only valid for 12 hours, set the expiration time to 12 hours. In the code below, we set the expiration time to be 2 days.

> ⓘ **Note**
>
> The default and maximum expiration time for local toast notifications is 3 days.

C#

```
// Create toast content and show the toast!
new ToastContentBuilder()
    .AddText("Expires in 2 days...")
    .Show(toast =>
    {
        toast.ExpirationTime = DateTime.Now.AddDays(2);
    });
```

Provide a primary key for your toast

If you want to programmatically remove or replace the notification you send, you need to use the Tag property (and optionally the Group property) to provide a primary key for your notification. Then, you can use this primary key in the future to remove or replace the notification.

To see more details on replacing/removing already delivered toast notifications, please see Quickstart: Managing toast notifications in action center (XAML).

Tag and Group combined act as a composite primary key. Group is the more generic identifier, where you can assign groups like "wallPosts", "messages", "friendRequests", etc. And then Tag should uniquely identify the notification itself from within the group. By using a generic group, you can then remove all notifications from that group by using the RemoveGroup API.

C#

```
// Create toast content and show the toast!
new ToastContentBuilder()
    .AddText("New post on your wall!")
    .Show(toast =>
    {
        toast.Tag = "18365";
        toast.Group = "wallPosts";
    });
```

Clear your notifications

Apps are responsible for removing and clearing their own notifications. When your app is launched, we do NOT automatically clear your notifications.

Windows will only automatically remove a notification if the user explicitly clicks the notification.

Here's an example of what a messaging app should do...

1. User receives multiple toasts about new messages in a conversation
2. User taps one of those toasts to open the conversation
3. The app opens the conversation and then clears all toasts for that conversation (by using RemoveGroup on the app-supplied group for that conversation)
4. User's Action Center now properly reflects the notification state, since there are no stale notifications for that conversation left in Action Center.

To learn about clearing all notifications or removing specific notifications, see Quickstart: Managing toast notifications in action center (XAML).

C#

```
ToastNotificationManagerCompat.History.Clear();
```

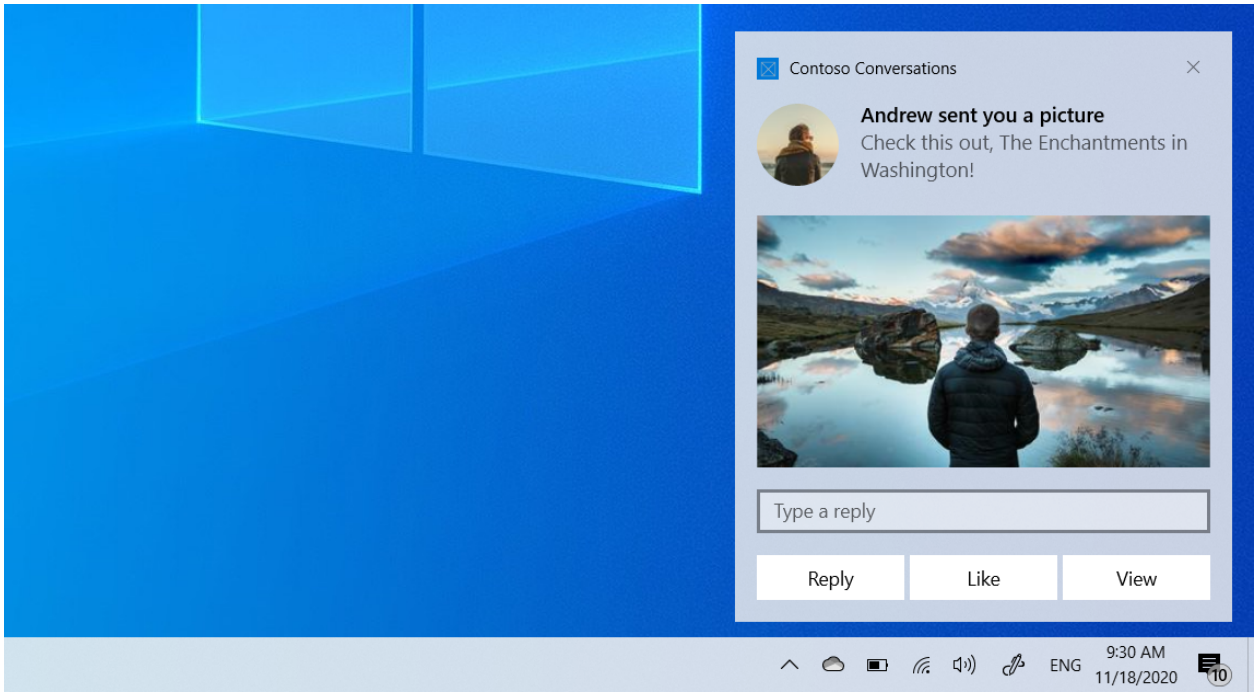
Resources

- Full code sample on GitHub
- Toast content documentation
- ToastNotification Class
- ToastNotificationActivatedEventArgs Class

Send a local toast notification from C++ UWP apps

Article • 10/27/2022

A toast notification is a message that your app can construct and deliver to your user while they are not currently inside your app.



This quickstart walks you through the steps to create, deliver, and display a Windows 10 or Windows 11 toast notification using rich content and interactive actions. This quickstart uses local notifications, which are the simplest notification to implement. All types of apps (WPF, UWP, WinForms, console) can send notifications!

> **ⓘ Important**
>
> If you're writing a C++ non-UWP app, please see the **C++ WRL** documentation. If you're writing a C# app, please see the **C# documentation**.

Step 1: Install NuGet package

You can create toast notifications with the Windows Community Toolkit (WCT)'s builder syntax OR with XML. If you prefer the latter, please skip to **Step 2** and refer to the **Without builder syntax** code examples.

Within your Visual Studio solution, right click your project, click "*Manage NuGet Packages...*" and search for and install the `Microsoft.Toolkit.Uwp.Notifications` NuGet

package version 7.0 or greater.

Our **Builder syntax** code examples will use this package. This package allows you to create toast notifications without using XML.

Step 2: Add namespace declarations

Builder syntax

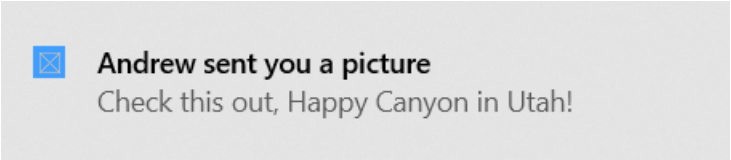
C++

```
using namespace Microsoft::Toolkit::Uwp::Notifications;
```

Step 3: Send a toast

In Windows 10 and Windows 11, your toast notification content is described using an adaptive language that allows great flexibility with how your notification looks. For more information, see the toast content documentation.

We'll start with a simple text-based notification. Construct the notification content (using the Notifications library), and show the notification! Note that the namespace is `Microsoft.Toolkit.Uwp.Notifications`.



If you are not using the WCT Notifications library builder syntax, you will instead construct the XML toast template, populate it with text and values, construct the notification, and show it.

Builder syntax

C++

```
// Construct the content and show the toast!
(ref new ToastContentBuilder())
    ->AddArgument("action", "viewConversation")
    ->AddArgument("conversationId", 9813)
    ->AddText("Andrew sent you a picture")
    ->AddText("Check this out, The Enchantments in Washington!")
    ->Show();
```

Step 4: Handling activation

When the user clicks your notification (or a button on the notification with foreground activation), your app's **App.xaml.cpp OnActivated** will be invoked.

App.xaml.cpp

C++

```
void App::OnActivated(IActivatedEventArgs^ e)
{
    // Handle notification activation
    if (e->Kind == ActivationKind::ToastNotification)
    {
        ToastNotificationActivatedEventArgs^ toastActivationArgs =
(ToastNotificationActivatedEventArgs^)e;

        // Obtain the arguments from the notification
        ToastArguments^ args = ToastArguments::Parse(toastActivationArgs-
>Argument);

        // Obtain any user input (text boxes, menu selections) from the
notification
        auto userInput = toastActivationArgs->UserInput;

        // TODO: Show the corresponding content
    }
}
```

> **ⓘ Important**
>
> You must initialize your frame and activate your window just like your **OnLaunched** code. **OnLaunched is NOT called if the user clicks on your toast**, even if your app was closed and is launching for the first time. We often recommend combining **OnLaunched** and **OnActivated** into your own `OnLaunchedOrActivated` method since the same initialization needs to occur in both.

Activation in depth

The first step in making your notifications actionable is to add some launch args to your notification, so that your app can know what to launch when the user clicks the notification (in this case, we're including some information that later tells us we should open a conversation, and we know which specific conversation to open).

Builder syntax

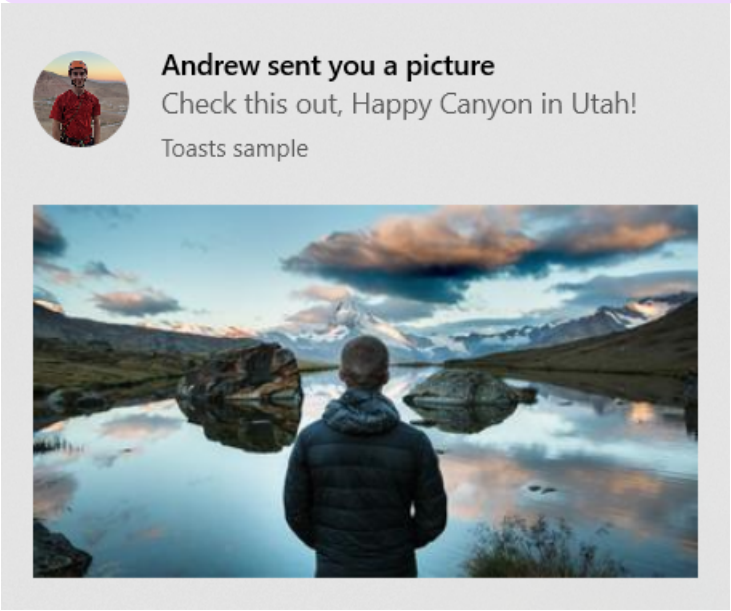
C++

```
// Construct the content and show the toast!
(ref new ToastContentBuilder())

    // Arguments returned when user taps body of notification
    ->AddArgument("action", "viewConversation")
    ->AddArgument("conversationId", 9813)

    ->AddText("Andrew sent you a picture")
    ->Show();
```

Adding images

You can add rich content to notifications. We'll add an inline image and a profile (app logo override) image.

> **Note**
>
> Images can be used from the app's package, the app's local storage, or from the web. As of the Fall Creators Update, web images can be up to 3 MB on normal connections and 1 MB on metered connections. On devices not yet running the Fall Creators Update, web images must be no larger than 200 KB.



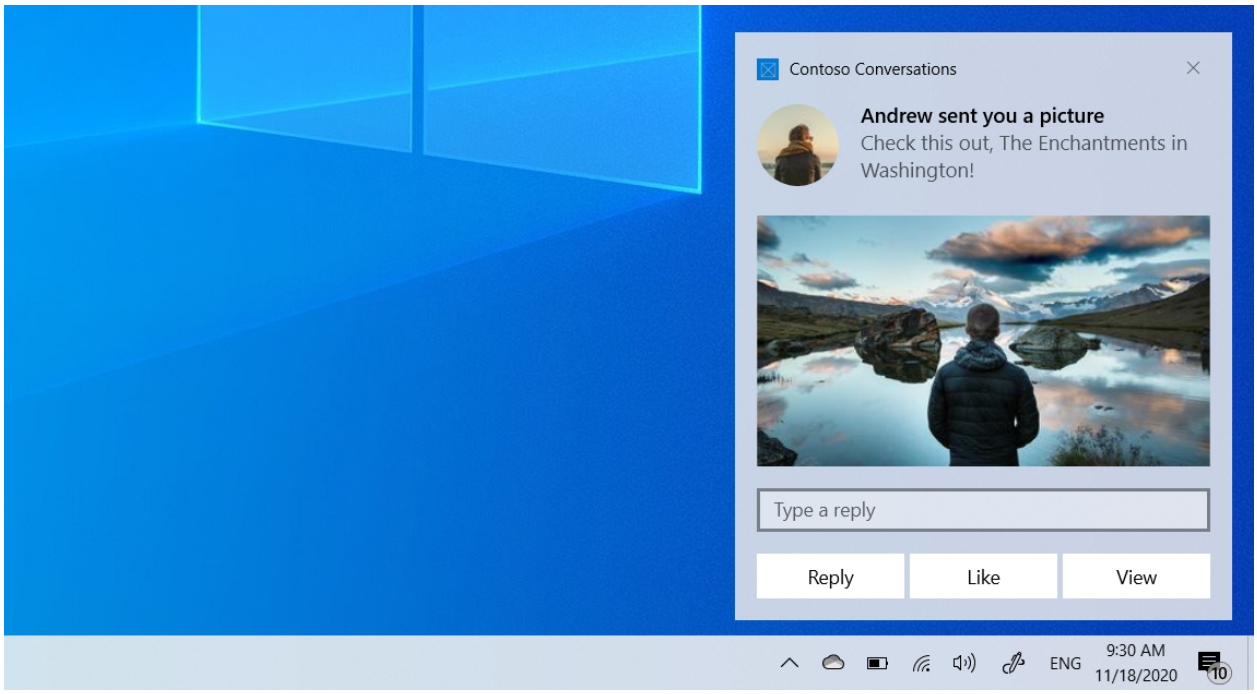
Builder syntax

C++

```
// Construct the content and show the toast!
(ref new ToastContentBuilder())
    ...

    // Inline image
    ->AddInlineImage(ref new Uri("https://picsum.photos/360/202?
image=883"))

    // Profile (app logo override) image
    ->AddAppLogoOverride(ref new Uri("ms-appdata:///local/Andrew.jpg"),
ToastGenericAppLogoCrop::Circle)

    ->Show();
```

Adding buttons and inputs

You can add buttons and inputs to make your notifications interactive. Buttons can launch your foreground app, a protocol, or your background task. We'll add a reply text box, a "Like" button, and a "View" button that opens the image.



Builder syntax

C++

```
// Construct the content
(ref new ToastContentBuilder())
    ->AddArgument("conversationId", 9813)
    ...
```

```
    // Text box for replying
    ->AddInputTextBox("tbReply", "Type a response")

    // Buttons
    ->AddButton((ref new ToastButton())
        ->SetContent("Reply")
        ->AddArgument("action", "reply")
        ->SetBackgroundActivation())

    ->AddButton((ref new ToastButton())
        ->SetContent("Like")
        ->AddArgument("action", "like")
        ->SetBackgroundActivation())

    ->AddButton((ref new ToastButton())
        ->SetContent("View")
        ->AddArgument("action", "view"))

    ->Show();
```

The activation of foreground buttons are handled in the same way as the main toast body (your App.xaml.cpp OnActivated will be called).

Handling background activation

When you specify background activation on your toast (or on a button inside the toast), your background task will be executed instead of activating your foreground app.

For more information on background tasks, please see Support your app with background tasks.

If you are targeting build 14393 or later, you can use in-process background tasks, which greatly simplify things. Note that in-process background tasks will fail to run on older versions of Windows. We'll use an in-process background task in this code sample.

C#

```
const string taskName = "ToastBackgroundTask";

// If background task is already registered, do nothing
if (BackgroundTaskRegistration.AllTasks.Any(i =>
i.Value.Name.Equals(taskName)))
    return;

// Otherwise request access
BackgroundAccessStatus status = await
BackgroundExecutionManager.RequestAccessAsync();
```

```
// Create the background task
BackgroundTaskBuilder builder = new BackgroundTaskBuilder()
{
    Name = taskName
};

// Assign the toast action trigger
builder.SetTrigger(new ToastNotificationActionTrigger());

// And register the task
BackgroundTaskRegistration registration = builder.Register();
```

Then in your App.xaml.cs, override the OnBackgroundActivated method. You can then retrieve the pre-defined arguments and user input, similar to the foreground activation.

App.xaml.cs

C#

```
protected override async void
OnBackgroundActivated(BackgroundActivatedEventArgs args)
{
    var deferral = args.TaskInstance.GetDeferral();

    switch (args.TaskInstance.Task.Name)
    {
        case "ToastBackgroundTask":
            var details = args.TaskInstance.TriggerDetails as
ToastNotificationActionTriggerDetail;
            if (details != null)
            {
                string arguments = details.Argument;
                var userInput = details.UserInput;

                // Perform tasks
            }
            break;
    }

    deferral.Complete();
}
```

Set an expiration time

In Windows 10 and 11, all toast notifications go in Action Center after they are dismissed or ignored by the user, so users can look at your notification after the popup is gone.

However, if the message in your notification is only relevant for a period of time, you should set an expiration time on the toast notification so the users do not see stale information from your app. For example, if a promotion is only valid for 12 hours, set the expiration time to 12 hours. In the code below, we set the expiration time to be 2 days.

> **Note**
>
> The default and maximum expiration time for local toast notifications is 3 days.

C++

```
// Create toast content and show the toast!
(ref new ToastContentBuilder())
    ->AddText("Expires in 2 days...")
    ->Show(toast =>
    {
        toast->ExpirationTime = DateTime::Now->AddDays(2);
    });
```

Provide a primary key for your toast

If you want to programmatically remove or replace the notification you send, you need to use the Tag property (and optionally the Group property) to provide a primary key for your notification. Then, you can use this primary key in the future to remove or replace the notification.

To see more details on replacing/removing already delivered toast notifications, please see Quickstart: Managing toast notifications in action center (XAML).

Tag and Group combined act as a composite primary key. Group is the more generic identifier, where you can assign groups like "wallPosts", "messages", "friendRequests", etc. And then Tag should uniquely identify the notification itself from within the group. By using a generic group, you can then remove all notifications from that group by using the RemoveGroup API.

C++

```
// Create toast content and show the toast!
(ref new ToastContentBuilder())
    ->AddText("New post on your wall!")
    ->Show(toast =>
    {
        toast.Tag = "18365";
```

```
        toast.Group = "wallPosts";
    });
```

Clear your notifications

Apps are responsible for removing and clearing their own notifications. When your app is launched, we do NOT automatically clear your notifications.

Windows will only automatically remove a notification if the user explicitly clicks the notification.

Here's an example of what a messaging app should do...

1. User receives multiple toasts about new messages in a conversation
2. User taps one of those toasts to open the conversation
3. The app opens the conversation and then clears all toasts for that conversation (by using RemoveGroup on the app-supplied group for that conversation)
4. User's Action Center now properly reflects the notification state, since there are no stale notifications for that conversation left in Action Center.

To learn about clearing all notifications or removing specific notifications, see Quickstart: Managing toast notifications in action center (XAML).

C++

```
ToastNotificationManagerCompat::History->Clear();
```

Resources

- Full C# code sample on GitHub
- Toast content documentation
- ToastNotification Class
- ToastNotificationActivatedEventArgs Class

Send a local toast notification from a WRL C++ desktop app

Article • 10/27/2022

Packaged and unpackaged desktop apps can send interactive toast notifications just like Universal Windows platform (UWP) apps can. That includes packaged apps (see Create a new project for a packaged WinUI 3 desktop app); packaged apps with external location (see Grant package identity by packaging with external location); and unpackaged apps (see Create a new project for an unpackaged WinUI 3 desktop app).

However, for an unpackaged desktop app, there are a few special steps. That's due to the different activation schemes, and the lack of package identity at runtime.

> **Important**
>
> If you're writing a UWP app, please see the **UWP documentation**. For other desktop languages, please see **Desktop C#**.

Step 1: Enable the Windows SDK

If you haven't enabled the Windows SDK for your app, then you must do that first. There are a few key steps.

1. Add `runtimeobject.lib` to **Additional Dependencies**.
2. Target the Windows SDK.

Right click your project and select **Properties**.

In the top **Configuration** menu, select **All Configurations** so that the following change is applied to both Debug and Release.

Under **Linker -> Input**, add `runtimeobject.lib` to the **Additional Dependencies**.

Then under **General**, make sure that the **Windows SDK Version** is set to version 10.0 or later.

Step 2: Copy compat library code

Copy the DesktopNotificationManagerCompat.h and DesktopNotificationManagerCompat.cpp file from GitHub into your project. The

compat library abstracts much of the complexity of desktop notifications. The following instructions require the compat library.

If you're using precompiled headers, make sure to `#include "stdafx.h"` as the first line of the DesktopNotificationManagerCompat.cpp file.

Step 3: Include the header files and namespaces

Include the compat library header file, and the header files and namespaces related to using the Windows toast APIs.

C++

```
#include "DesktopNotificationManagerCompat.h"
#include <NotificationActivationCallback.h>
#include <windows.ui.notifications.h>

using namespace ABI::Windows::Data::Xml::Dom;
using namespace ABI::Windows::UI::Notifications;
using namespace Microsoft::WRL;
```

Step 4: Implement the activator

You must implement a handler for toast activation, so that when the user clicks on your toast, your app can do something. This is required for your toast to persist in Action Center (since the toast could be clicked days later when your app is closed). This class can be placed anywhere in your project.

Implement the **INotificationActivationCallback** interface as seen below, including a UUID, and also call **CoCreatableClass** to flag your class as COM creatable. For your UUID, create a unique GUID using one of the many online GUID generators. This GUID CLSID (class identifier) is how Action Center knows what class to COM activate.

C++

```
// The UUID CLSID must be unique to your app. Create a new GUID if copying
this code.
class DECLSPEC_UUID("replaced-with-your-guid-C173E6ADF0C3")
NotificationActivator WrlSealed WrlFinal
    : public RuntimeClass<RuntimeClassFlags<ClassicCom>,
INotificationActivationCallback>
{
public:
```

```
    virtual HRESULT STDMETHODCALLTYPE Activate(
        _In_ LPCWSTR appUserModelId,
        _In_ LPCWSTR invokedArgs,
        _In_reads_(dataCount) const NOTIFICATION_USER_INPUT_DATA* data,
        ULONG dataCount) override
    {
        // TODO: Handle activation
    }
};

// Flag class as COM creatable
CoCreatableClass(NotificationActivator);
```

Step 5: Register with notification platform

Then, you must register with the notification platform. There are different steps depending on whether your app is packaged or unpackaged. If you support both, then you must perform both sets of steps (however, there's no need to fork your code since our library handles that for you).

Packaged

If your app is packaged (see Create a new project for a packaged WinUI 3 desktop app) or packaged with external location (see Grant package identity by packaging with external location), or if you support both, then in your **Package.appxmanifest** add:

1. Declaration for **xmlns:com**
2. Declaration for **xmlns:desktop**
3. In the **IgnorableNamespaces** attribute, **com** and **desktop**
4. **com:Extension** for the COM activator using the GUID from step #4. Be sure to include the `Arguments="-ToastActivated"` so that you know your launch was from a toast
5. **desktop:Extension** for **windows.toastNotificationActivation** to declare your toast activator CLSID (the GUID from step #4).

Package.appxmanifest

XML

```
<Package
  ...
  xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"

xmlns:desktop="http://schemas.microsoft.com/appx/manifest/desktop/windows10"
  IgnorableNamespaces="... com desktop">
```

```
  ...
  <Applications>
    <Application>
      ...
      <Extensions>

        <!--Register COM CLSID LocalServer32 registry key-->
        <com:Extension Category="windows.comServer">
          <com:ComServer>
            <com:ExeServer Executable="YourProject\YourProject.exe"
Arguments="-ToastActivated" DisplayName="Toast activator">
              <com:Class Id="replaced-with-your-guid-C173E6ADF0C3"
DisplayName="Toast activator"/>
            </com:ExeServer>
          </com:ComServer>
        </com:Extension>

        <!--Specify which CLSID to activate when toast clicked-->
        <desktop:Extension Category="windows.toastNotificationActivation">
          <desktop:ToastNotificationActivation
ToastActivatorCLSID="replaced-with-your-guid-C173E6ADF0C3" />
        </desktop:Extension>

      </Extensions>
    </Application>
  </Applications>
 </Package>
```

Unpackaged

If your app is unpackaged (see Create a new project for an unpackaged WinUI 3 desktop app), or if you support both, then you have to declare your Application User Model ID (AUMID) and toast activator CLSID (the GUID from step #4) on your app's shortcut in Start.

Pick a unique AUMID that will identify your app. This is typically in the form of [CompanyName].[AppName]. But you want to ensure that it's unique across all apps (so feel free to add some digits at the end).

Step 5.1: WiX Installer

If you're using WiX for your installer, edit the **Product.wxs** file to add the two shortcut properties to your Start menu shortcut as seen below. Be sure that your GUID from step #4 is enclosed in `{}` as seen below.

Product.wxs

XML

```
<Shortcut Id="ApplicationStartMenuShortcut" Name="Wix Sample"
Description="Wix Sample" Target="[INSTALLFOLDER]WixSample.exe"
WorkingDirectory="INSTALLFOLDER">

    <!--AUMID-->
    <ShortcutProperty Key="System.AppUserModel.ID"
Value="YourCompany.YourApp"/>

    <!--COM CLSID-->
    <ShortcutProperty Key="System.AppUserModel.ToastActivatorCLSID" Value="
{replaced-with-your-guid-C173E6ADF0C3}"/>

</Shortcut>
```

> **ⓘ Important**
>
> In order to actually use notifications, you must install your app through the installer once before debugging normally, so that the Start shortcut with your AUMID and CLSID is present. After the Start shortcut is present, you can debug using F5 from Visual Studio.

Step 5.2: Register AUMID and COM server

Then, regardless of your installer, in your app's startup code (before calling any notification APIs), call the **RegisterAumidAndComServer** method, specifying your notification activator class from step #4 and your AUMID used above.

C++

```
// Register AUMID and COM server (for a packaged app, this is a no-
operation)
hr =
DesktopNotificationManagerCompat::RegisterAumidAndComServer(L"YourCompany.Yo
urApp", __uuidof(NotificationActivator));
```

If your app supports both packaged and unpackaged deployment, then feel free to call this method regardless. If you're running packaged (that is, with package identity at runtime), then this method will simply return immediately. There's no need to fork your code.

This method allows you to call the compat APIs to send and manage notifications without having to constantly provide your AUMID. And it inserts the LocalServer32 registry key for the COM server.

Step 6: Register COM activator

For both packaged and unpackaged apps, you must register your notification activator type, so that you can handle toast activations.

In your app's startup code, call the following **RegisterActivator** method. This must be called in order for you to receive any toast activations.

C++

```
// Register activator type
hr = DesktopNotificationManagerCompat::RegisterActivator();
```

Step 7: Send a notification

Sending a notification is identical to UWP apps, except that you'll use **DesktopNotificationManagerCompat** to create a **ToastNotifier**. The compat library automatically handles the difference between packaged and unpackaged apps, so you don't need to fork your code. For an unpackaged app, the compat library caches the AUMID that you provided when you called **RegisterAumidAndComServer** so that you don't need to worry about when to provide or not provide the AUMID.

Make sure you use the **ToastGeneric** binding as seen below, since the legacy Windows 8.1 toast notification templates won't activate the COM notification activator that you created in step #4.

> **ⓘ Important**
>
> Http images are supported only in packaged apps that have the internet capability in their manifest. Unpackaged apps don't support http images; you must download the image to your local app data, and reference it locally.

C++

```
// Construct XML
ComPtr<IXmlDocument> doc;
hr = DesktopNotificationManagerCompat::CreateXmlDocumentFromString(
    L"<toast><visual><binding template='ToastGeneric'><text>Hello
world</text></binding></visual></toast>",
    &doc);
if (SUCCEEDED(hr))
{
    // See full code sample to learn how to inject dynamic text, buttons,
and more
```

```
    // Create the notifier
    // Desktop apps must use the compat method to create the notifier.
    ComPtr<IToastNotifier> notifier;
    hr = DesktopNotificationManagerCompat::CreateToastNotifier(&notifier);
    if (SUCCEEDED(hr))
    {
        // Create the notification itself (using helper method from compat
library)
        ComPtr<IToastNotification> toast;
        hr =
DesktopNotificationManagerCompat::CreateToastNotification(doc.Get(),
&toast);
        if (SUCCEEDED(hr))
        {
            // And show it!
            hr = notifier->Show(toast.Get());
        }
    }
}
```

> **ⓘ Important**
>
> Desktop apps can't use legacy toast templates (such as ToastText02). Activation of the legacy templates will fail when the COM CLSID is specified. You must use the Windows ToastGeneric templates, as seen above.

Step 8: Handling activation

When the user clicks on your toast, or buttons in the toast, the **Activate** method of your **NotificationActivator** class is invoked.

Inside the Activate method, you can parse the args that you specified in the toast and obtain the user input that the user typed or selected, and then activate your app accordingly.

> **ⓘ Note**
>
> The **Activate** method is called on a separate thread from your main thread.

C++

```
// The GUID must be unique to your app. Create a new GUID if copying this
code.
class DECLSPEC_UUID("replaced-with-your-guid-C173E6ADF0C3")
```

```
NotificationActivator WrlSealed WrlFinal
    : public RuntimeClass<RuntimeClassFlags<ClassicCom>,
INotificationActivationCallback>
{
public:
    virtual HRESULT STDMETHODCALLTYPE Activate(
        _In_ LPCWSTR appUserModelId,
        _In_ LPCWSTR invokedArgs,
        _In_reads_(dataCount) const NOTIFICATION_USER_INPUT_DATA* data,
        ULONG dataCount) override
    {
        std::wstring arguments(invokedArgs);
        HRESULT hr = S_OK;

        // Background: Quick reply to the conversation
        if (arguments.find(L"action=reply") == 0)
        {
            // Get the response user typed.
            // We know this is first and only user input since our toasts
only have one input
            LPCWSTR response = data[0].Value;

            hr = DesktopToastsApp::SendResponse(response);
        }

        else
        {
            // The remaining scenarios are foreground activations,
            // so we first make sure we have a window open and in foreground
            hr = DesktopToastsApp::GetInstance()->OpenWindowIfNeeded();
            if (SUCCEEDED(hr))
            {
                // Open the image
                if (arguments.find(L"action=viewImage") == 0)
                {
                    hr = DesktopToastsApp::GetInstance()->OpenImage();
                }

                // Open the app itself
                // User might have clicked on app title in Action Center
which launches with empty args
                else
                {
                    // Nothing to do, already launched
                }
            }
        }

        if (FAILED(hr))
        {
            // Log failed HRESULT
        }

        return S_OK;
    }
```

```
    ~NotificationActivator()
    {
        // If we don't have window open
        if (!DesktopToastsApp::GetInstance()->HasWindow())
        {
            // Exit (this is for background activation scenarios)
            exit(0);
        }
    }
};

// Flag class as COM creatable
CoCreatableClass(NotificationActivator);
```

To properly support being launched while your app is closed, in your WinMain function, you'll want to determine whether you're being launched from a toast or not. If launched from a toast, there will be a launch arg of "-ToastActivated". When you see this, you should stop performing any normal launch activation code, and allow your **NotificationActivator** to handle launching windows if needed.

C++

```
// Main function
int WINAPI wWinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE, _In_
LPWSTR cmdLineArgs, _In_ int)
{
    RoInitializeWrapper winRtInitializer(RO_INIT_MULTITHREADED);

    HRESULT hr = winRtInitializer;
    if (SUCCEEDED(hr))
    {
        // Register AUMID and COM server (for a packaged app, this is a no-
operation)
        hr =
DesktopNotificationManagerCompat::RegisterAumidAndComServer(L"WindowsNotific
ations.DesktopToastsCpp", __uuidof(NotificationActivator));
        if (SUCCEEDED(hr))
        {
            // Register activator type
            hr = DesktopNotificationManagerCompat::RegisterActivator();
            if (SUCCEEDED(hr))
            {
                DesktopToastsApp app;
                app.SetHInstance(hInstance);

                std::wstring cmdLineArgsStr(cmdLineArgs);

                // If launched from toast
                if (cmdLineArgsStr.find(TOAST_ACTIVATED_LAUNCH_ARG) !=
std::string::npos)
                {
```

```
                    // Let our NotificationActivator handle activation
                }

                else
                {
                    // Otherwise launch like normal
                    app.Initialize(hInstance);
                }

                app.RunMessageLoop();
            }
        }
    }

    return SUCCEEDED(hr);
}
```

Activation sequence of events

The activation sequence is the following...

If your app is already running:

1. **Activate** in your **NotificationActivator** is called

If your app is not running:

1. Your app is EXE launched, you get a command line args of "-ToastActivated"
2. **Activate** in your **NotificationActivator** is called

Foreground vs background activation

For desktop apps, foreground and background activation is handled identically—your COM activator is called. It's up to your app's code to decide whether to show a window or to simply perform some work and then exit. Therefore, specifying an **activationType** of **background** in your toast content doesn't change the behavior.

Step 9: Remove and manage notifications

Removing and managing notifications is identical to UWP apps. However, we recommend you use our compat library to obtain a **DesktopNotificationHistoryCompat** so you don't have to worry about providing the AUMID for a desktop app.

```
C++
```

```
std::unique_ptr<DesktopNotificationHistoryCompat> history;
auto hr = DesktopNotificationManagerCompat::get_History(&history);
if (SUCCEEDED(hr))
{
    // Remove a specific toast
    hr = history->Remove(L"Message2");

    // Clear all toasts
    hr = history->Clear();
}
```

Step 10: Deploying and debugging

To deploy and debug your packaged app, see Run, debug, and test a packaged desktop app.

To deploy and debug your desktop app, you must install your app through the installer once before debugging normally, so that the Start shortcut with your AUMID and CLSID is present. After the Start shortcut is present, you can debug using F5 from Visual Studio.

If your notifications simply fail to appear in your desktop app (and no exceptions are thrown), that likely means the Start shortcut isn't present (install your app via the installer), or the AUMID you used in code doesn't match the AUMID in your Start shortcut.

If your notifications appear but aren't persisted in Action Center (disappearing after the popup is dismissed), that means you haven't implemented the COM activator correctly.

If you've installed both your packaged and unpackaged desktop app, note that the packaged app will supersede the unpackaged app when handling toast activations. That means that toasts from the unpackaged app will launch the packaged app when clicked. Uninstalling the packaged app will revert activations back to the unpackaged app.

If you receive `HRESULT 0x800401f0 CoInitialize has not been called.`, be sure to call `CoInitialize(nullptr)` in your app before calling the APIs.

If you receive `HRESULT 0x8000000e A method was called at an unexpected time.` while calling the Compat APIs, that likely means you failed to call the required Register methods (or if a packaged app, you're not currently running your app under the packaged context).

If you get numerous `unresolved external symbol` compilation errors, you likely forgot to add `runtimeobject.lib` to the **Additional Dependencies** in step #1 (or you only added it to the Debug configuration and not Release configuration).

Handling older versions of Windows

If you support Windows 8.1 or lower, you'll want to check at runtime whether you're running on Windows before calling any **DesktopNotificationManagerCompat** APIs or sending any ToastGeneric toasts.

Windows 8 introduced toast notifications, but used the legacy toast templates, like ToastText01. Activation was handled by the in-memory **Activated** event on the **ToastNotification** class since toasts were only brief popups that weren't persisted. Windows 10 introduced interactive ToastGeneric toasts, and also introduced Action Center where notifications are persisted for multiple days. The introduction of Action Center required the introduction of a COM activator, so that your toast can be activated days after you created it.

Expand table

OS	ToastGeneric	COM activator	Legacy toast templates
Windows 10 and later	Supported	Supported	Supported (but won't activate COM server)
Windows 8.1 / 8	N/A	N/A	Supported
Windows 7 and lower	N/A	N/A	N/A

To check whether you're running on Windows 10 or later, include the `<VersionHelpers.h>` header, and check the **IsWindows10OrGreater** method. If that returns `true`, then continue calling all the methods described in this documentation.

C++

```
#include <VersionHelpers.h>

if (IsWindows10OrGreater())
{
    // Running on Windows 10 or later, continue with sending toasts!
}
```

Known issues

FIXED: App doesn't become focused after clicking toast: In builds 15063 and earlier, foreground rights weren't being transferred to your application when we activated the

COM server. Therefore, your app would simply flash when you tried to move it to the foreground. There was no workaround for this issue. We fixed this in builds 16299 or later.

Resources

- Full code sample on GitHub
- Toast notifications from desktop apps
- Toast content documentation

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Send a local toast notification from other types of unpackaged apps

Article • 10/24/2022

This topic is for you if you're developing an unpackaged app that's not C# or C++.

That is, if you're *not* developing a packaged app (see Create a new project for a packaged WinUI 3 desktop app), and you're not developing a packaged app with external location (see Grant package identity by packaging with external location), and your app isn't C# or C++.

A toast notification is a message that an app can construct and deliver to the user while the user is not currently using your app. This quickstart walks you through the steps to create, deliver, and display a Windows toast notification. These quickstart uses local notifications, which are the simplest notification to implement.

> **ⓘ Important**
>
> If you're writing a C# app, then please see the **C# documentation**. If you're writing a C++ app, the please see the **C++ UWP** or **C++ WRL** documentation.

Step 1: Register your app in the registry

You first need to register your app's information in the registry, including a unique AUMID that identifies your app, your app's display name, your icon, and a COM activator's GUID.

XML

```
<registryKey keyName="HKEY_LOCAL_MACHINE\Software\Classes\AppUserModelId\
<YOUR_AUMID>">
    <registryValue
        name="DisplayName"
        value="My App"
        valueType="REG_EXPAND_SZ" />
    <registryValue
        name="IconUri"
        value="C:\icon.png"
        valueType="REG_EXPAND_SZ" />
    <registryValue
        name="IconBackgroundColor"
        value="AARRGGBB"
        valueType="REG_SZ" />
```

```
    <registryValue
        name="CustomActivator"
        value="{YOUR COM ACTIVATOR GUID HERE}"
        valueType="REG_SZ" />
</registryKey>
```

Step 2: Set up your COM activator

Notifications can be clicked at any point in time, even when your app isn't running. Thus, notification activation is handled through a COM activator. Your COM class must implement the `INotificationActivationCallback` interface. The GUID for your COM class must match the GUID you specified in the registry CustomActivator value.

C++

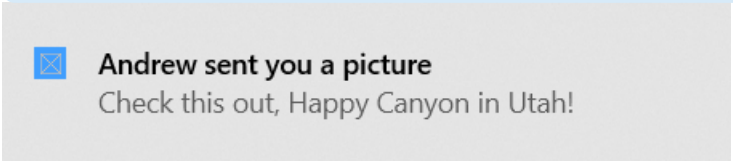
```
struct callback : winrt::implements<callback,
INotificationActivationCallback>
{
    HRESULT __stdcall Activate(
        LPCWSTR app,
        LPCWSTR args,
        [[maybe_unused]] NOTIFICATION_USER_INPUT_DATA const* data,
        [[maybe_unused]] ULONG count) noexcept final
    {
        try
        {
            std::wcout << this_app_name << L" has been called back from a
notification." << std::endl;
            std::wcout << L"Value of the 'app' parameter is '" << app <<
L"'." << std::endl;
            std::wcout << L"Value of the 'args' parameter is '" << args <<
L"'." << std::endl;
            return S_OK;
        }
        catch (...)
        {
            return winrt::to_hresult();
        }
    }
};
```

Step 3: Send a toast

In Windows 10, your toast notification content is described using an adaptive language that allows great flexibility with how your notification looks. See the toast content documentation for more information.

We'll start with a simple text-based notification. Construct the notification content (using the Notifications library), and show the notification!

> **ⓘ Important**
>
> You must use your AUMID from earlier when sending the notification so that the notification appears from your app.



C++

```
// Construct the toast template
XmlDocument doc;
doc.LoadXml(L"<toast>\
    <visual>\
        <binding template=\"ToastGeneric\">\
            <text></text>\
            <text></text>\
        </binding>\
    </visual>\
</toast>");

// Populate with text and values
doc.SelectSingleNode(L"//text[1]").InnerText(L"Andrew sent you a picture");
doc.SelectSingleNode(L"//text[2]").InnerText(L"Check this out, The
Enchantments in Washington!");

// Construct the notification
ToastNotification notif{ doc };

// And send it! Use the AUMID you specified earlier.
ToastNotificationManager::CreateToastNotifier(L"MyPublisher.MyApp").Show(not
if);
```

Step 4: Handling activation

Your COM activator will be activated when your notification is clicked.

More details

AUMID restrictions

The AUMID should be at most 129 characters long. If the AUMID is more than 129 characters long, scheduled toast notifications won't work - you'll get the following exception when adding a scheduled notification: *The data area passed to a system call is too small. (0x8007007A)*.

Activating toast notifications from desktop apps

Article • 03/03/2025

Packaged and unpackaged Win32 apps can send interactive toast notifications just like UWP apps can. That includes packaged apps (see Create a new project for a packaged WinUI 3 desktop app); packaged apps with external location (see Grant package identity by packaging with external location); and unpackaged apps (see Create a new project for an unpackaged WinUI 3 desktop app).

However, for an unpackaged Win32 app there are a few special steps. That's due to the different activation schemes, and the lack of package identity at runtime.

In this topic, we list out the options you have for sending a toast notification on Windows 10. Every option fully supports...

- Persisting in Action Center
- Being activatable from both the popup and inside Action Center
- Being activatable while your EXE isn't running

All options

The table below illustrates your options for supporting toasts within your desktop app, and the corresponding supported features. You can use the table to select the best option for your scenario.

Expand table

Option	Visuals	Actions	Inputs	Activates in-process
COM activator	**Supported**	**Supported**	**Supported**	**Supported**
No COM / Stub CLSID	**Supported**	**Supported**	Not supported	Not supported

Preferred option - COM activator

This is the preferred option that works for desktop apps, and supports all notification features. Don't be afraid of the "COM activator"; we have a library for C# and C++ apps

that makes this very straightforward, even if you've never written a COM server before.

Expand table

Visuals	Actions	Inputs	Activates in-process
Supported	Supported	Supported	Supported

With the COM activator option, you can use the following notification templates and activation types in your app.

Expand table

Template and activation type	Packaged	Unpackaged
ToastGeneric Foreground	**Supported**	**Supported**
ToastGeneric Background	**Supported**	**Supported**
ToastGeneric Protocol	**Supported**	**Supported**
Legacy templates	**Supported**	Not supported

> **Note**
>
> If you add the COM activator to your existing packaged app, then Foreground/Background and Legacy notification activations will activate your COM activator instead of your command line.

To learn how to use this option, see Send a local toast notification from desktop C# apps or Send a local toast notification from Win32 C++ WRL apps.

Alternative option - No COM / Stub CLSID

This is an alternative option if you can't implement a COM activator. However, you'll sacrifice a few features, such as input support (text boxes on toasts) and activating in-process.

Expand table

Visuals	Actions	Inputs	Activates in-process
Supported	**Supported**	Not supported	Not supported

With this option, if you support desktop, then you're much more limited in the notification templates and activation types that you can use, as seen below.

Expand table

Template and activation type	Packaged	Unpackaged
ToastGeneric Foreground	**Supported**	Not supported
ToastGeneric Background	**Supported**	Not supported
ToastGeneric Protocol	**Supported**	**Supported**
Legacy templates	**Supported**	Not supported

For packaged apps, just send toast notifications like a UWP app would. When the user clicks on your toast, your app will be command-line launched with the launch args that you specified in the toast.

For unpackaged apps, set up the AUMID so that you can send toasts, and then also specify a CLSID on your shortcut. That can be any random GUID. Don't add the COM server/activator. You're adding a "stub" COM CLSID, which will cause Action Center to persist the notification. Note that you can use only protocol activation toasts, because the stub CLSID will break activation of any other toast activations. Therefore, you have to update your app to support protocol activation, and have the toast's protocol activate your own app.

Resources

- Send a local toast notification from desktop C# apps
- Send a local toast notification from Win32 C++ WRL apps
- Toast content documentation

App notification content

Article • 04/12/2024

App notifications are flexible notifications with text, images, and buttons/inputs. This article describes the UI elements that can be used in an app notification and provides code examples for generating the XML format for an app notification.

> **Note**
>
> The term "toast notification" is being replaced with "app notification". These terms both refer to the same feature of Windows, but over time we will phase out the use of "toast notification" in the documentation.

Getting started

App notifications are defined with an XML payload which is defined by the App notification schema. Currently, there are three ways of generating the XML payload fo an app notification. The code examples in this article demonstrate all three methods:

- The Microsoft.Windows.AppNotifications.Builder APIs - Introduced in Windows App SDK 1.2, this namespace provides APIs that allow you to easily build the XML payload for a notification programmatically without having to worry about the specifics of the XML format. Code examples using these APIs are in the tabs labeled "Windows App SDK".
- The `Microsoft.Toolkit.Uwp.Notifications` builder syntax - These APIs are part of the UWP Community Toolkit and provides support for UWP apps. Although these APIs can also be used for Windows App SDK apps, and continue to be supported, we recommend that new implementations use the **Microsoft.Windows.AppNotifications.Builder** APIs. To use Community Toolkit APIs, add the UWP Community Toolkit Notifications nuget package to your project. The C# samples provided in this article use version 7.0.0 of the NuGet package. Code examples using these APIs are in the tabs labeled "Windows Community Toolkit".
- Raw XML - If you prefer, you can create your own custom code to generate XML strings in the required format. The raw XML examples are in the tabs labeled "XML".

Install Notifications Visualizer. This free Windows app helps you design interactive app notifications by providing an instant visual preview of your toast as you edit it, similar to

Visual Studio's XAML editor/design view. See Notifications Visualizer for more information, or download Notifications Visualizer from the Store.

This article only covers creating the app notification content. For information on sending a notification after you have generated the XML payload, see Send a local app notification.

App notification structure

Some important, high-level components of an app notification XML payload include:

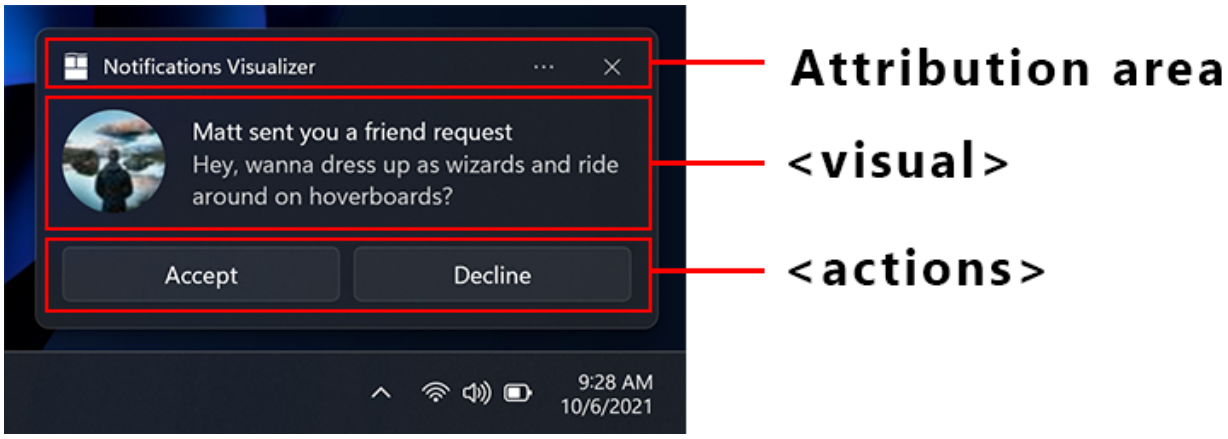
- **toast**: The **launch** attribute of this element defines what arguments will be passed back to your app when the user clicks your toast, allowing you to deep link into the correct content that the toast was displaying. To learn more, see Send a local app notification.
- **visual**: This element represents visual portion of the toast, including the generic binding that contains text and images.
- **actions**: This element represents interactive portion of the toast, including inputs and actions.
- **audio**: This element specifies the audio played when the toast is shown to the user.

Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddArgument("conversationId", "9813")

    .AddText("Some text")

    .AddButton(new AppNotificationButton("Archive")
        .AddArgument("action", "archive"))

    .SetAudioUri(new Uri("ms-appx:///Sound.mp3"));
```

Here is a visual representation of the app notification's content:



Attribution area

The attribution area is at the top of the app notification. Starting with Windows 11, your app's name and icon are displayed in this area. The attribution area also includes a close button that allows the user to quickly dismiss the notification and an ellipses menu that allows the user to quickly disable notifications for your app or go to the Windows Settings page for your app's notifications. The attribution area is configured by the shell and can't be overridden in the toast XML payload, although your app can add items to the attribution area context menu. For more information see Context menu actions.

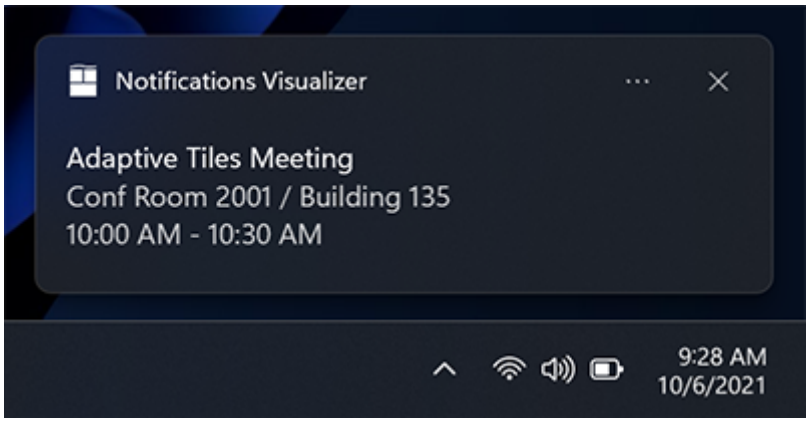
Visual

Each app notification must specify a **visual** element, where you must provide a generic toast binding and which can contain text and images. These elements will be rendered on various Windows devices, including desktop, phones, tablets, and Xbox.

For all attributes supported in the **visual** section and its child elements, see App notification schema.

Text elements

Each app notification must have at least one text element, and can contain two additional text elements, all of type **AdaptiveText**.



Since the Windows 10 Anniversary Update, you can control how many lines of text are displayed by using the **HintMaxLines** property on the text. The default (and maximum) is up to 2 lines of text for the title, and up to 4 lines (combined) for the two additional description elements (the second and third **AdaptiveText**).

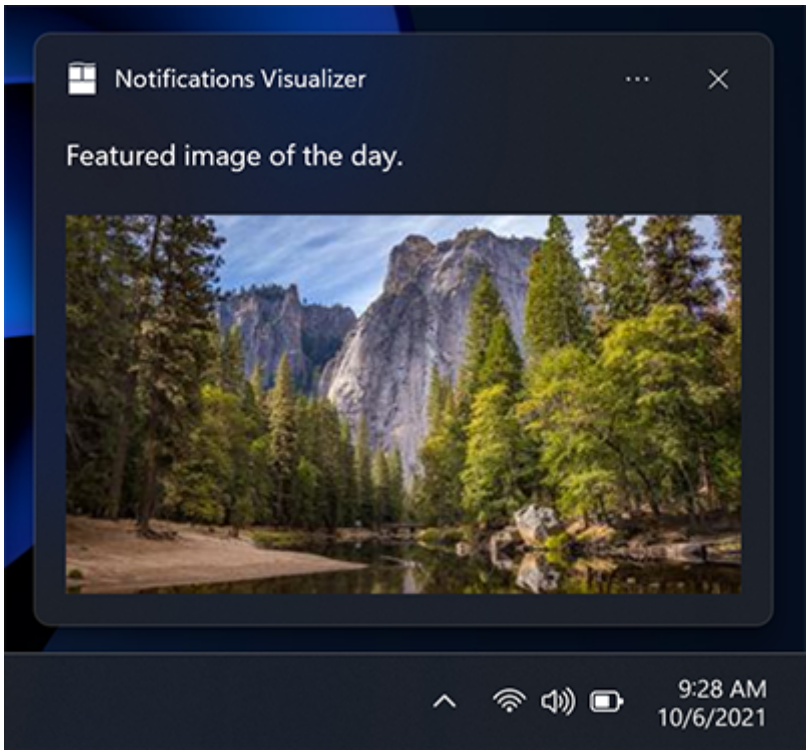
Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddArgument("conversationId", 9813)
    .AddText("Adaptive Tiles Meeting", new
AppNotificationTextProperties().SetMaxLines(1))
    .AddText("Conf Room 2001 / Building 135")
    .AddText("10:00 AM - 10:30 AM");
```

Inline image

By default, images are displayed inline, after any text elements, filling the full width of the visual area.



Windows App SDK

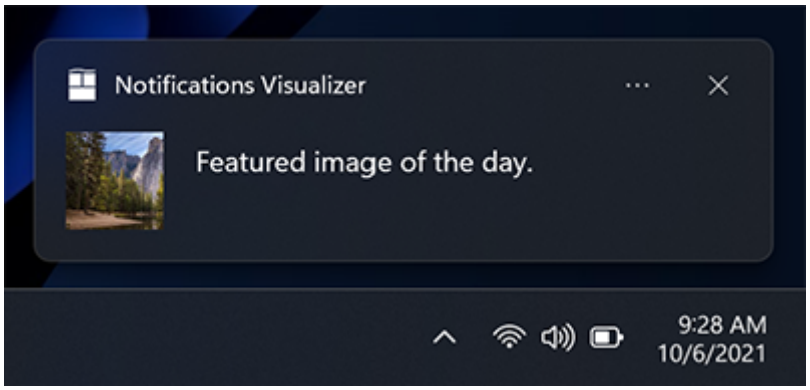
C#

```
var builder = new AppNotificationBuilder()
    .AddText("Featured image of the day.")
    .SetInlineImage(new Uri("ms-appx:///Images/InlineImage.png"));

AppNotificationManager.Default.Show(builder.BuildNotification());
```

App logo override

Specifying a **placement** value of "appLogoOverride" will cause the image to be displayed in a square on the left side of the visual area. The name of this property reflects the behavior in previous versions of Windows, where the image would replace the default app logo image. In Windows 11, the app logo is displayed in the attribution area, so it is not overridden by the **appLogoOverride** image placement.

Image dimensions are 48x48 pixels at 100% scaling. We generally recommend providing a version each icon asset for each scale factor: 100%, 125%, 150%, 200%, and 400%.



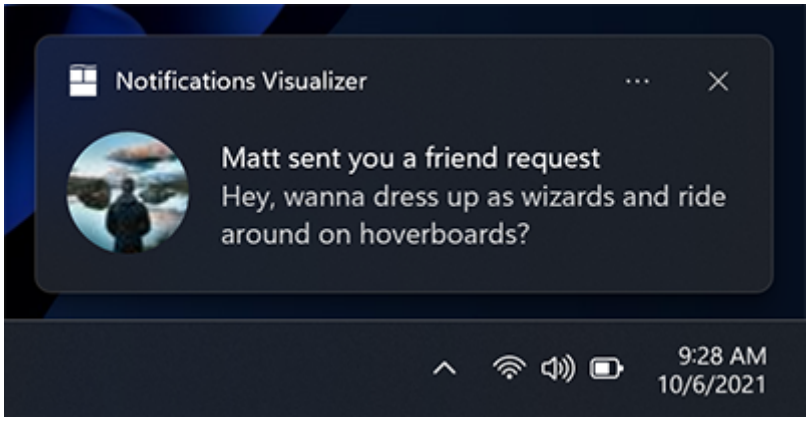
Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddText("Featured image of the day.")
    .SetAppLogoOverride(new Uri("ms-appx:///Images/AppLogo.png"));
```

Hint crop

Microsoft style guidelines recommend representing profile pictures with a circular image to provide a consistent representation of people across apps and the shell. Set the **HintCrop** property to **Circle** to render the image with a circular crop.



Windows App SDK

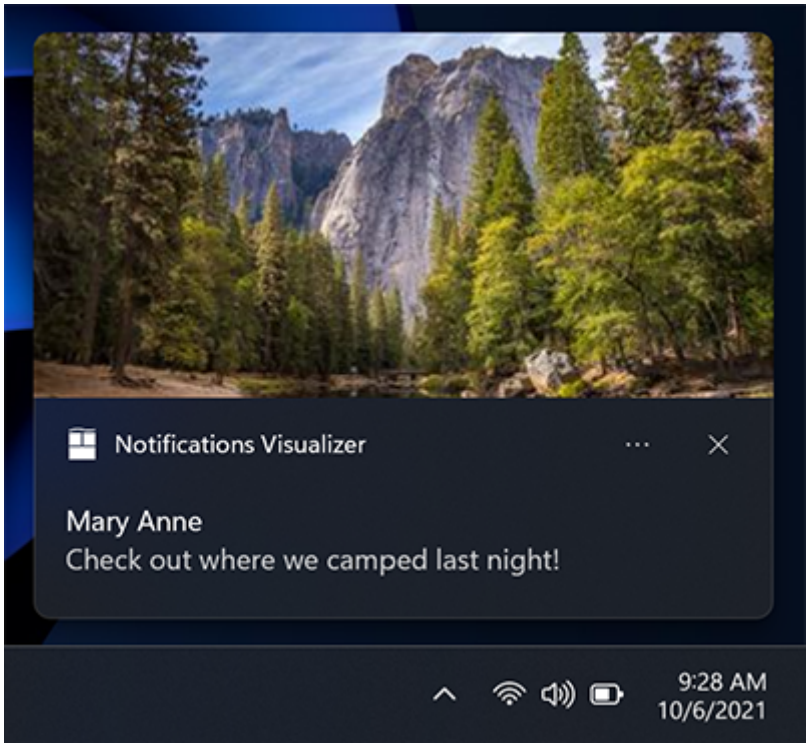
C#

```
var builder = new AppNotificationBuilder()
    .AddText("Matt sent you a friend request")
    .AddText("Hey, wanna dress up as wizards and ride around on
hoverboards?")
```

```
    .SetAppLogoOverride(new Uri("ms-appx:///Images/Profile.png"),
AppNotificationImageCrop.Circle);
```

Hero image

New in Anniversary Update: App notifications can display a hero image, which is a featured ToastGenericHeroImage displayed prominently within the toast banner and while inside Notification Center. Image dimensions are 364x180 pixels at 100% scaling.



Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddText("Marry Anne")
    .AddText("Check out where we camped last night!")
    .SetHeroImage(new Uri("ms-appx:///Images/HeroImage.png"));
```

Image size restrictions

The images you use in your toast notification can be sourced from...

- http://
- ms-appx:///

- ms-appdata:///

For http and https remote web images, there are limits on the file size of each individual image. In the Fall Creators Update (16299), we increased the limit to be 3 MB on normal connections and 1 MB on metered connections. Before that, images were always limited to 200 KB.

Expand table

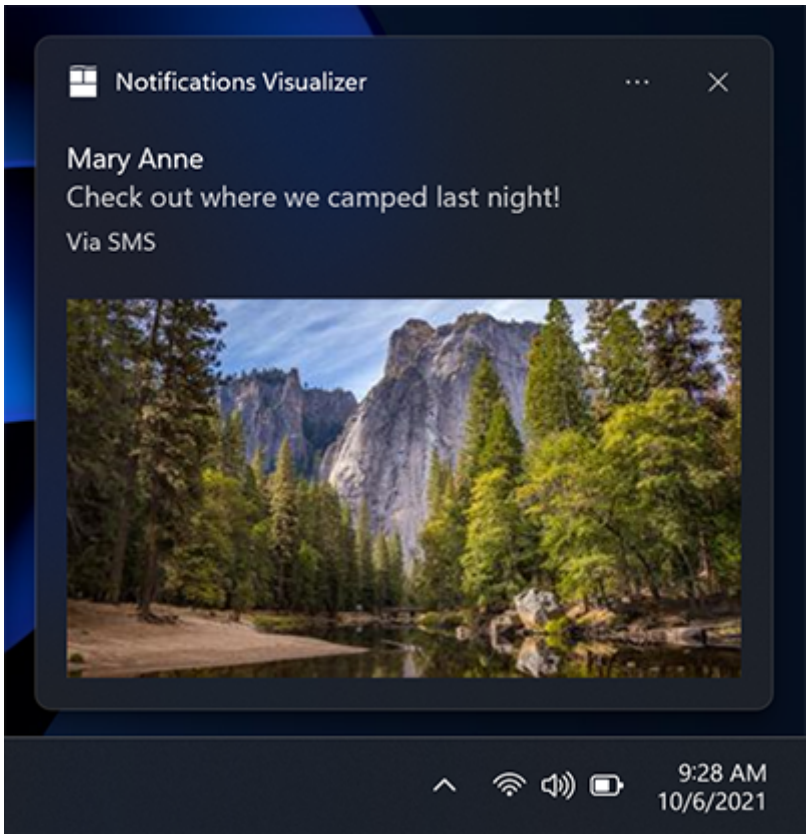
Normal connection	Metered connection	Before Fall Creators Update
3 MB	1 MB	200 KB

If an image exceeds the file size, or fails to download, or times out, the image will be dropped and the rest of the notification will be displayed.

Attribution text

New in Anniversary Update: If you need to reference the source of your content, you can use attribution text. This text is always displayed below any text elements, but above inline images. The text uses a slightly smaller size than standard text elements to help to distinguish from regular text elements.

On older versions of Windows that don't support attribution text, the text will simply be displayed as another text element (assuming you don't already have the maximum of three text elements).



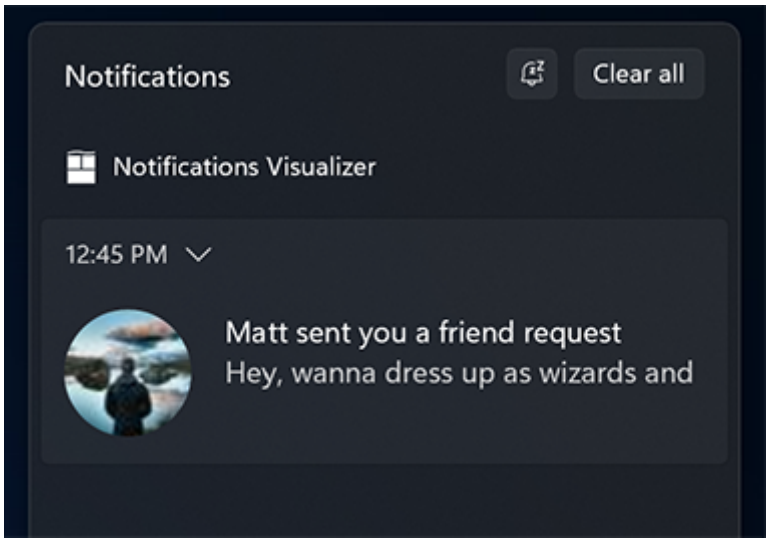
Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddText("Marry Anne")
    .AddText("Check out where we camped last night!")
    .SetAttributionText("via SMS");
    .SetHeroImage(new Uri("ms-appx:///Images/HeroImage.png"));
```

Custom timestamp

New in Creators Update: You can now override the system-provided timestamp with your own timestamp that accurately represents when the message/information/content was generated. This timestamp is visible within Notification Center.



To learn more about using a custom timestamp, please see custom timestamps on toasts.

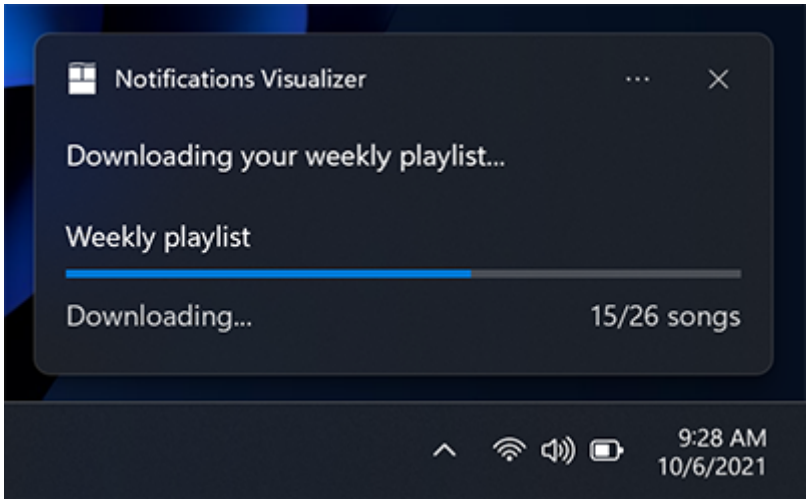
Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddText("Matt sent you a friend request")
    .AddText("Hey, wanna dress up as wizards and ride around on
hoverboards?")
    .SetTimeStamp(new DateTime(2017, 04, 15, 19, 45, 00,
DateTimeKind.Utc));
```

Progress bar

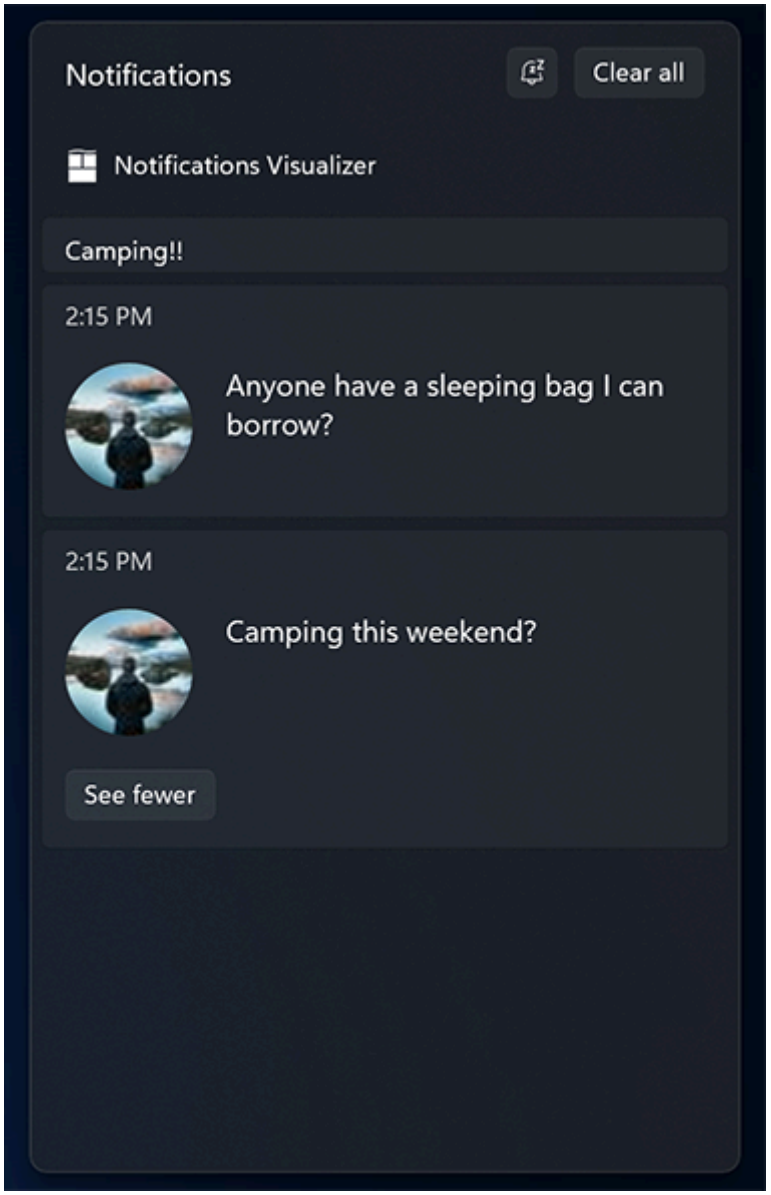
New in Creators Update: You can provide a progress bar on your app notification to keep the user informed of the progress of operations such as downloads.



To learn more about using a progress bar, please see Toast progress bar.

Headers

New in Creators Update: You can group notifications under headers within Notification Center. For example, you can group messages from a group chat under a header, or group notifications of a common theme under a header, or more.



To learn more about using headers, please see Toast headers.

Adaptive content

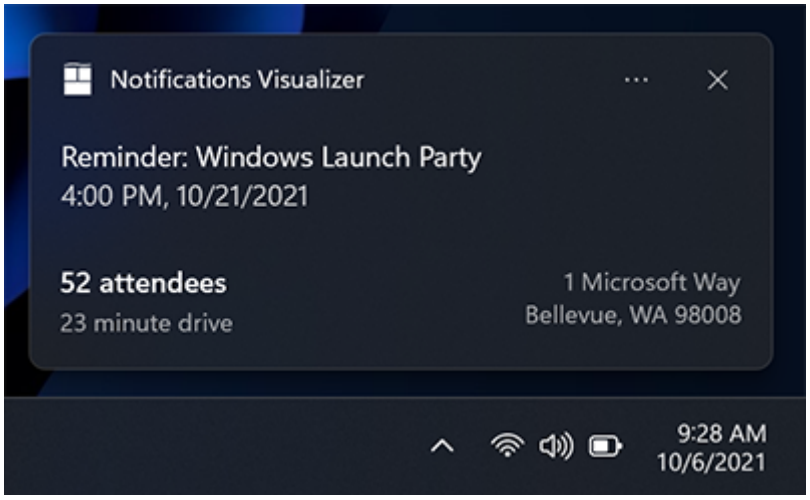
New in Anniversary Update: In addition to the content specified above, you can also display additional adaptive content that is visible when the toast is expanded.

This additional content is specified using Adaptive, which you can learn more about by reading the Adaptive Tiles documentation.

Note that any adaptive content must be contained within an **AdaptiveGroup**. Otherwise it will not be rendered using adaptive.

Columns and text elements

Here's an example where columns and some advanced adaptive text elements are used. Since the text elements are within an **AdaptiveGroup**, they support all the rich adaptive styling properties.



Windows App SDK

C#

```
// The Microsoft.Windows.AppNotifications.Builder syntax does not
currently support adaptive text elements.
```

Buttons

Buttons make your toast interactive, letting the user take quick actions on your app notification without interrupting their current workflow. For example, users can reply to a message directly from within a toast, or delete an email without even opening the email app. Buttons appear in the expanded portion of your notification.

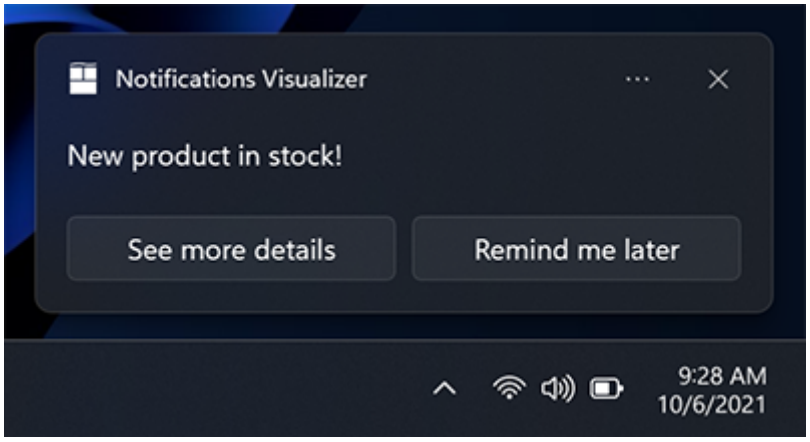
To learn more about implementing buttons end-to-end, see Send local toast.

Buttons can activate an app in the following ways:

- The app is activated in the foreground, with an argument that can be used to navigate to a specific page/context.
- Another app is activated via protocol launch.

- Background activation is supported explicitly for UWP apps. For Windows App SDK apps, the app is always launched in the foreground. The app can call AppInstance.GetActivatedEventArgs to detect if the activation was launched by a notification and determine from the passed arguments whether to fully launch the foreground app or just handle the notification and exit.
- System actions, such as snoozing or dismissing the notification, are supported both for UWP apps and for Windows App SDK. The **AppNotificationBuilder** APIs don't support this scenario but Windows App SDK apps can implement this scenario using the **Microsoft.Windows.AppNotifications.Builder** APIs or raw XML.

> **Note**
>
> You can only have up to 5 buttons (including context menu items which we discuss later).



Windows App SDK

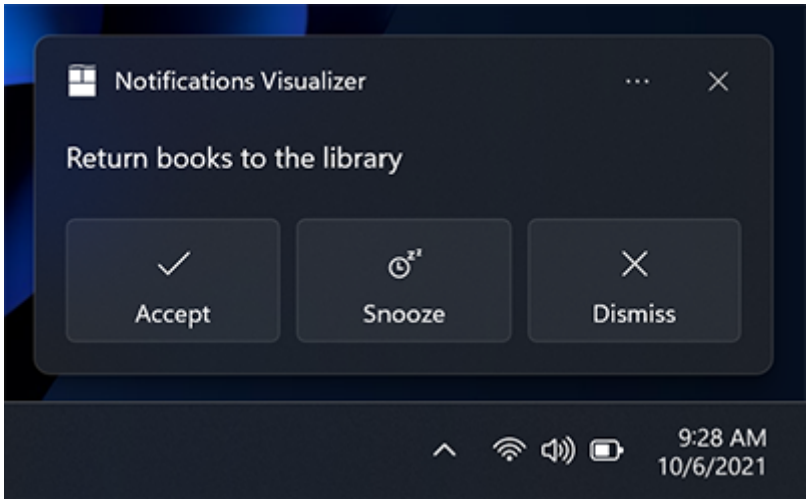
C#

```
new ToastContentBuilder()
    var builder = new AppNotificationBuilder()
        .AddText("New product in stock!")
        .AddButton(new AppNotificationButton("See more details")
            .AddArgument("action", "viewDetails"))
            .AddArgument("contentId", "351")
        .AddButton(new AppNotificationButton("Remind me later")
            .AddArgument("action", "remindLater"))
            .AddArgument("contentId", "351");
```

Buttons with icons

You can add icons to your buttons. These icons are white transparent 16x16 pixel images at 100% scaling, and should have no padding included in the image itself. If you choose to provide icons on a toast notification, you must provide icons for ALL of your buttons in the notification, as it transforms the style of your buttons into icon buttons.

> **Note**
>
> For accessibility, be sure to include a contrast-white version of the icon (a black icon for white backgrounds), so that when the user turns on High Contrast White mode, your icon is visible. For more information, see Tile and toast notification support for language, scale, and high contrast.



Windows App SDK

C#

```
new ToastContentBuilder()
    var builder = new AppNotificationBuilder()
        .AddText("Return books to the library.")
        .AddButton(new AppNotificationButton("Accept")
            .AddArgument("action", "accept")
            .SetIcon(new Uri("ms-appx:///Images/Accept.png")))
        .AddButton(new AppNotificationButton("Snooze")
            .AddArgument("action", "snooze")
            .SetIcon(new Uri("ms-appx:///Images/Snooze.png")))
        .AddButton(new AppNotificationButton("Dismiss")
            .AddArgument("action", "dismiss")
            .SetIcon(new Uri("ms-appx:///Images/Dismiss.png")));
```

New in Windows 11 Update: You can add tooltips to your icons with the **HintToolTip** property in XML. This is ideal if your buttons have icons but no content, as this will make

sure you can pass text that Windows Narrator can read. However, if content is present, then Narrator will read the content, no matter what is passed in the tooltip.

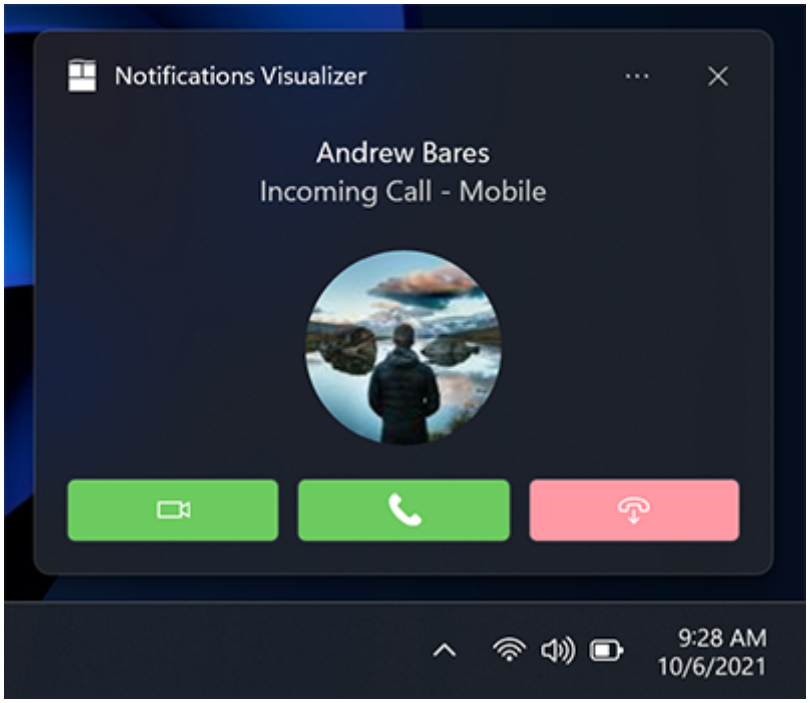
Windows App SDK

C#

```
var button = new AppNotificationButton("Reply")
    .AddArgument("action", "reply");

if (AppNotificationButton.IsToolTipSupported())
{
    button.ToolTip = "Click to reply.";
}

var builder = new AppNotificationBuilder()
    .AddText("Notification text.")
    .AddButton(button);
```

Buttons with colors

New in Windows 11 Update: You can add red or green colors to your buttons by adding the **useButtonStyle** attribute to the **toast** XML element and the **hint-buttonStyle** attribute to the action XML element as seen below.



Windows App SDK

C#

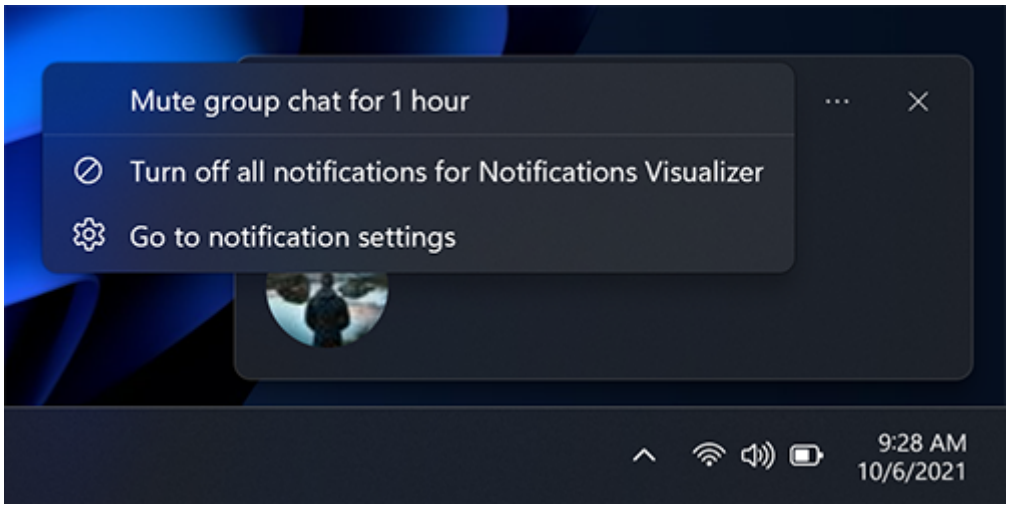
```
var builder = new AppNotificationBuilder()
    .SetScenario(AppNotificationScenario.IncomingCall)
    .AddText("Andrew Bares", new AppNotificationTextProperties()
        .SetIncomingCallAlignment())
      .AddText("Incoming Call - Mobile", new
AppNotificationTextProperties()
        .SetIncomingCallAlignment())
      .SetInlineImage(new Uri("ms-appx:///Images/Profile.png"),
        AppNotificationImageCrop.Circle)
    .AddButton(new AppNotificationButton()
        .SetToolTip("Answer Video Call")
        .SetButtonStyle(AppNotificationButtonStyle.Success)
        .SetIcon(new Uri("ms-appx:///Images/Video.png"))
        .AddArgument("videoId", "123"))
    .AddButton(new AppNotificationButton()
        .SetToolTip("Answer Phone Call")
        .SetButtonStyle(AppNotificationButtonStyle.Success)
        .SetIcon(new Uri("ms-appx:///Images/Call.png"))
        .AddArgument("callId", "123"))
    .AddButton(new AppNotificationButton()
        .SetToolTip("Hang Up")
        .SetButtonStyle(AppNotificationButtonStyle.Critical)
        .SetIcon(new Uri("ms-appx:///Images/HangUp.png"))
        .AddArgument("hangUpId", "123"));
```

Context menu actions

New in Anniversary Update: You can add additional context menu actions to the existing context menu that appears when the user right clicks your toast notification or selects the context menu icon.

> ⓘ **Note**
>
> On older devices, these additional context menu actions will simply appear as normal buttons on your notification.

The additional context menu actions you add (such as "Mute group chat for 1 hour") appear above the two default system entries.



Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddText("Camping this weekend?")
    .SetAppLogoOverride(new Uri("ms-appx:///images/Reply.png"),
AppNotificationImageCrop.Circle)
    .AddButton(new AppNotificationButton("Mute group chat for 1 hour")
        .AddArgument("action", "mute")
        .SetContextMenuPlacement());
```

> **Note**
>
> Additional context menu items contribute to the total limit of 5 buttons on a toast.

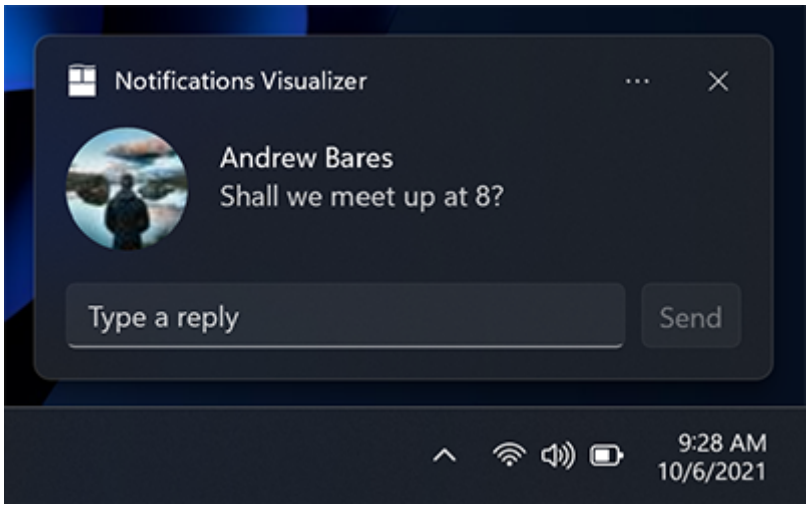
Activation of additional context menu items is handled identical to toast buttons.

Inputs

Inputs are specified within the **Actions** region of the app notification, meaning they are only visible when the notification is expanded.

Quick reply text box

To enable a quick reply text box (for example, in a messaging app) add a text input and a button, and reference the ID of the text input field so that the button is displayed next to the input field. The optional icon for the button, if provided, should be a 32x32 pixel image with no padding, white pixels set to transparent, and 100% scale.



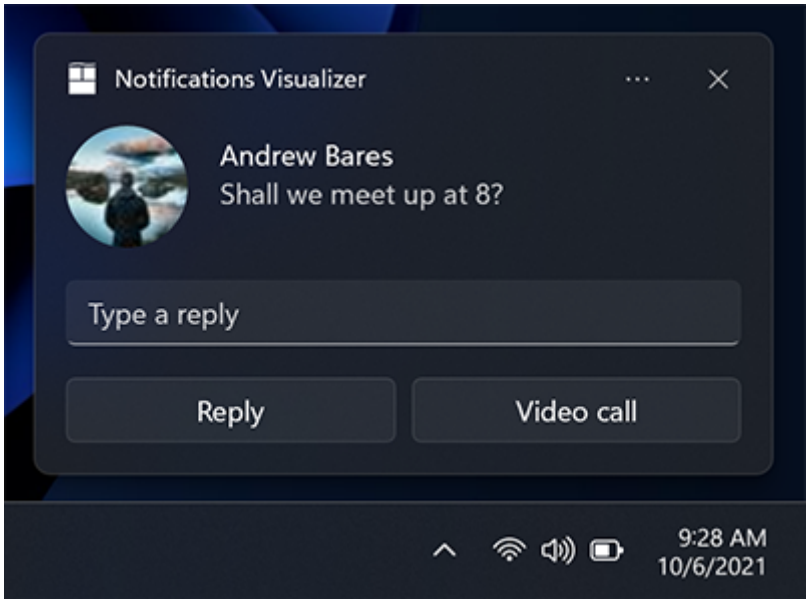
Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddTextBox("textBox", "Type a reply", "Reply")
    .AddButton(AppNotificationButton("Send")
        .AddArguments("action", "Send")
        .SetInputId("textBox"))
    .BuildNotification();
```

Inputs with buttons bar

You also can have one (or many) inputs with normal buttons displayed below the inputs.



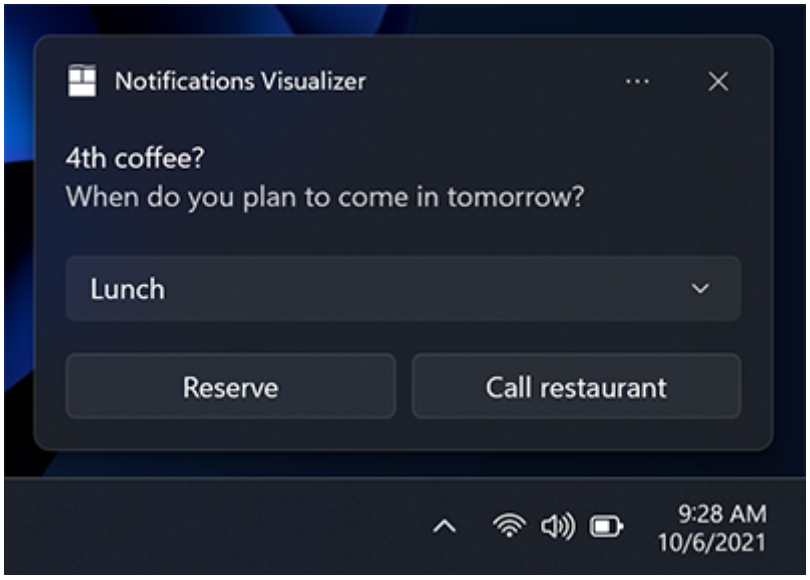
Windows App SDK

C#

```
// The Microsoft.Windows.AppNotifications.Builder syntax does not
currently support quick reply text boxes.
```

Selection input

In addition to text boxes, you can also use a selection menu.



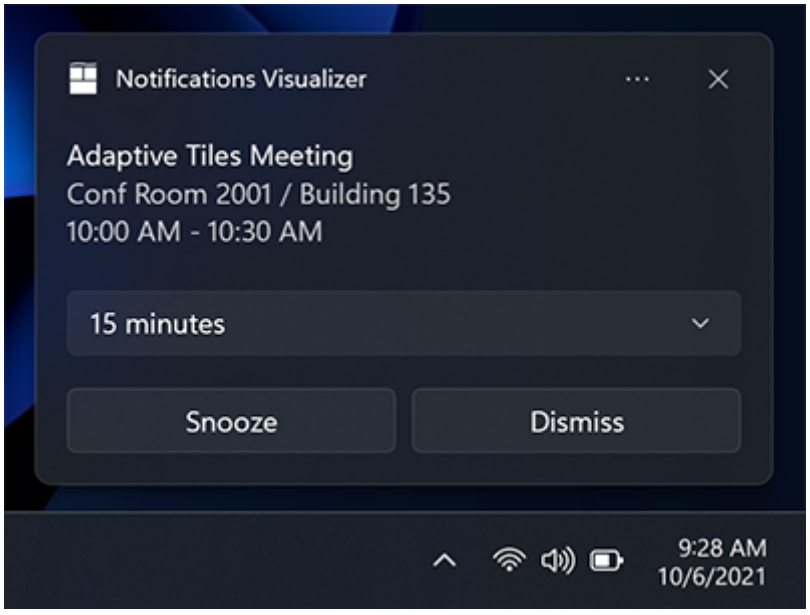
Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddText("4th coffee?")
    .AddText("When do you plan to come in tomorrow?")
    .AddComboBox(new AppNotificationComboBox("time")
        .SetTitle("Select an item:")
        .AddItem("breakfast", "Breakfast")
        .AddItem("lunch", "Lunch")
        .AddItem("dinner", "Dinner")
        .SetSelectedItem("lunch"))
    .AddButton(new AppNotificationButton("Reply")
        .AddArgument("action", "reply")
        .AddArgument("threadId", "9218")
        .SetContextMenuPlacement())
    .AddButton(new AppNotificationButton("Call restaurant")
        .AddArgument("action", "videocall")
        .AddArgument("threadId", "9218")
        .SetContextMenuPlacement());
```

Snooze/dismiss

Using a selection menu and two buttons, we can create a reminder notification that utilizes the system snooze and dismiss actions. Make sure to set the scenario to "Reminder" for the notification to behave like a reminder.



We link the Snooze button to the selection menu input using the **SelectionBoxId** property on the toast button.

The **Microsoft.Windows.AppNotifications.Builder** syntax does not currently support system activation. But this scenario is supported for Windows App SDK apps, and you can build notifications for this scenario using the `Microsoft.Toolkit.Uwp.Notifications` APIs or raw XML.

Windows App SDK

C#

```
// The Microsoft.Windows.AppNotifications.Builder syntax does not
currently support system activation.
// But this scenario is supported for Windows App SDK apps, and you can
build notifications for this
// scenario using the `Microsoft.Toolkit.Uwp.Notifications` APIs or raw
XML.
```

To use the system snooze and dismiss actions:

- Specify a **ToastButtonSnooze** or **ToastButtonDismiss**
- Optionally specify a custom content string:

- If you don't provide a string, we'll automatically use localized strings for "Snooze" and "Dismiss".
- Optionally specify the **SelectionBoxId**:
- If you don't want the user to select a snooze interval and instead just want your notification to snooze only once for a system-defined time interval (that is consistent across the OS), then don't construct any <input> at all.
- If you want to provide snooze interval selections: - Specify **SelectionBoxId** in the snooze action - Match the id of the input with the **SelectionBoxId** of the snooze action - Specify **ToastSelectionBoxItem**'s value to be a nonNegativeInteger which represents snooze interval in minutes.

Audio

Custom audio has always been supported on Mobile, and is supported in Desktop Version 1511 (build 10586) or later. Custom audio can be referenced via the following paths:

- ms-appx:///
- ms-appdata:///

Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddText("Notification text.")
    .SetAudioUri(new Uri("ms-appx:///Audio/NotificationSound.mp3"));
```

Alternatively, you can pick from the list of ms-winsoundevents, which have always been supported on both platforms.

Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddText("Notification text.")
    .SetAudioEvent(AppNotificationSoundEvent.Alarm,
AppNotificationAudioLooping.Loop);
```

See the audio schema page for information on audio in app notifications. To learn how to send an app notification that uses custom audio, see custom audio on toasts.

Scenarios

To create important notifications, alarms, reminders, and incoming call notifications, you simply use a normal app notification with a **Scenario** value assigned to it. The scenario adjusts a few behaviors to create a consistent and unified user experience. There are four possible **Scenario** values:

- **Reminder**
- **Alarm**
- **IncomingCall**
- **Urgent**

Reminders

In the reminder scenario, the notification will stay on screen until the user dismisses it or takes action. On Windows Mobile, the app notification will also show pre-expanded. A reminder sound will be played. You must provide at least one button on your app notification. Otherwise, the notification will be treated as a normal notification.

Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddText("Notification text.")
    .SetScenario(AppNotificationScenario.Reminder);
```

Alarms

Alarms behave the same as reminders, except alarms will additionally loop audio with a default alarm sound. You must provide at least one button on your app notification. Otherwise, the notification will be treated as a normal notification.

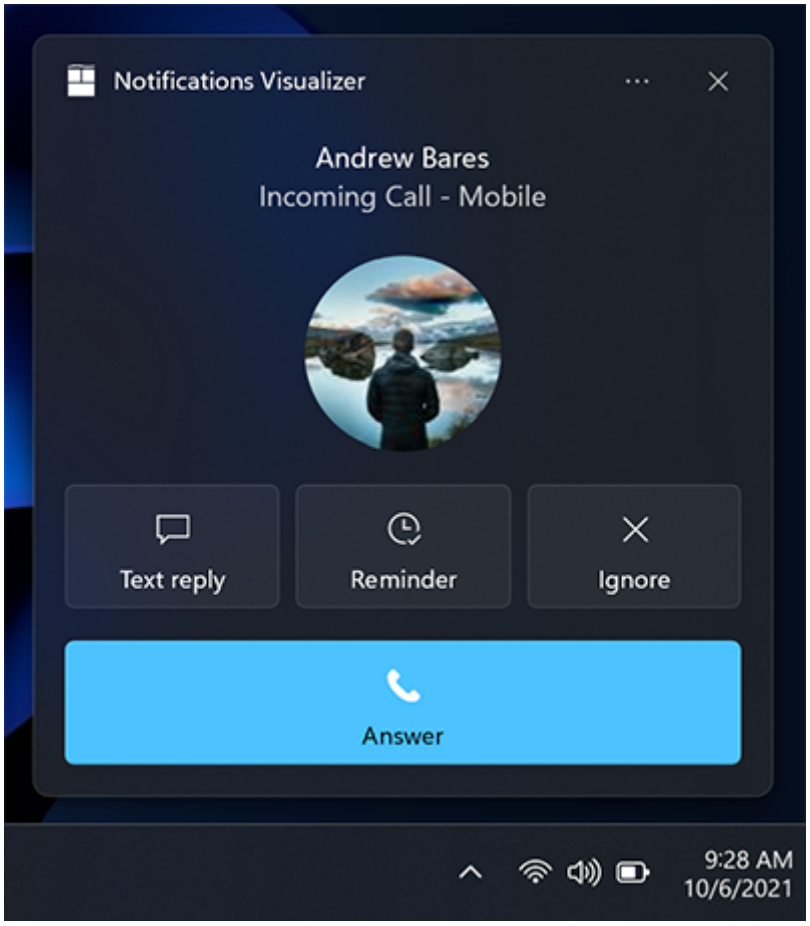
Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddText("Notification text.")
    .SetScenario(AppNotificationScenario.Alarm)
    .AddButton(new AppNotificationButton("Dismiss")
        .AddArgument("action", "dismiss"));
```

Incoming calls

Incoming call notifications are displayed pre-expanded in a special call format and stay on the user's screen till dismissed. Ringtone audio will loop by default. On Windows Mobile devices, they display full screen.



Windows App SDK

C#

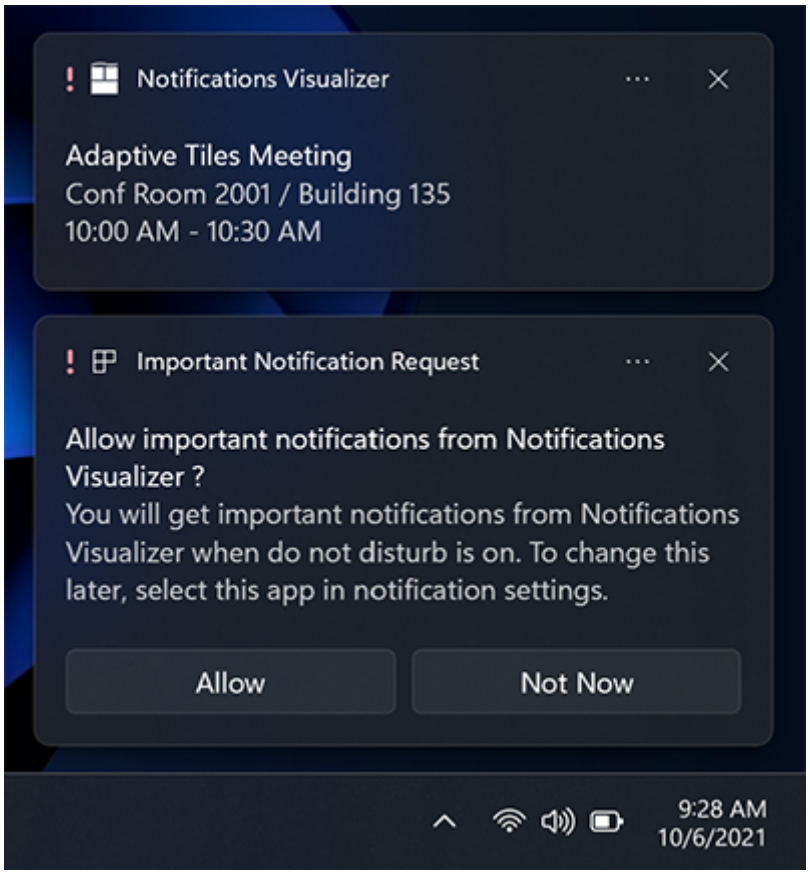
```
var builder = new AppNotificationBuilder()
    .SetScenario(AppNotificationScenario.IncomingCall)
    .AddText("Andrew Bares", new AppNotificationTextProperties()
        .SetIncomingCallAlignment())
    .AddText("incoming call - mobile", new
AppNotificationTextProperties()
        .SetIncomingCallAlignment())
```

```
        .SetInlineImage(new Uri("ms-appx:///images/profile.png"),
          AppNotificationImageCrop.Circle)
    .AddButton(new AppNotificationButton("Text reply")
        .SetToolTip("Text reply")
        .SetIcon(new Uri("ms-appx:///images/reply.png"))
        .AddArgument("textId", "123"))
    .AddButton(new AppNotificationButton("Reminder")
        .SetToolTip("Reminder")
        .SetIcon(new Uri("ms-appx:///images/reminder.png"))
        .AddArgument("reminderId", "123"))
    .AddButton(new AppNotificationButton("Ignore")
        .SetToolTip("Ignore")
        .SetIcon(new Uri("ms-appx:///images/ignore.png"))
        .AddArgument("ignoreId", "123"))
    .AddButton(new AppNotificationButton("Answer")
        .SetToolTip("Answer")
        .SetIcon(new Uri("ms-appx:///images/answer.png"))
        .AddArgument("answerId", "123"));
```

Important Notifications

> ⓘ **Important**
>
> **Requires**: You must be running Windows Insider Preview Build 22546 or later to use important notifications.

Important notifications allow users to have more control over what 1st party and 3rd party apps can send them high-priority app notifications (urgent/important) that can break through Focus Assist (Do not Disturb). This can be modified in the notifications settings.



Windows App SDK

C#

```
var builder = new AppNotificationBuilder()
    .AddText("Adaptive Tiles Meeting",
        new AppNotificationTextProperties()
            .SetMaxLines(1))
    .AddText("Conf Room 2001 / Building 135")
    .AddText("10:00 AM - 10:30 AM");

if (AppNotificationBuilder.IsUrgentScenarioSupported())
{
    builder.SetScenario(AppNotificationScenario.Urgent);
}
```

Localization and accessibility

Your tiles and app notifications can load strings and images tailored for display language, display scale factor, high contrast, and other runtime contexts. For more info, see Tile and toast notification support for language, scale, and high contrast.

Handling activation

To learn how to handle app activations (the user clicking your toast or buttons on the toast), see Send local toast.

Related topics

- Send a local toast and handle activation
- Notifications library on GitHub (part of the UWP Community Toolkit)
- Tile and toast notification support for language, scale, and high contrast

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Toast content schema

Article • 05/24/2022

The following describes all of the properties and elements within toast content.

If you would rather use raw XML instead of the Notifications library, please see the XML schema.

ToastContent

- ToastVisual
  - ToastBindingGeneric
    - IToastBindingGenericChild
    - ToastGenericAppLogo
    - ToastGenericHeroImage
    - ToastGenericAttributionText
- IToastActions
- ToastAudio
- ToastHeader

ToastContent

ToastContent is the top level object that describes a notification's content, including visuals, actions, and audio.

Property	Type	Required	Description
Launch	string	false	A string that is passed to the application when it is activated by the Toast. The format and contents of this string are defined by the app for its own use. When the user taps or clicks the Toast to launch its associated app, the launch string provides the context to the app that allows it to show the user a view relevant to the Toast content, rather than launching in its default way.
Visual	ToastVisual	true	Describes the visual portion of the toast notification.
Actions	IToastActions	false	Optionally create custom actions with buttons and inputs.
Audio	ToastAudio	false	Describes the audio portion of the toast notification.
ActivationType	ToastActivationType	false	Specifies what activation type will be used when the user clicks the body of this Toast.

Property	Type	Required	Description
ActivationOptions	ToastActivationOptions	false	New in Creators Update: Additional options relating to activation of the toast notification.
Scenario	ToastScenario	false	Declares the scenario your toast is used for, like an alarm or reminder.
DisplayTimestamp	DateTimeOffset?	false	New in Creators Update: Override the default timestamp with a custom timestamp representing when your notification content was actually delivered, rather than the time the notification was received by the Windows platform.
Header	ToastHeader	false	New in Creators Update: Add a custom header to your notification to group multiple notifications together within Action Center.

ToastScenario

Specifies what scenario the toast represents.

Value	Meaning
Default	The normal toast behavior.
Reminder	A reminder notification. This will be displayed pre-expanded and stay on the user's screen till dismissed.
Alarm	An alarm notification. This will be displayed pre-expanded and stay on the user's screen till dismissed. Audio will loop by default and will use alarm audio.
IncomingCall	An incoming call notification. This will be displayed pre-expanded in a special call format and stay on the user's screen till dismissed. Audio will loop by default and will use ringtone audio.

ToastVisual

The visual portion of toasts contains the bindings, which contains text, images, adaptive content, and more.

Property	Type	Required	Description
BindingGeneric	ToastBindingGeneric	true	The generic toast binding, which can be rendered on all devices. This binding is required and cannot be null.
BaseUri	Uri	false	A default base URL that is combined with relative URLs in image source attributes.

Property	Type	Required	Description
AddImageQuery	bool?	false	Set to "true" to allow Windows to append a query string to the image URL supplied in the toast notification. Use this attribute if your server hosts images and can handle query strings, either by retrieving an image variant based on the query strings or by ignoring the query string and returning the image as specified without the query string. This query string specifies scale, contrast setting, and language; for instance, a value of "www.website.com/images/hello.png" given in the notification becomes "www.website.com/images/hello.png?ms-scale=100&ms-contrast=standard&ms-lang=en-us"
Language	string	false	The target locale of the visual payload when using localized resources, specified as BCP-47 language tags such as "en-US" or "fr-FR". This locale is overridden by any locale specified in binding or text. If not provided, the system locale will be used instead.

ToastBindingGeneric

The generic binding is the default binding for toasts, and is where you specify the text, images, adaptive content, and more.

Property	Type	Required	Description
Children	IList<IToastBindingGenericChild>	false	The contents of the body of the Toast, which can include text, images, and groups (added in Anniversary Update). Text elements must come before any other elements, and only 3 text elements are supported. If a text element is placed after any other element, it will either be pulled to the top or dropped. And finally, certain text properties like HintStyle aren't supported on the root children text elements, and only work inside an AdaptiveSubgroup. If you use AdaptiveGroup on devices without the Anniversary Update, the group content will simply be dropped.
AppLogoOverride	ToastGenericAppLogo	false	An optional logo to override the app logo.

Property	Type	Required	Description
HeroImage	ToastGenericHeroImage	false	An optional featured "hero" image that is displayed on the toast and within Action Center.
Attribution	ToastGenericAttributionText	false	Optional attribution text which will be displayed at the bottom of the toast notification.
BaseUri	Uri	false	A default base URL that is combined with relative URLs in image source attributes.
AddImageQuery	bool?	false	Set to "true" to allow Windows to append a query string to the image URL supplied in the toast notification. Use this attribute if your server hosts images and can handle query strings, either by retrieving an image variant based on the query strings or by ignoring the query string and returning the image as specified without the query string. This query string specifies scale, contrast setting, and language; for instance, a value of "www.website.com/images/hello.png" given in the notification becomes "www.website.com/images/hello.png?ms-scale=100&ms-contrast=standard&ms-lang=en-us"
Language	string	false	The target locale of the visual payload when using localized resources, specified as BCP-47 language tags such as "en-US" or "fr-FR". This locale is overridden by any locale specified in binding or text. If not provided, the system locale will be used instead.

IToastBindingGenericChild

Marker interface for toast child elements that include text, images, groups, and more.

Implementations
AdaptiveText
AdaptiveImage
AdaptiveGroup
AdaptiveProgressBar

AdaptiveText

An adaptive text element. If placed in the top level ToastBindingGeneric.Children, only HintMaxLines will be applied. But if this is placed as a child of a group/subgroup, full text styling is supported.

Property	Type	Required	Description
Text	string or BindableString	false	The text to display. Data binding support added in Creators Update, but only works for top-level text elements.
HintStyle	AdaptiveTextStyle	false	The style controls the text's font size, weight, and opacity. Only works for text elements inside a group/subgroup.
HintWrap	bool?	false	Set this to true to enable text wrapping. Top-level text elements ignore this property and always wrap (you can use HintMaxLines = 1 to disable wrapping for top-level text elements). Text elements inside groups/subgroups default to false for wrapping.
HintMaxLines	int?	false	The maximum number of lines the text element is allowed to display.
HintMinLines	int?	false	The minimum number of lines the text element must display. Only works for text elements inside a group/subgroup.
HintAlign	AdaptiveTextAlign	false	The horizontal alignment of the text. Only works for text elements inside a group/subgroup.
Language	string	false	The target locale of the XML payload, specified as a BCP-47 language tags such as "en-US" or "fr-FR". The locale specified here overrides any other specified locale, such as that in binding or visual. If this value is a literal string, this attribute defaults to the user's UI language. If this value is a string reference, this attribute defaults to the locale chosen by Windows Runtime in resolving the string.

BindableString

A binding value for strings.

Property	Type	Required	Description
BindingName	string	true	Gets or sets the name that maps to your binding data value.

AdaptiveTextStyle

Text style controls font size, weight, and opacity. Subtle opacity is 60% opaque.

Value	Meaning
Default	Default value. Style is determined by the renderer.
Caption	Smaller than paragraph font size.
CaptionSubtle	Same as Caption but with subtle opacity.
Body	Paragraph font size.
BodySubtle	Same as Body but with subtle opacity.
Base	Paragraph font size, bold weight. Essentially the bold version of Body.
BaseSubtle	Same as Base but with subtle opacity.
Subtitle	H4 font size.
SubtitleSubtle	Same as Subtitle but with subtle opacity.
Title	H3 font size.
TitleSubtle	Same as Title but with subtle opacity.
TitleNumeral	Same as Title but with top/bottom padding removed.
Subheader	H2 font size.
SubheaderSubtle	Same as Subheader but with subtle opacity.
SubheaderNumeral	Same as Subheader but with top/bottom padding removed.
Header	H1 font size.
HeaderSubtle	Same as Header but with subtle opacity.
HeaderNumeral	Same as Header but with top/bottom padding removed.

AdaptiveTextAlign

Controls the horizontal alignmen of text.

Value	Meaning
Default	Default value. Alignment is automatically determined by the renderer.
Auto	Alignment determined by the current language and culture.
Left	Horizontally align the text to the left.
Center	Horizontally align the text in the center.
Right	Horizontally align the text to the right.

AdaptiveImage

An inline image.

Property	Type	Required	Description
Source	string	true	The URL to the image. ms-appx, ms-appdata, and http are supported. As of the Fall Creators Update, web images can be up to 3 MB on normal connections and 1 MB on metered connections. On devices not yet running the Fall Creators Update, web images must be no larger than 200 KB.
HintCrop	AdaptiveImageCrop	false	New in Anniversary Update: Control the desired cropping of the image.
HintRemoveMargin	bool?	false	By default, images inside groups/subgroups have an 8px margin around them. You can remove this margin by setting this property to true.
HintAlign	AdaptiveImageAlign	false	The horizontal alignment of the image. Only works for images inside a group/subgroup.
AlternateText	string	false	Alternate text describing the image, used for accessibility purposes.
AddImageQuery	bool?	false	Set to "true" to allow Windows to append a query string to the image URL supplied in the toast notification. Use this attribute if your server hosts images and can handle query strings, either by retrieving an image variant based on the query strings or by ignoring the query string and returning the image as specified without the query string. This query string specifies scale, contrast setting, and language; for instance, a value of "www.website.com/images/hello.png" given in the notification becomes "www.website.com/images/hello.png?ms-scale=100&ms-contrast=standard&ms-lang=en-us"

AdaptiveImageCrop

Specifies the desired cropping of the image.

Value	Meaning
Default	Default value. Cropping behavior determined by renderer.
None	Image is not cropped.

Value	Meaning
Circle	Image is cropped to a circle shape.

AdaptiveImageAlign

Specifies the horizontal alignment for an image.

Value	Meaning
Default	Default value. Alignment behavior determined by renderer.
Stretch	Image stretches to fill available width (and potentially available height too, depending on where the image is placed).
Left	Align the image to the left, displaying the image at its native resolution.
Center	Align the image in the center horizontally, displayign the image at its native resolution.
Right	Align the image to the right, displaying the image at its native resolution.

AdaptiveGroup

New in Anniversary Update: Groups semantically identify that the content in the group must either be displayed as a whole, or not displayed if it cannot fit. Groups also allow creating multiple columns.

Property	Type	Required	Description
Children	IList<AdaptiveSubgroup>	false	Subgroups are displayed as vertical columns. You must use subgroups to provide any content inside an AdaptiveGroup.

AdaptiveSubgroup

New in Anniversary Update: Subgroups are vertical columns that can contain text and images.

Property	Type	Required	Description
Children	IList<IAdaptiveSubgroupChild>	false	AdaptiveText and AdaptiveImage are valid children of subgroups.
HintWeight	int?	false	Control the width of this subgroup column by specifying the weight, relative to the other subgroups.
HintTextStacking	AdaptiveSubgroupTextStacking	false	Control the vertical alignment of this subgroup's content.

IAdaptiveSubgroupChild

Marker interface for subgroup children.

Implementations
AdaptiveText
AdaptiveImage

AdaptiveSubgroupTextStacking

TextStacking specifies the vertical alignment of content.

Value	Meaning
Default	Default value. Renderer automatically selects the default vertical alignment.
Top	Vertical align to the top.
Center	Vertical align to the center.
Bottom	Vertical align to the bottom.

AdaptiveProgressBar

New in Creators Update: A progress bar. Only supported on toasts on Desktop, build 15063 or later.

Property	Type	Required	Description
Title	string or BindableString	false	Gets or sets an optional title string. Supports data binding.
Value	double or AdaptiveProgressBarValue or BindableProgressBarValue	false	Gets or sets the value of the progress bar. Supports data binding. Defaults to 0.
ValueStringOverride	string or BindableString	false	Gets or sets an optional string to be displayed instead of the default percentage string. If this isn't provided, something like "70%" will be displayed.
Status	string or BindableString	true	Gets or sets a status string (required), which is displayed underneath the progress bar on the left. This string should reflect the status of the operation, like "Downloading..." or "Installing..."

AdaptiveProgressBarValue

A class that represents the progress bar's value.

Property	Type	Required	Description
Value	double	false	Gets or sets the value (0.0 - 1.0) representing the percent complete.
IsIndeterminate	bool	false	Gets or sets a value indicating whether the progress bar is indeterminate. If this is true, **Value** will be ignored.

BindableProgressBarValue

A bindable progress bar value.

Property	Type	Required	Description
BindingName	string	true	Gets or sets the name that maps to your binding data value.

ToastGenericAppLogo

A logo to be displayed instead of the app logo.

Property	Type	Required	Description
Source	string	true	The URL to the image. ms-appx, ms-appdata, and http are supported. Http images must be 200 KB or less in size.
HintCrop	ToastGenericAppLogoCrop	false	Specify how you would like the image to be cropped.
AlternateText	string	false	Alternate text describing the image, used for accessibility purposes.

Property	Type	Required	Description
AddImageQuery	bool?	false	Set to "true" to allow Windows to append a query string to the image URL supplied in the toast notification. Use this attribute if your server hosts images and can handle query strings, either by retrieving an image variant based on the query strings or by ignoring the query string and returning the image as specified without the query string. This query string specifies scale, contrast setting, and language; for instance, a value of "www.website.com/images/hello.png" given in the notification becomes "www.website.com/images/hello.png?ms-scale=100&ms-contrast=standard&ms-lang=en-us"

ToastGenericAppLogoCrop

Controls the cropping of the app logo image.

Value	Meaning
Default	Cropping uses the default behavior of the renderer.
None	Image is not cropped, displayed square.
Circle	Image is cropped to a circle.

ToastGenericHeroImage

A featured "hero" image that is displayed on the toast and within Action Center.

Property	Type	Required	Description
Source	string	true	The URL to the image. ms-appx, ms-appdata, and http are supported. Http images must be 200 KB or less in size.
AlternateText	string	false	Alternate text describing the image, used for accessibility purposes.
AddImageQuery	bool?	false	Set to "true" to allow Windows to append a query string to the image URL supplied in the toast notification. Use this attribute if your server hosts images and can handle query strings, either by retrieving an image variant based on the query strings or by ignoring the query string and returning the image as specified without the query string. This query string specifies scale, contrast setting, and language; for instance, a value of "www.website.com/images/hello.png" given in the notification becomes "www.website.com/images/hello.png?ms-scale=100&ms-contrast=standard&ms-lang=en-us"

ToastGenericAttributionText

Attribution text displayed at the bottom of the toast notification.

Property	Type	Required	Description
Text	string	true	The text to display.
Language	string	false	The target locale of the visual payload when using localized resources, specified as BCP-47 language tags such as "en-US" or "fr-FR". If not provided, the system locale will be used instead.

IToastActions

Marker interface for toast actions/inputs.

Implementations
ToastActionsCustom
ToastActionsSnoozeAndDismiss

ToastActionsCustom

Implements IToastActions

Create your own custom actions and inputs, using controls like buttons, text boxes, and selection inputs.

Property	Type	Required	Description
Inputs	IList<IToastInput>	false	Inputs like text boxes and selection inputs. Only up to 5 inputs are allowed.
Buttons	IList<IToastButton>	false	Buttons are displayed after all the inputs (or adjacent to an input if the button is used as a quick reply button). Only up to 5 buttons are allowed (or fewer if you also have context menu items).
ContextMenuItems	IList<ToastContextMenuItem>	false	New in Anniversary Update: Custom context menu items, providing additional actions if the user right clicks the notification. You can only have up to 5 buttons and context menu items *combined*.

IToastInput

Marker interface for toast inputs.

Implementations
ToastTextBox
ToastSelectionBox

ToastTextBox

Implements IToastInput

A text box control that the user can type text into.

Property	Type	Required	Description
Id	string	true	The Id is required, and is used to map the user-inputted text into a key-value pair of id/value which your app later consumes.
Title	string	false	Title text to display above the text box.
PlaceholderContent	string	false	Placeholder text to be displayed on the text box when the user hasn't typed any text yet.
DefaultInput	string	false	The initial text to place in the text box. Leave this null for a blank text box.

ToastSelectionBox

Implements IToastInput

A selection box control, which lets users pick from a dropdown list of options.

Property	Type	Required	Description
Id	string	true	The Id is required. If the user selected this item, this Id will be passed back to your app's code, representing which selection they chose.
Content	string	true	Content is required, and is a string that is displayed on the selection item.

ToastSelectionBoxItem

A selection box item (an item that the user can select from the drop down list).

Property	Type	Required	Description

Property	Type	Required	Description
Id	string	true	The Id is required, and is used to map the user-inputted text into a key-value pair of id/value which your app later consumes.
Title	string	false	Title text to display above the selection box.
DefaultSelectionBoxItemId	string	false	This controls which item is selected by default, and refers to the Id property of the ToastSelectionBoxItem. If you do not provide this, the default selection will be empty (user sees nothing).
Items	IList<ToastSelectionBoxItem>	false	The selection items that the user can pick from in this SelectionBox. Only 5 items can be added.

IToastButton

Marker interface for toast buttons.

Implementations
ToastButton
ToastButtonSnooze
ToastButtonDismiss

ToastButton

Implements IToastButton

A button that the user can click.

Property	Type	Required	Description
Content	string	true	Required. The text to display on the button.
Arguments	string	true	Required. App-defined string of arguments that the app will later receive if the user clicks this button.
ActivationType	ToastActivationType	false	Controls what type of activation this button will use when clicked. Defaults to Foreground.

Property	Type	Required	Description
ActivationOptions	ToastActivationOptions	false	New in Creators Update: Gets or sets additional options relating to activation of the toast button.

ToastActivationType

Decides the type of activation that will be used when the user interacts with a specific action.

Value	Meaning
Foreground	Default value. Your foreground app is launched.
Background	Your corresponding background task (assuming you set everything up) is triggered, and you can execute code in the background (like sending the user's quick reply message) without interrupting the user.
Protocol	Launch a different app using protocol activation.

ToastActivationOptions

New in Creators Update: Additional options relating to activation.

Property	Type	Required	Description
AfterActivationBehavior	ToastAfterActivationBehavior	false	New in Fall Creators Update: Gets or sets the behavior that the toast should use when the user invokes this action. This only works on Desktop, for ToastButton and ToastContextMenuItem.
ProtocolActivationTargetApplicationPfn	string	false	If you are using *ToastActivationType.Protocol*, you can optionally specify the target PFN, so that regardless of whether multiple apps are registered to handle the same protocol uri, your desired app will always be launched.

ToastAfterActivationBehavior

Specifies the behavior that the toast should use when the user takes action on the toast.

Value	Meaning

Value	Meaning
Default	Default behavior. The toast will be dismissed when the user takes action on the toast.
PendingUpdate	After the user clicks a button on your toast, the notification will remain present, in a "pending update" visual state. You should immediately update your toast from a background task so that the user does not see this "pending update" visual state for too long.

ToastButtonSnooze

Implements IToastButton

A system-handled snooze button that automatically handles snoozing of the notification.

Property	Type	Required	Description
CustomContent	string	false	Optional custom text displayed on the button that overrides the default localized "Snooze" text.

ToastButtonDismiss

Implements IToastButton

A system-handled dismiss button that dismisses the notification when clicked.

Property	Type	Required	Description
CustomContent	string	false	Optional custom text displayed on the button that overrides the default localized "Dismiss" text.

ToastActionsSnoozeAndDismiss

*Implements IToastActions

Automatically constructs a selection box for snooze intervals, and snooze/dismiss buttons, all automatically localized, and snoozing logic is automatically handled by the system.

Property	Type	Required	Description
ContextMenuItems	IList<ToastContextMenuItem>	false	New in Anniversary Update: Custom context menu items, providing additional actions if the user right clicks the notification. You can only have up to 5 items.

ToastContextMenuItem

A context menu item entry.

Property	Type	Required	Description
Content	string	true	Required. The text to display.
Arguments	string	true	Required. App-defined string of arguments that the app can later retrieve once it is activated when the user clicks the menu item.
ActivationType	ToastActivationType	false	Controls what type of activation this menu item will use when clicked. Defaults to Foreground.
ActivationOptions	ToastActivationOptions	false	New in Creators Update: Additional options relating to activation of the toast context menu item.

ToastAudio

Specify audio to be played when the Toast notification is received.

Property	Type	Required	Description
Src	uri	false	The media file to play in place of the default sound. Only ms-appx and ms-resource are supported. All else (ms-appdata, http, C:, etc.) is not supported.
Loop	boolean	false	Set to true if the sound should repeat as long as the Toast is shown; false to play only once (default).
Silent	boolean	false	True to mute the sound; false to allow the toast notification sound to play (default).

ToastHeader

New in Creators Update: A custom header that groups multiple notifications together within Action Center.

Property	Type	Required	Description
Id	string	true	A developer-created identifier that uniquely identifies this header. If two notifications have the same header id, they will be displayed underneath the same header in Action Center.
Title	string	true	A title for the header.

Property	Type	Required	Description
Arguments	string	true	Gets or sets a developer-defined string of arguments that is returned to the app when the user clicks this header. Cannot be null.
ActivationType	ToastActivationType	false	Gets or sets the type of activation this header will use when clicked. Defaults to Foreground. Note that only Foreground and Protocol are supported.
ActivationOptions	ToastActivationOptions	false	Gets or sets additional options relating to activation of the toast header.

Related topics

- Quickstart: Send a local toast and handle activation
- Notifications library on GitHub

Schedule a toast notification

Article • 10/06/2022

Scheduled toast notifications allow you to schedule a notification to appear at a later time, regardless of whether your app is running at that time. This is useful for scenarios like displaying reminders or other follow-up tasks for the user, where the time and content of the notification is known ahead-of-time.

Note that scheduled toast notifications have a delivery window of 5 minutes. If the computer is turned off during the scheduled delivery time, and remains off for longer than 5 minutes, the notification will be "dropped" as no longer relevant to the user. If you need guaranteed delivery of notifications regardless of how long the computer was off, we recommend using a background task with a time trigger, as illustrated in this code sample.

> **Important**
>
> Desktop applications (both packaged and unpackaged) have slightly different steps for sending notifications and handling activation. Follow along with the instructions below, however replace `ToastNotificationManager` with the `DesktopNotificationManagerCompat` class from the **desktop apps** documentation.

> **Important APIs**: ScheduledToastNotification Class

Prerequisites

To fully understand this topic, the following will be helpful...

- A working knowledge of toast notification terms and concepts. For more information, see Toast and action center overview.
- A familiarity with Windows 10 toast notification content. For more information, see toast content documentation.
- A Windows 10 UWP app project

Step 1: Install NuGet package

Install the Microsoft.Toolkit.Uwp.Notifications NuGet package. Our code sample will use this package. At the end of the article we'll provide the "plain" code snippets that

don't use any NuGet packages. This package allows you to create toast notifications without using XML.

Step 2: Add namespace declarations

```
C#

using Microsoft.Toolkit.Uwp.Notifications; // Notifications library
```

Step 3: Schedule the notification

We'll use a simple text-based notification reminding a student about the homework they have due today. Construct the notification and schedule it!

```
C#

// Construct the content and schedule the toast!
new ToastContentBuilder()
    .AddArgument("action", "viewItemsDueToday")
    .AddText("ASTR 170B1")
    .AddText("You have 3 items due today!");
    .Schedule(DateTime.Now.AddSeconds(5));
```

Provide a primary key for your toast

If you want to programmatically cancel, remove, or replace the scheduled notification, you need to use the Tag property (and optionally the Group property) to provide a primary key for your notification. Then, you can use this primary key in the future to cancel, remove, or replace the notification.

To see more details on replacing/removing already delivered toast notifications, please see Quickstart: Managing toast notifications in action center (XAML).

Tag and Group combined act as a composite primary key. Group is the more generic identifier, where you can assign groups like "wallPosts", "messages", "friendRequests", etc. And then Tag should uniquely identify the notification itself from within the group. By using a generic group, you can then remove all notifications from that group by using the RemoveGroup API.

```
C#
```

```
// Construct the content and schedule the toast!
new ToastContentBuilder()
    .AddArgument("action", "viewItemsDueToday")
    .AddText("ASTR 170B1")
    .AddText("You have 3 items due today!");
    .Schedule(DateTime.Now.AddSeconds(5), toast =>
    {
        toast.Tag = "18365";
        toast.Group = "ASTR 170B1";
    });
```

Cancel scheduled notifications

To cancel a scheduled notification, you first have to retrieve the list of all scheduled notifications.

Then, find your scheduled toast matching the tag (and optionally group) you specified earlier, and call RemoveFromSchedule().

C#

```
// Create the toast notifier
ToastNotifierCompat notifier =
ToastNotificationManagerCompat.CreateToastNotifier();

// Get the list of scheduled toasts that haven't appeared yet
IReadOnlyList<ScheduledToastNotification> scheduledToasts =
notifier.GetScheduledToastNotifications();

// Find our scheduled toast we want to cancel
var toRemove = scheduledToasts.FirstOrDefault(i => i.Tag == "18365" &&
i.Group == "ASTR 170B1");
if (toRemove != null)
{
    // And remove it from the schedule
    notifier.RemoveFromSchedule(toRemove);
}
```

> **ⓘ Important**
>
> An unpackaged (lacks package identity at runtime) Win32 app must use the **ToastNotificationManagerCompat** class as seen above. If you use **ToastNotificationManager** itself, then you'll receive an element-not-found exception. All types of apps can use the **Compat** class, and it will work correctly.

Activation handling

See the send a local toast docs to learn more about handling activation. Activation of a scheduled toast notification is handled the same as activation of a local toast notification.

Adding actions, inputs, and more

See the send a local toast docs to learn more about advanced topics like actions and inputs. Actions and inputs work the same in local toasts as they do in scheduled toasts.

Resources

- Toast content documentation
- ScheduledToastNotification Class

Custom audio on toasts

Article • 10/19/2021

Toast notifications can use custom audio, which lets your app express your brand's unique sound effects. For example, a messaging app can use their own messaging sound on their Toast notifications, so that the user can instantly know that they received a notification from the app, rather than hearing the generic notification sound.

Install UWP Community Toolkit NuGet package

In order to create notifications via code, we strongly recommend using the UWP Community Toolkit Notifications library, which provides an object model for the notification XML content. You could manually construct the notification XML, but that is error-prone and messy. The Notifications library inside UWP Community Toolkit is built and maintained by the team that owns notifications at Microsoft.

Install Microsoft.Toolkit.Uwp.Notifications from NuGet.

Add namespace declarations

```
C#

using Microsoft.Toolkit.Uwp.Notifications;
```

Add the custom audio

Windows Mobile has always supported custom audio in Toast notifications. However, Desktop only added support for custom audio in Version 1511 (build 10586). If you send a Toast that contains custom audio to a Desktop device before Version 1511, the toast will be silent. Therefore, for Desktop pre-Version 1511, you should NOT include the custom audio in your Toast notification, so that the notification will at least use the default notification sound.

Known Issue: If you're using Desktop Version 1511, the custom toast audio will only work if your app is installed via the Store. That means you cannot locally test your custom audio on Desktop before submitting to the Store - but the audio will work fine once installed from the Store. We fixed this in the Anniversary Update, so that custom audio from your locally deployed app will work correctly.

```
```

C#

```
var contentBuilder = new ToastContentBuilder()
    .AddText("New message");


bool supportsCustomAudio = true;

// If we're running on Desktop before Version 1511, do NOT include custom
audio
// since it was not supported until Version 1511, and would result in a
silent toast.
if (AnalyticsInfo.VersionInfo.DeviceFamily.Equals("Windows.Desktop")
    &&
!ApiInformation.IsApiContractPresent("Windows.Foundation.UniversalApiContrac
t", 2))
{
    supportsCustomAudio = false;
}

if (supportsCustomAudio)
{
    contentBuilder.AddAudio(new Uri("ms-
appx:///Assets/Audio/CustomToastAudio.m4a"));
}

// Send the toast
contentBuilder.Show();
```

Supported audio file types include:

- .aac
- .flac
- .m4a
- .mp3
- .wav
- .wma

Supported audio file sources:

- ms-appx:///
- ms-resource

Not supported audio file sources:

- ms-appdata
- http://, https://
- C:/, F:/, etc.

Send the notification

Sending a notification with audio is the same as sending a regular notification (just call the Show method). See Send local toast to learn more.

Related topics

- Full code sample on GitHub
- Send a local toast
- Toast content documentation

Toast progress bar and data binding

Article • 10/27/2022

Using a progress bar inside your toast notification allows you to convey the status of long-running operations to the user, like downloads, video rendering, exercise goals, and more.

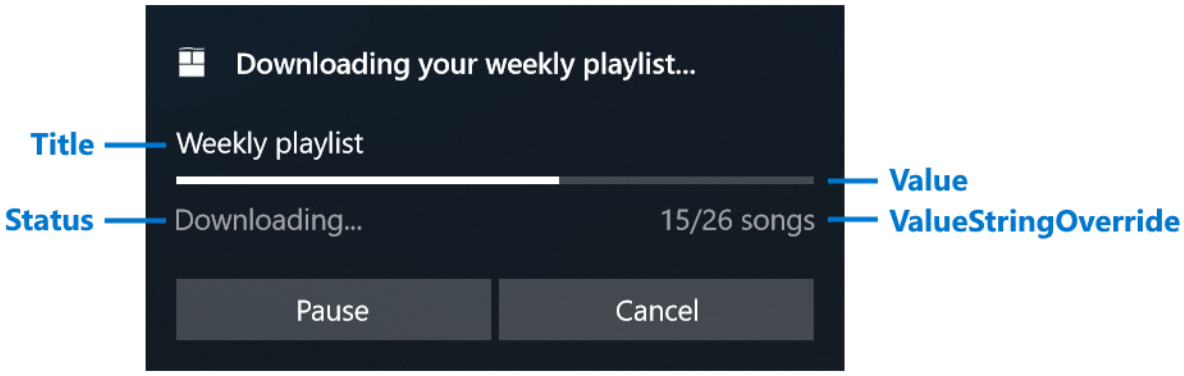
> **Important**
>
> **Requires Creators Update and 1.4.0 of Notifications library**: You must target SDK 15063 and be running build 15063 or later to use progress bars on toasts. You must use version 1.4.0 or later of the **UWP Community Toolkit Notifications NuGet library** to construct the progress bar in your toast's content.

A progress bar inside a toast can either be "indeterminate" (no specific value, animated dots indicate an operation is occurring) or "determinate" (a specific percent of the bar is filled, like 60%).

> **Important APIs**: NotificationData class, ToastNotifier.Update method, ToastNotification class

> **Note**
>
> Only Desktop supports progress bars in toast notifications. On other devices, the progress bar will be dropped from your notification.

The picture below shows a determinate progress bar with all of its corresponding properties labeled.



Property	Type	Required	Description
Title	string or BindableString	false	Gets or sets an optional title string. Supports data binding.

Property	Type	Required	Description
Value	double or AdaptiveProgressBarValue or BindableProgressBarValue	false	Gets or sets the value of the progress bar. Supports data binding. Defaults to 0. Can either be a double between 0.0 and 1.0, `AdaptiveProgressBarValue.Indeterminate`, or `new BindableProgressBarValue("myProgressValue")`.
ValueStringOverride	string or BindableString	false	Gets or sets an optional string to be displayed instead of the default percentage string. If this isn't provided, something like "70%" will be displayed.
Status	string or BindableString	true	Gets or sets a status string (required), which is displayed underneath the progress bar on the left. This string should reflect the status of the operation, like "Downloading..." or "Installing..."

Here's how you would generate the notification seen above...

Builder syntax

C#

```
new ToastContentBuilder()
    .AddText("Downloading your weekly playlist...")
    .AddVisualChild(new AdaptiveProgressBar()
    {
        Title = "Weekly playlist",
        Value = 0.6,
        ValueStringOverride = "15/26 songs",
        Status = "Downloading..."
    });
```

However, you'll need to dynamically update the values of the progress bar for it to actually be "live". This can be done by using data binding to update the toast.

Using data binding to update a toast

Using data binding involves the following steps...

1. Construct toast content that utilizes data bound fields
2. Assign a **Tag** (and optionally a **Group**) to your **ToastNotification**
3. Define your initial **Data** values on your **ToastNotification**
4. Send the toast

5. Utilize **Tag** and **Group** to update the **Data** values with new values

The following code snippet shows steps 1-4. The next snippet will show how to update the toast **Data** values.

C#

```
using Windows.UI.Notifications;
using Microsoft.Toolkit.Uwp.Notifications;

public void SendUpdatableToastWithProgress()
{
    // Define a tag (and optionally a group) to uniquely identify the
notification, in order update the notification data later;
    string tag = "weekly-playlist";
    string group = "downloads";

    // Construct the toast content with data bound fields
    var content = new ToastContentBuilder()
        .AddText("Downloading your weekly playlist...")
        .AddVisualChild(new AdaptiveProgressBar()
        {
            Title = "Weekly playlist",
            Value = new BindableProgressBarValue("progressValue"),
            ValueStringOverride = new BindableString("progressValueString"),
            Status = new BindableString("progressStatus")
        })
        .GetToastContent();

    // Generate the toast notification
    var toast = new ToastNotification(content.GetXml());

    // Assign the tag and group
    toast.Tag = tag;
    toast.Group = group;

    // Assign initial NotificationData values
    // Values must be of type string
    toast.Data = new NotificationData();
    toast.Data.Values["progressValue"] = "0.6";
    toast.Data.Values["progressValueString"] = "15/26 songs";
    toast.Data.Values["progressStatus"] = "Downloading...";

    // Provide sequence number to prevent out-of-order updates, or assign 0 to
indicate "always update"
    toast.Data.SequenceNumber = 1;

    // Show the toast notification to the user
    ToastNotificationManager.CreateToastNotifier().Show(toast);
}
```

Then, when you want to change your **Data** values, use the **Update** method to provide the new data without re-constructing the entire toast payload.

C#

```
using Windows.UI.Notifications;

public void UpdateProgress()
{
    // Construct a NotificationData object;
    string tag = "weekly-playlist";
    string group = "downloads";

    // Create NotificationData and make sure the sequence number is incremented
    // since last update, or assign 0 for updating regardless of order
    var data = new NotificationData
    {
        SequenceNumber = 2
    };

    // Assign new values
    // Note that you only need to assign values that changed. In this example
    // we don't assign progressStatus since we don't need to change it
    data.Values["progressValue"] = "0.7";
    data.Values["progressValueString"] = "18/26 songs";

    // Update the existing notification's data by using tag/group
    ToastNotificationManager.CreateToastNotifier().Update(data, tag, group);
}
```

Using the **Update** method rather than replacing the entire toast also ensures that the toast notification stays in the same position in Action Center and doesn't move up or down. It would be quite confusing to the user if the toast kept jumping to the top of Action Center every few seconds while the progress bar filled!

The **Update** method returns an enum, **NotificationUpdateResult**, which lets you know whether the update succeeded or whether the notification couldn't be found (which means the user has likely dismissed your notification and you should stop sending updates to it). We do not recommend popping another toast until your progress operation has been completed (like when the download completes).

Elements that support data binding

The following elements in toast notifications support data binding

- All properties on **AdaptiveProgress**
- The **Text** property on the top-level **AdaptiveText** elements

Update or replace a notification

Since Windows 10, you could always **replace** a notification by sending a new toast with the same **Tag** and **Group**. So what's the difference between **replacing** the toast and **updating** the toast's data?

	Replacing	**Updating**
Position in Action Center	Moves the notification to the top of Action Center.	Leaves the notification in place within Action Center.
Modifying content	Can completely change all content/layout of the toast	Can only change properties that support data binding (progress bar and top-level text)
Reappearing as popup	Can reappear as a toast popup if you leave **SuppressPopup** set to `false` (or set to true to silently send it to Action Center)	Won't reappear as a popup; the toast's data is silently updated within Action Center
User dismissed	Regardless of whether user dismissed your previous notification, your replacement toast will always be sent	If the user dismissed your toast, the toast update will fail

In general, **updating is useful for...**

- Information that frequently changes in a short period of time and doesn't require being brought to the front of the user's attention
- Subtle changes to your toast content, like changing 50% to 65%

Often times, after your sequence of updates have completed (like the file has been downloaded), we recommend replacing for the final step, because...

- Your final notification likely has drastic layout changes, like removal of the progress bar, addition of new buttons, etc
- The user might have dismissed your pending progress notification since they don't care about watching it download, but still want to be notified with a popup toast when the operation is completed

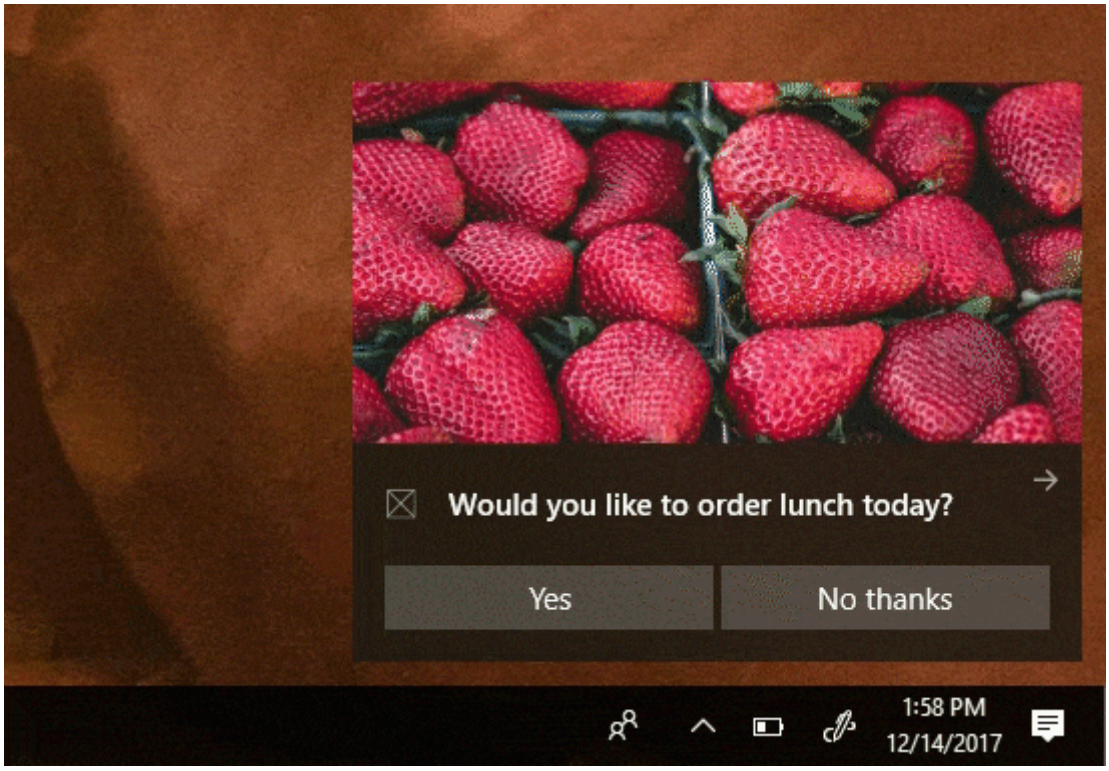
Related topics

- Full code sample on GitHub
- Toast content documentation

Toast with pending update activation

Article • 10/27/2022

You can use **PendingUpdate** to create multi-step interactions in your toast notifications. For example, as seen below, you can create a series of toasts where the subsequent toasts depend on responses from the previous toasts.



> **Important**
>
> **Requires Desktop Fall Creators Update and 2.0.0 of Notifications library**: You must be running Desktop build 16299 or later to see pending update work. You must use version 2.0.0 or later of the **UWP Community Toolkit Notifications NuGet library** to assign **PendingUpdate** on your buttons. **PendingUpdate** is only supported on Desktop and will be ignored on other devices.

Prerequisites

This article assumes a working knowledge of...

- Constructing toast content
- Sending a toast and handling background activation

Overview

To implement a toast that uses pending update as its after activation behavior...

1. On your toast background activation buttons, specify an **AfterActivationBehavior** of **PendingUpdate**
2. Assign a **Tag** (and optionally **Group**) when sending your toast
3. When the user clicks your button, your background task will be activated, and the toast will be kept on-screen in a pending update state
4. In your background task, send a new toast with your new content, using the same **Tag** and **Group**

Assign PendingUpdate

On your background activation buttons, set the **AfterActivationBehavior** to **PendingUpdate**. Note that this only works for buttons that have an **ActivationType** of **Background**.

Builder syntax

C#

```
new ToastContentBuilder()

    .AddText("Would you like to order lunch today?")

    .AddButton(new ToastButton("Yes", "action=orderLunch")
    {
        ActivationType = ToastActivationType.Background,

        ActivationOptions = new ToastActivationOptions()
        {
            AfterActivationBehavior =
ToastAfterActivationBehavior.PendingUpdate
        }
    });
```

Use a Tag on the notification

In order to later replace the notification, we have to assign the **Tag** (and optionally the **Group**) on the notification.

C#

```
// Create the notification
var notif = new ToastNotification(content.GetXml())
{
    Tag = "lunch"
};

// And show it
ToastNotificationManager.CreateToastNotifier().Show(notif);
```

Replace the toast with new content

In response to the user clicking your button, your background task gets triggered and you need to replace the toast with new content. You replace the toast by simply sending a new toast with the same **Tag** and **Group**.

We strongly recommend **setting the audio to silent** on replacements in response to a button click, since the user is already interacting with your toast.

C#

```
// Generate new content
ToastContent content = new ToastContent()
{
    ...

    // We disable audio on all subsequent toasts since they appear right
after the user
    // clicked something, so the user's attention is already captured
    Audio = new ToastAudio() { Silent = true }
};

// Create the new notification
var notif = new ToastNotification(content.GetXml())
{
    Tag = "lunch"
};

// And replace the old one with this one
ToastNotificationManager.CreateToastNotifier().Show(notif);
```

Related topics

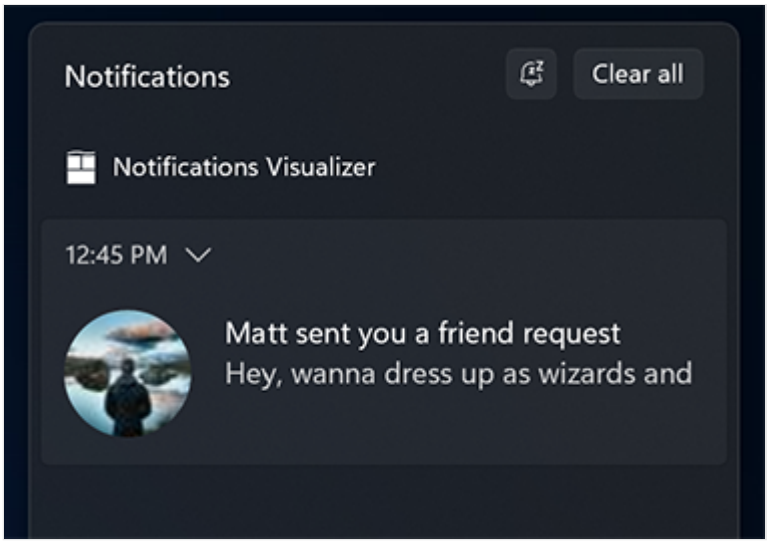
- Full code sample on GitHub
- Send a local toast and handle activation
- Toast content documentation
- Toast progress bar

Custom timestamps on app notifications

Article • 12/13/2022

By default, the timestamp on app notifications, which is visible within Notification Center, is set to the time that the notification was sent. You can optionally override the timestamp with your own custom date and time, so that the timestamp represents the time the message/information/content was actually created, rather than the time that the notification was sent. This also ensures that your notifications appear in the correct order within Notification Center, which is sorted by time. We recommend that most apps specify a custom timestamp.

This feature is available in Windows Build 15063 and later.



> **Note**
>
> The term "toast notification" is being replaced with "app notification". These terms both refer to the same feature of Windows, but over time we will phase out the use of "toast notification" in the documentation.

To use a custom timestamp, simply assign the **displayTimestamp** property on the **toast** element of your app notification XML payload. Starting with Windows App SDK 1.2, you can add a custom timestamp to an app notification with the Microsoft.Windows.AppNotifications.Builder. For UWP apps, you can use version 1.4.0 or later of the UWP Community Toolkit Notifications NuGet library. YOu an also specify the timestamp using raw xml.

Windows App SDK

```
var builder = new AppNotificationBuilder()
    .AddText("Matt sent you a friend request")
    .AddText("Hey, wanna dress up as wizards and ride around on
hoverboards?")
    .SetTimeStamp(new DateTime(2017, 04, 15, 19, 45, 00,
DateTimeKind.Utc));
```

If you are using XML, the date must be formatted in ISO 8601.

> **Note**
>
> You can only use at most 3 decimal places on the seconds (although realistically there's no value in providing anything that granular). If you provide more, the payload will be invalid and you will receive the "New notification" notification.

Usage guidance

In general, we recommend that most apps specify a custom timestamp. This ensures that the notification's timestamp accurately represents when the message/information/content was generated, regardless of network delays, airplane mode, or the fixed interval of periodic background tasks.

For example, a news app might run a background task every 15 minutes that checks for new articles and displays notifications. Before custom timestamps, the timestamp corresponded to when the app notification was generated (therefore always in 15 minute intervals). However, now the app can set the timestamp to the time the article was actually published. Similarly, email apps and social network apps can benefit from this feature if a similar pattern of periodic pulling is used for their notifications.

Additionally, providing a custom timestamp ensures that the timestamp is correct even if the user was disconnected from the internet. For example, when the user turns their computer on and your background task runs, you can finally ensure that the timestamp on your notifications represents the time that the messages were sent, rather than the time the user turned on their computer.

Default timestamp

If you don't provide a custom timestamp, we use the time that your notification was sent.

If you sent a push notification through WNS, we use the time when the notification was received by WNS server (so any latency on delivering the notification to the device won't impact the timestamp).

If you sent a local notification, we use the time when the notification platform received the notification (which should be immediately).

Related topics

- Send a local toast
- App notification content documentation

Grouping toast notifications with collections

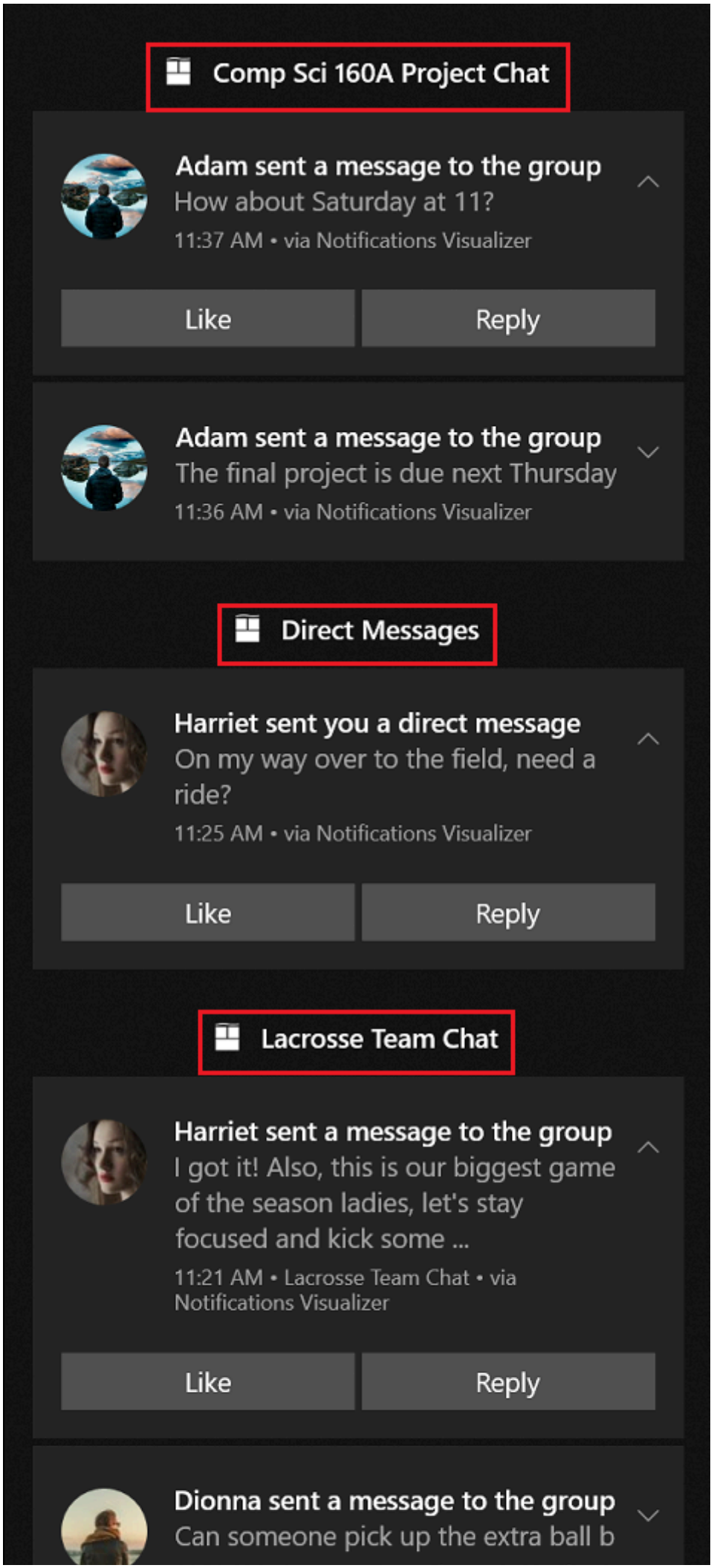
Article • 10/27/2022

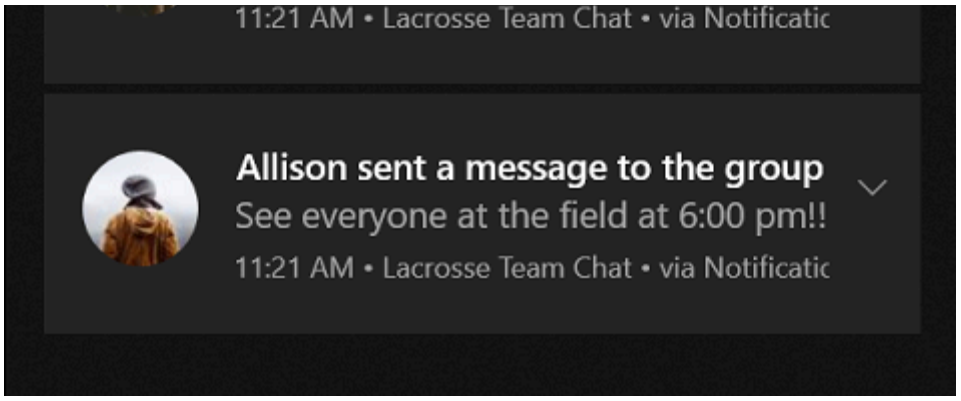
Use collections to organize your app's toasts in Action Center. Collections help users locate information within Action Center more easily and allow for developers to better manage their notifications. The APIs below allow for removing, creating, and updating notification collections.

> **ⓘ Important**
>
> **Requires Creators Update**: You must target SDK 15063 and be running build 15063 or later to use toast collections. Related APIs include Windows.UI.Notifications.ToastCollection, and Windows.UI.Notifications.ToastCollectionManager

You can see the example below with a messaging app that separates the notifications based on the chat group; each title (Comp Sci 160A Project Chat, Direct Messages, Lacrosse Team Chat) is a separate collection. Notice how the notifications are distinctly grouped as if they were from a separate app, even though they are all notifications from the same app. If you are looking for a more subtle way to organize your notifications,

see toast headers.


Comp Sci 160A Project Chat
Adam sent a message to the group
How about Saturday at 11?
11:37 AM • via Notifications Visualizer
Like
Reply
Adam sent a message to the group
The final project is due next Thursday
11:36 AM • via Notifications Visualizer
Direct Messages
Harriet sent you a direct message
On my way over to the field, need a ride?
11:25 AM • via Notifications Visualizer
Like
Reply
Lacrosse Team Chat
Harriet sent a message to the group
I got it! Also, this is our biggest game of the season ladies, let's stay focused and kick some ...
11:21 AM • Lacrosse Team Chat • via Notifications Visualizer
Like
Reply
Dionna sent a message to the group
Can someone pick up the extra ball b



Creating collections

When creating each collection, you are required to provide a display name and an icon, which are displayed inside Action Center as part of the collection's title, as shown in the image above. Collections also require a launch argument to help the app navigate to the right location within the app when the collection's title is clicked on by the user.

Create a collection

C#

```
// Create a toast collection
public async void CreateToastCollection()
{
    string displayName = "Work Email";
    string launchArg = "NavigateToWorkEmailInbox";
    Uri icon = new Windows.Foundation.Uri("ms-
appx:///Assets/workEmail.png");

    // Constructor
    ToastCollection workEmailToastCollection = new ToastCollection(
        "MyToastCollection",
        displayName,
        launchArg,
        icon);

    // Calls the platform to create the collection
    await
ToastNotificationManager.GetDefault().GetToastCollectionManager().SaveToastC
ollectionAsync(workEmailToastCollection);
}
```

Sending notifications to a collection

We will cover sending notifications from three different toast pipelines: local, scheduled, and push. For each of these examples we will be creating a sample toast to send with the code immediately below, then we will focus on how to add the toast to a collection via each pipeline.

Construct the toast content:

```
C#

// Construct the content
var content = new ToastContentBuilder()
    .AddText("Adam sent a message to the group")
    .GetToastContent();
```

Send a toast to a collection

```
C#

// Create the toast
ToastNotification toast = new ToastNotification(content.GetXml());

// Get the collection notifier
var notifier = await
ToastNotificationManager.GetDefault().GetToastNotifierForToastCollectionIdAs
ync("MyToastCollection");

// And show the toast
notifier.Show(toast);
```

Add a scheduled toast to a collection

```
C#

// Create scheduled toast from XML above
ScheduledToastNotification scheduledToast = new
ScheduledToastNotification(content.GetXml(),
DateTimeOffset.Now.AddSeconds(10));

// Get notifier
var notifier = await
ToastNotificationManager.GetDefault().GetToastNotifierForToastCollectionIdAs
ync("MyToastCollection");

// Add to schedule
notifier.AddToSchedule(scheduledToast);
```

Send a push toast to a collection

For push toasts, you need to add the X-WNS-CollectionId header to the POST message.

C#

```
// Add header to HTTP request
request.Headers.Add("X-WNS-CollectionId", collectionId);
```

Managing collections

Create the toast collection manager

For the rest of the code snippets in this 'Managing Collections' section we will be using the collectionManager below.

C#

```
ToastCollectionManger collectionManager =
ToastNotificationManager.GetDefault().GetToastCollectionManager();
```

Get all collections

C#

```
IReadOnlyList<ToastCollection> collections = await
collectionManager.FindAllToastCollectionsAsync();
```

Get the number of collections created

C#

```
int toastCollectionCount = (await
collectionManager.FindAllToastCollectionsAsync()).Count;
```

Remove a collection

C#

```
await collectionManager.RemoveToastCollectionAsync("MyToastCollection");
```

Update a collection

You can update collections by creating a new collection with the same ID and saving the new instance of the collection.

C#

```
string displayName = "Updated Display Name";
string launchArg = "UpdatedLaunchArgs";
Uri icon = new Windows.Foundation.Uri("ms-
appx:///Assets/updatedPicture.png");

// Construct a new toast collection with the same collection id
ToastCollection updatedToastCollection = new ToastCollection(
    "MyToastCollection",
    displayName,
    launchArg,
    icon);

// Calls the platform to update the collection by saving the new instance
await collectionManager.SaveToastCollectionAsync(updatedToastCollection);
```

Managing toasts within a collection

Group and tag properties

The group and tag properties together uniquely identify a notification within a collection. Group (and Tag) serves as a composite primary key (more than one identifier) to identify your notification. For example, if you want to remove or replace a notification, you have to be able to specify *what notification* you want to remove/replace; you do that by specifying the Tag and Group. An example is a messaging app. The developer could use the conversation ID as the Group, and the message ID as the Tag.

Remove a toast from a collection

You can remove individual toasts using the tag and group IDs, or clear all toasts in a collection.

C#

```
// Get the history
var collectionHistory = await
ToastNotificationManager.GetDefault().GetHistoryForToastCollectionAsync("MyT
oastCollection");

// Remove toast
collectionHistory.Remove(tag, group);
```

Clear all toasts within a collection

C#

```
// Get the history
var collectionHistory = await
ToastNotificationManager.GetDefault().GetHistoryForToastCollectionAsync("MyT
oastCollection");

// Remove toast
collectionHistory.Clear();
```

Collections in Notifications Visualizer

You can use the Notifications Visualizer tool to help design your collections. Follow the steps below:

- Click on the gear icon in the bottom right corner.
- Select 'Toast collections'.
- Above the preview of the toast, there is a 'Toast Collection' dropdown menu. Select manage collections.
- Click 'Add collection', fill out the details for the collection, and save.
- You can add more collections, or click off of the manage collections box to return to the main screen.
- Select the collection you want to add the toast to from the 'Toast Collection' dropdown menu.
- When you fire the toast, it will be added to the appropriate collection in Action Center.

Other details

The toast collections that you create will also be reflected in the user's notification settings. Users can toggle the settings for each individual collection to turn these subgroups on or off. If notifications are turned off at the top level for the app, then all

collections notifications will be turned off as well. In addition, each collection will by default show 3 notifications in Action Center, and the user can expand it to show up to 20 notifications.

Related topics

- Toast content
- Toast headers
- Notifications library on GitHub (part of the Windows Community Toolkit)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

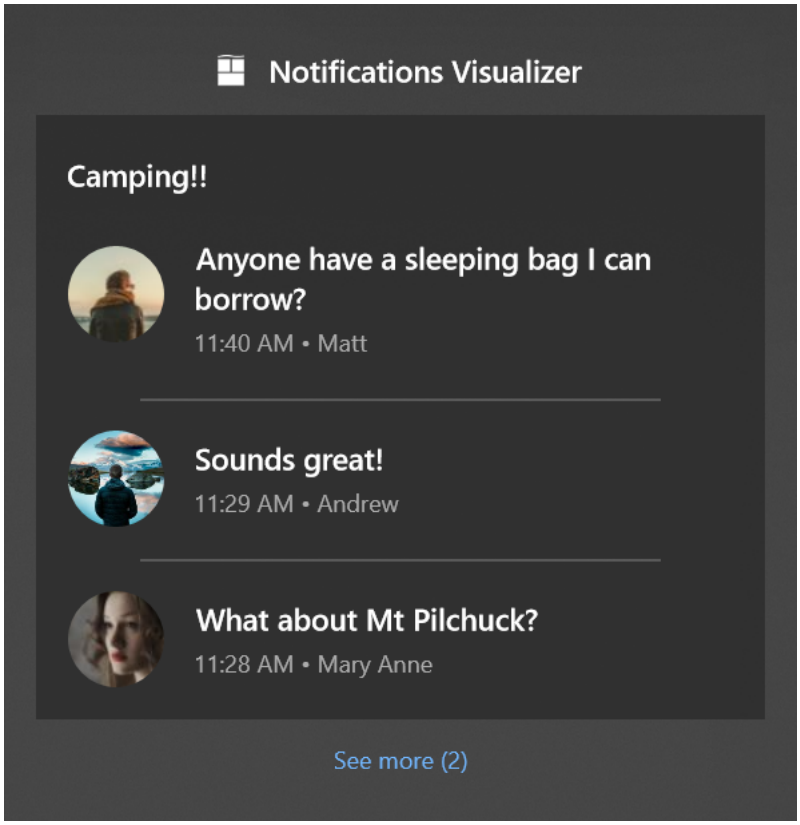
Toast headers

Article • 10/27/2022

You can visually group a set of related notifications inside Action Center by using a toast header on your notifications.

> **Important**
>
> **Requires Desktop Creators Update and 1.4.0 of Notifications library**: You must be running Desktop build 15063 or later to see toast headers. You must use version 1.4.0 or later of the **UWP Community Toolkit Notifications NuGet library** to construct the header in your toast's content. Headers are only supported on Desktop.

As seen below, this group conversation is unified under a single header, "Camping!!". Each individual message in the conversation is a separate toast notification sharing the same toast header.



You can also choose to visually group your notifications by category too, like flight reminders, package tracking, and more.

Add a header to a toast

Here's how you add a header to a toast notification.

> **Note**
>
> Headers are only supported on Desktop. Devices that don't support headers will simply ignore the header.

Builder syntax

C#

```
new ToastContentBuilder()
    .AddHeader("6289", "Camping!!", "action=openConversation&id=6289")
    .AddText("Anyone have a sleeping bag I can borrow?");
```

In summary...

1. Add the **Header** to your **ToastContent**
2. Assign the required **Id**, **Title**, and **Arguments** properties
3. Send your notification (learn more)
4. On another notification, use the same header **Id** to unify them under the header. The **Id** is the only property used to determine whether the notifications should be grouped, meaning the **Title** and **Arguments** can be different. The **Title** and **Arguments** from the most recent notification within a group are used. If that notification gets removed, then the **Title** and **Arguments** falls back to the next most recent notification.

Handle activation from a header

Headers are clickable by users, so that the user can click the header to find out more from your app.

Therefore, apps can provide **Arguments** on the header, similar to the launch arguments on the toast itself.

Activation is handled identical to normal toast activation, meaning you can retrieve these arguments in the **OnActivated** method of `App.xaml.cs` just like you do when the user clicks the body of your toast or a button on your toast.

C#

```
protected override void OnActivated(IActivatedEventArgs e)
{
    // Handle toast activation
    if (e is ToastNotificationActivatedEventArgs)
    {
        // Arguments specified from the header
        string arguments = (e as
ToastNotificationActivatedEventArgs).Argument;
    }
}
```

Additional info

The header visually separates and groups notifications. It doesn't change any other logistics about the maximum number of notifications an app can have (20) and the first-in-first-out behavior of the notifications list.

The order of notifications within headers are as follows... For a given app, the most recent notification from the app (and the entire header group if part of a header) will appear first.

The **Id** can be any string you choose. There are no length or character restrictions on any of the properties in **ToastHeader**. The only constraint is that your entire XML toast content cannot be larger than 5 KB.

Creating headers doesn't change the number of notifications shown inside Action Center before the "See more" button appears (this number is 3 by default and can be configured by the user for each app in system Settings for notifications).

Clicking on the header, just like clicking on the app title, does not clear any notifications belonging to this header (your app should use the toast APIs to clear the relevant notifications).

Related topics

- Send a local toast and handle activation
- Toast content documentation

Notification listener: Access all notifications

Article • 10/27/2022

The notification listener provides access to a user's notifications. Smartwatches and other wearables can use the notification listener to send the phone's notifications to the wearable device. Home automation apps can use notification listener to perform specific actions when notifications are received, such as making the lights blink when you receive a call.

> **ⓘ Important**
>
> **Requires Anniversary Update**: You must target SDK 14393 and be running build 14393 or later to use Notification Listener.

> **Important APIs**: UserNotificationListener class, UserNotificationChangedTrigger class

Enable the listener by adding the User Notification capability

To use the notification listener, you must add the User Notification Listener capability to your app manifest.

1. In Visual Studio, in the Solution Explorer, double click your `Package.appxmanifest` file to open the manifest designer.
2. Open the Capabilities tab.
3. Check the **User Notification Listener** capability.

Check whether the listener is supported

If your app supports older versions of Windows 10, you need to use the ApiInformation class to check whether the listener is supported. If the listener isn't supported, avoid executing any calls to the listener APIs.

C#

```
if
(ApiInformation.IsTypePresent("Windows.UI.Notifications.Management.UserNotif
```

```
icationListener"))
{
    // Listener supported!
}

else
{
    // Older version of Windows, no Listener
}
```

Requesting access to the listener

Since the listener allows access to the user's notifications, users must give your app permission to access their notifications. During your app's first-run experience, you should request access to use the notification listener. If you want, you can show some preliminary UI that explains why your app needs access to the user's notifications before you call RequestAccessAsync, so that the user understands why they should allow access.

C#

```
// Get the listener
UserNotificationListener listener = UserNotificationListener.Current;

// And request access to the user's notifications (must be called from UI
thread)
UserNotificationListenerAccessStatus accessStatus = await
listener.RequestAccessAsync();

switch (accessStatus)
{
    // This means the user has granted access.
    case UserNotificationListenerAccessStatus.Allowed:

        // Yay! Proceed as normal
        break;

    // This means the user has denied access.
    // Any further calls to RequestAccessAsync will instantly
    // return Denied. The user must go to the Windows settings
    // and manually allow access.
    case UserNotificationListenerAccessStatus.Denied:

        // Show UI explaining that listener features will not
        // work until user allows access.
        break;

    // This means the user closed the prompt without
    // selecting either allow or deny. Further calls to
    // RequestAccessAsync will show the dialog again.
```

```
    case UserNotificationListenerAccessStatus.Unspecified:

        // Show UI that allows the user to bring up the prompt again
        break;
}
```

The user can revoke access at any time via Windows Settings. Therefore, your app should always check the access status via the GetAccessStatus method before executing code that uses the notification listener. If the user revokes access, the APIs will silently fail rather than throwing an exception (for example, the API to get all notifications will simply return an empty list).

Access the user's notifications

With the notification listener, you can get a list of the user's current notifications. Simply call the GetNotificationsAsync method, and specify the type of notifications you want to get (currently, the only type of notifications supported are toast notifications).

C#

```
// Get the toast notifications
IReadOnlyList<UserNotification> notifs = await
listener.GetNotificationsAsync(NotificationKinds.Toast);
```

Displaying the notifications

Each notification is represented as a UserNotification, which provides information about the app that the notification is from, the time the notification was created, the notification's ID, and the notification itself.

C#

```
public sealed class UserNotification
{
    public AppInfo AppInfo { get; }
    public DateTimeOffset CreationTime { get; }
    public uint Id { get; }
    public Notification Notification { get; }
}
```

The AppInfo property provides the info you need to display the notification.

> ⓘ Note

We recommend surrounding all your code for processing a single notification in a try/catch, in case an unexpected exception occurs when you are capturing a single notification. You shouldn't completely fail to display other notifications just because of an issue with one specific notification.

C#

```
// Select the first notification
UserNotification notif = notifs[0];

// Get the app's display name
string appDisplayName = notif.AppInfo.DisplayInfo.DisplayName;

// Get the app's logo
BitmapImage appLogo = new BitmapImage();
RandomAccessStreamReference appLogoStream =
notif.AppInfo.DisplayInfo.GetLogo(new Size(16, 16));
await appLogo.SetSourceAsync(await appLogoStream.OpenReadAsync());
```

The content of the notification itself, such as the notification text, is contained in the Notification property. This property contains the visual portion of the notification. (If you are familiar with sending notifications on Windows, you will notice that the Visual and Visual.Bindings properties in the Notification object correspond to what developers send when popping a notification.)

We want to look for the toast binding (for error-proof code, you should check that the binding isn't null). From the binding, you can obtain the text elements. You can choose to display as many text elements as you would like. (Ideally, you should display them all.) You can choose to treat the text elements differently; for example, treat the first one as title text, and subsequent elements as body text.

C#

```
// Get the toast binding, if present
NotificationBinding toastBinding =
notif.Notification.Visual.GetBinding(KnownNotificationBindings.ToastGeneric)
;

if (toastBinding != null)
{
    // And then get the text elements from the toast binding
    IReadOnlyList<AdaptiveNotificationText> textElements =
toastBinding.GetTextElements();

    // Treat the first text element as the title text
    string titleText = textElements.FirstOrDefault()?.Text;

    // We'll treat all subsequent text elements as body text,
```

```
    // joining them together via newlines.
    string bodyText = string.Join("\n", textElements.Skip(1).Select(t =>
t.Text));
}
```

Remove a specific notification

If your wearable or service allows the user to dismiss notifications, you can remove the actual notification so the user doesn't see it later on their phone or PC. Simply provide the notification ID (obtained from the UserNotification object) of the notification you'd like to remove:

C#

```
// Remove the notification
listener.RemoveNotification(notifId);
```

Clear all notifications

The UserNotificationListener.ClearNotifications method clears all the user's notifications. Use this method with caution. You should only clear all notifications if your wearable or service displays ALL notifications. If your wearable or service only displays certain notifications, when the user clicks your "Clear notifications" button, the user is only expecting those specific notifications to be removed; however, calling the ClearNotifications method would actually cause all the notifications, including ones that your wearable or service wasn't displaying, to be removed.

C#

```
// Clear all notifications. Use with caution.
listener.ClearNotifications();
```

Background task for notification added/dismissed

A common way to enable an app to listen to notifications is to set up a background task, so that you can know when a notification was added or dismissed regardless of whether your app is currently running.

Thanks to the single process model added in the Anniversary Update, adding background tasks is fairly simple. In your main app's code, after you have obtained the user's access to Notification Listener and obtained access to run background tasks by calling UserNotificationListener.Current.RequestAccessAsync and BackgroundExecutionManager.RequestAccessAsync respectively, simply register a new background task, and set the UserNotificationChangedTrigger using the Toast notification kind.

C#

```
// TODO: Request/check Listener access via
UserNotificationListener.Current.RequestAccessAsync

// TODO: Request/check background task access via
BackgroundExecutionManager.RequestAccessAsync

// If background task isn't registered yet
if (!BackgroundTaskRegistration.AllTasks.Any(i =>
i.Value.Name.Equals("UserNotificationChanged")))
{
    // Specify the background task
    var builder = new BackgroundTaskBuilder()
    {
        Name = "UserNotificationChanged"
    };

    // Set the trigger for Listener, listening to Toast Notifications
    builder.SetTrigger(new
UserNotificationChangedTrigger(NotificationKinds.Toast));

    // Register the task
    builder.Register();
}
```

Then, in your App.xaml.cs, override the OnBackgroundActivated method if you haven't yet, and use a switch statement on the task name to determine which of your many background task triggers was invoked.

C#

```
protected override async void
OnBackgroundActivated(BackgroundActivatedEventArgs args)
{
    var deferral = args.TaskInstance.GetDeferral();

    switch (args.TaskInstance.Task.Name)
    {
        case "UserNotificationChanged":
            // Call your own method to process the new/removed notifications
            // The next section of documentation discusses this code
```

```
            await MyWearableHelpers.SyncNotifications();
            break;
    }

    deferral.Complete();
}
```

The background task is simply a "shoulder tap": it doesn't provide any information about which specific notification was added or removed. When your background task is triggered, you should sync the notifications on your wearable so that they reflect the notifications in the platform. This ensures that if your background task fails, notifications on your wearable can still be recovered the next time your background task executes.

`SyncNotifications` is a method you implement; the next section shows how.

Determining which notifications were added and removed

In your `SyncNotifications` method, to determine which notifications have been added or removed (syncing notifications with your wearable), you have to calculate the delta between your current notification collection, and the notifications in the platform.

C#

```
// Get all the current notifications from the platform
IReadOnlyList<UserNotification> userNotifications = await
listener.GetNotificationsAsync(NotificationKinds.Toast);

// Obtain the notifications that our wearable currently has displayed
IList<uint> wearableNotificationIds = GetNotificationsOnWearable();

// Copy the currently displayed into a list of notification ID's to be
removed
var toBeRemoved = new List<uint>(wearableNotificationIds);

// For each notification in the platform
foreach (UserNotification userNotification in userNotifications)
{
    // If we've already displayed this notification
    if (wearableNotificationIds.Contains(userNotification.Id))
    {
        // We want to KEEP it displayed, so take it out of the list
        // of notifications to remove.
        toBeRemoved.Remove(userNotification.Id);
    }

    // Otherwise it's a new notification
    else
```

```
    {
        // Display it on the Wearable
        SendNotificationToWearable(userNotification);
    }
}

// Now our toBeRemoved list only contains notification ID's that no longer
exist in the platform.
// So we will remove all those notifications from the wearable.
foreach (uint id in toBeRemoved)
{
    RemoveNotificationFromWearable(id);
}
```

Foreground event for notification added/dismissed

> **ⓘ Important**
>
> Known issue: In builds before Build 17763 / October 2018 Update / Version 1809, The foreground event will cause a CPU loop and/or didn't work. If you need support on those earlier builds, use the background task instead.

You can also listen to notifications from an in-memory event handler...

C#

```
// Subscribe to foreground event
listener.NotificationChanged += Listener_NotificationChanged;

private void Listener_NotificationChanged(UserNotificationListener sender,
UserNotificationChangedEventArgs args)
{
    // Your code for handling the notification
}
```

How to fix delays in the background task

When testing your app, you might notice that the background task is sometimes delayed and doesn't trigger for several minutes. To fix the delay, prompt the user to go to the system settings -> System -> Battery -> Battery usage by app, find your app in the list, select it, and set it to be "Always allowed in background." After this, the

background task should always be triggered within around a second of the notification being received.

Notification mirroring

Article • 10/27/2022

Notification mirroring, powered by Action Center in the Cloud, allows you to see your phone's notifications on your PC.

> **Important**
>
> **Requires Anniversary Update**: You must be running build 14393 or later to see notification mirroring work. If you would like to opt your app out of notification mirroring, you must target SDK 14393 to access the mirroring APIs.

With notification mirroring and Cortana, users can receive and act on their phone's notifications (Windows Mobile and Android) from the convenience of their PC. As a developer, you don't have to do anything to enable notification mirroring, mirroring automatically works! Clicking buttons on the mirrored toast, like message quick replies, will be routed back to the phone, invoking you background task or launching your foreground app.

Developers get two great benefits from notification mirroring: The mirrored notifications result in more user engagement with your service, and they also help users discover your Microsoft Store desktop app! Your users might not even know that you have an awesome Windows app available for their Windows 10 desktop. When users receive the mirrored notification from their phone, users can click the toast notification to be taken to the Microsoft Store, where they can install your Windows app.

Mirroring works with both Windows Phone and Android. Users need to be logged into Cortana on both their phone and desktop for notification mirroring to work.

What if the app is installed on both devices?

If the user already has your app on their PC, we will automatically mute the mirrored phone notification so that they don't see duplicate notifications. Mirrored notifications will be auto-muted based on the following criteria...

1. An app on the PC exists with either the **same display name or the same PFN** (Package Family Name)
2. That PC app has sent a toast notification

If the PC app hasn't sent a toast yet, we'll still show the phone notifications, since chances are, the user hasn't actually launched the PC app yet).

How to opt out of mirroring

Windows app developers, enterprises, and users can choose to disable notification mirroring.

> ⓘ **Note**
>
> Disabling mirroring will also disable **Universal Dismiss**.

As a developer, opt out an individual notification

You occasionally might have a device-specific notification that you don't want to be mirrored to other devices. You can prevent a specific notification from being mirrored by setting the **Mirroring** property on the toast notification. Currently, this mirroring property can only be set on local notifications (it can not be specified when sending a WNS push notification).

Known Issue: Retrieving the Mirroring property via the `ToastNotificationHistory.GetHistory()` API's will always return the default value (**Allowed**) rather than the option you specified. Don't worry, everything is functional - it's only retrieving the value that's broken.

C#

```
var toast = new ToastNotification(xml)
{
    // Disable mirroring of this notification
    Mirroring = NotificationMirroring.Disabled
};

ToastNotificationManager.CreateToastNotifier().Show(toast);
```

As a developer, opt out completely

Some developers might choose to completely opt their app out of notification mirroring. While we believe that all apps would benefit from mirroring, we make it easy to opt out. Just call the following method once, and your app will be opted out. For example, you can place this call in your app's constructor inside `App.xaml.cs`...

C#

```
public App()
{
    this.InitializeComponent();
    this.Suspending += OnSuspending;

    // Disable notification mirroring for entire app

ToastNotificationManager.ConfigureNotificationMirroring(NotificationMirrorin
g.Disabled);
}
```

As an enterprise, how do I opt out?

Enterprises can choose to completely disable notification mirroring. To do so, they simply edit the Group Policy to turn off notification mirroring.

As a user, how do I opt out?

Users are able to opt out on individual apps, or completely opt out by disabling the feature. You may not want a specific app's notifications mirrored to your desktop, so you can simply disable that specific app. You can find these options in Cortana's settings on both your phone and PC.

Universal Dismiss

Article • 10/27/2022

Universal Dismiss, powered by Action Center in the Cloud, means that when you dismiss a notification from one device, the same notification on your other devices is also dismissed.

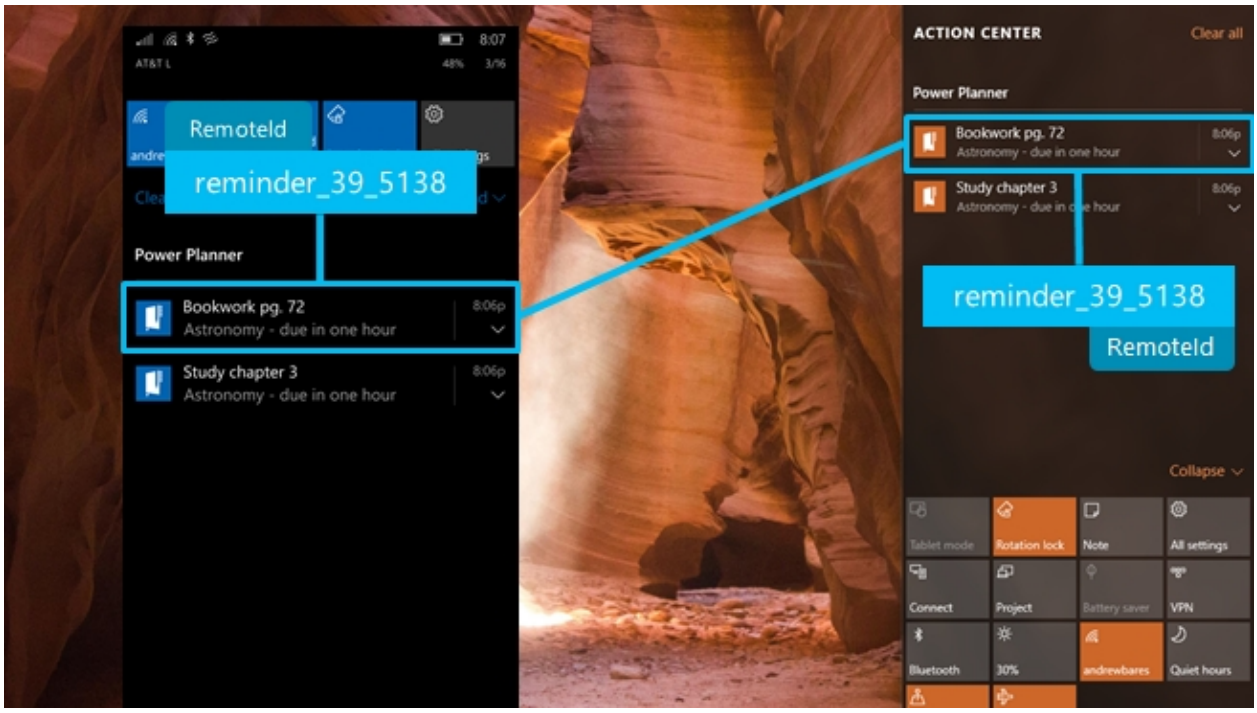
> ⓘ **Important**
>
> **Requires Anniversary Update**: You must target SDK 14393 and be running build 14393 or later to use Universal Dismiss.

The common example of this scenario is calendar reminders... you have a calendar app on both of your devices... you get a reminder on your phone and desktop... you click dismiss on your desktop... thanks to Universal Dismiss, the reminder on your phone is also dismissed! **Enabling Universal Dismiss only requires one line of code!**

In this scenario, the key fact is that **the same app is installed on multiple devices**, meaning that **each device is already receiving notifications**. A calendar app is the iconic example, since you typically have the same calendar app installed on both your Windows PC and your phone, and each instance of the app already sends you reminders on each device. By adding support for Universal Dismiss, those instances of the same reminders can be linked across devices.

How to enable Universal Dismiss

As a developer, enabling Universal Dismiss is extremely easy. You simply need to provide an ID that allows us to link each notification across devices, so that when the user dismisses a notification from one device, the corresponding linked notification is dismissed from the other device.



> **RemoteId**: An identifier that uniquely identifies a notification *across devices*.

t only takes one line of code to add RemoteId, enabling support for Universal Dismiss! How you generate your RemoteId is up to you - however, you need to make sure that it uniquely identifies your notification across devices, and that the same identifier can be generated from different instances of your app running on different devices.

For example, in my homework planner app, I generate my RemoteId by saying that it is of type "reminder", and then I include the online account ID and the online identifier of the homework item. I can consistently generate the exact same RemoteId, regardless of which device is sending the notification, since these online IDs are shared across the devices.

C#

```
var toast = new ScheduledToastNotification(content.GetXml(), startTime);

// If the RemoteId property is present
if
(ApiInformation.IsPropertyPresent(typeof(ScheduledToastNotification).FullNam
e, nameof(ScheduledToastNotification.RemoteId)))
{
    // Assign the RemoteId to add support for Universal Dismiss
    toast.RemoteId = $"reminder_{account.AccountId}_{homework.Identifier}"
}

ToastNotificationManager.CreateToastNotifier().AddToSchedule(toast);
```

The following code runs on both my phone and desktop app, meaning that the notification on both devices will have the same RemoteId.

That's all you have to do! When the user dismisses (or clicks on) a notification, we'll check if it has a RemoteId, and if so, we'll fan out a dismiss of that RemoteId across all the user's devices.

Known Issue: Retrieving the **RemoteId** via the `ToastNotificationHistory.GetHistory()` API's will always return empty string rather than the **RemoteId** you specified. Don't worry, everything is functional - it's only retrieving the value that's broken.

> ⓘ **Note**
>
> If the user or enterprise disables **notification mirroring** for your app (or completely disables notification mirroring), then Universal Dismiss will not work, since we do not have your notifications in the cloud.

Supported devices

Since the Anniversary Update, Universal Dismiss is supported on Windows Mobile and Windows Desktop. Universal Dismiss works both directions, between PC-PC, PC-Phone, and Phone-Phone.

Badge notifications for Windows apps

Article • 02/27/2025

A notification badge conveys summary or status information specific to your app. They can be numeric (1-99) or one of a set of system-provided glyphs. Examples of information best conveyed through a badge include network connection status in an online game, user status in a messaging app, number of unread mails in a mail app, and number of new posts in a social media app.

Notification badges appear on your app's taskbar icon and in the lower-right corner of its start tile, regardless of whether the app is running. Badges can be displayed on all tile sizes.

> **Note**
>
> You cannot provide your own badge image; only system-provided badge images can be used.

Numeric badges

Expand table

Value	Badge	XML
A number from 1 to 99. A value of 0 is equivalent to the glyph value "none" and will clear the badge.		`<badge value="1"/>`
Any number greater than 99.		`<badge value="100"/>`

Glyph badges

Instead of a number, a badge can display one of a non-extensible set of status glyphs.

Expand table

Status	Glyph	XML
none	(No badge shown.)	`<badge value="none"/>`

Status	Glyph	XML
activity		`<badge value="activity"/>`
alarm		`<badge value="alarm"/>`
alert		`<badge value="alert"/>`
attention		`<badge value="attention"/>`
available		`<badge value="available"/>`
away		`<badge value="away"/>`
busy		`<badge value="busy"/>`
error		`<badge value="error"/>`
newMessage		`<badge value="newMessage"/>`
paused		`<badge value="paused"/>`
playing		`<badge value="playing"/>`
unavailable		`<badge value="unavailable"/>`

Create a badge

These examples show you how to create a badge update.

Create a numeric badge

C#

```
private void setBadgeNumber(int num)
{

    // Get the blank badge XML payload for a badge number
    XmlDocument badgeXml =

BadgeUpdateManager.GetTemplateContent(BadgeTemplateType.BadgeNumber);

    // Set the value of the badge in the XML to our number
    XmlElement badgeElement = badgeXml.SelectSingleNode("/badge") as
XmlElement;
    badgeElement.SetAttribute("value", num.ToString());

    // Create the badge notification
    BadgeNotification badge = new BadgeNotification(badgeXml);

    // Create the badge updater for the application
    BadgeUpdater badgeUpdater =
        BadgeUpdateManager.CreateBadgeUpdaterForApplication();

    // And update the badge
    badgeUpdater.Update(badge);

}
```

Create a glyph badge

C#

```
private void updateBadgeGlyph()
{
    string badgeGlyphValue = "alert";

    // Get the blank badge XML payload for a badge glyph
    XmlDocument badgeXml =
        BadgeUpdateManager.GetTemplateContent(BadgeTemplateType.BadgeGlyph);

    // Set the value of the badge in the XML to our glyph value
    Windows.Data.Xml.Dom.XmlElement badgeElement =
        badgeXml.SelectSingleNode("/badge") as
Windows.Data.Xml.Dom.XmlElement;
    badgeElement.SetAttribute("value", badgeGlyphValue);

    // Create the badge notification
    BadgeNotification badge = new BadgeNotification(badgeXml);

    // Create the badge updater for the application
```

```
    BadgeUpdater badgeUpdater =
        BadgeUpdateManager.CreateBadgeUpdaterForApplication();

    // And update the badge
    badgeUpdater.Update(badge);

}
```

Clear a badge

C#

```
private void clearBadge()
{
    BadgeUpdateManager.CreateBadgeUpdaterForApplication().Clear();
}
```

Get the sample code

- Notifications sample
  Shows how to create live tiles, send badge updates, and display toast notifications.

Related articles

- Adaptive and interactive toast notifications
- Create tiles
- Create adaptive tiles

Windows Push Notification Services (WNS) overview

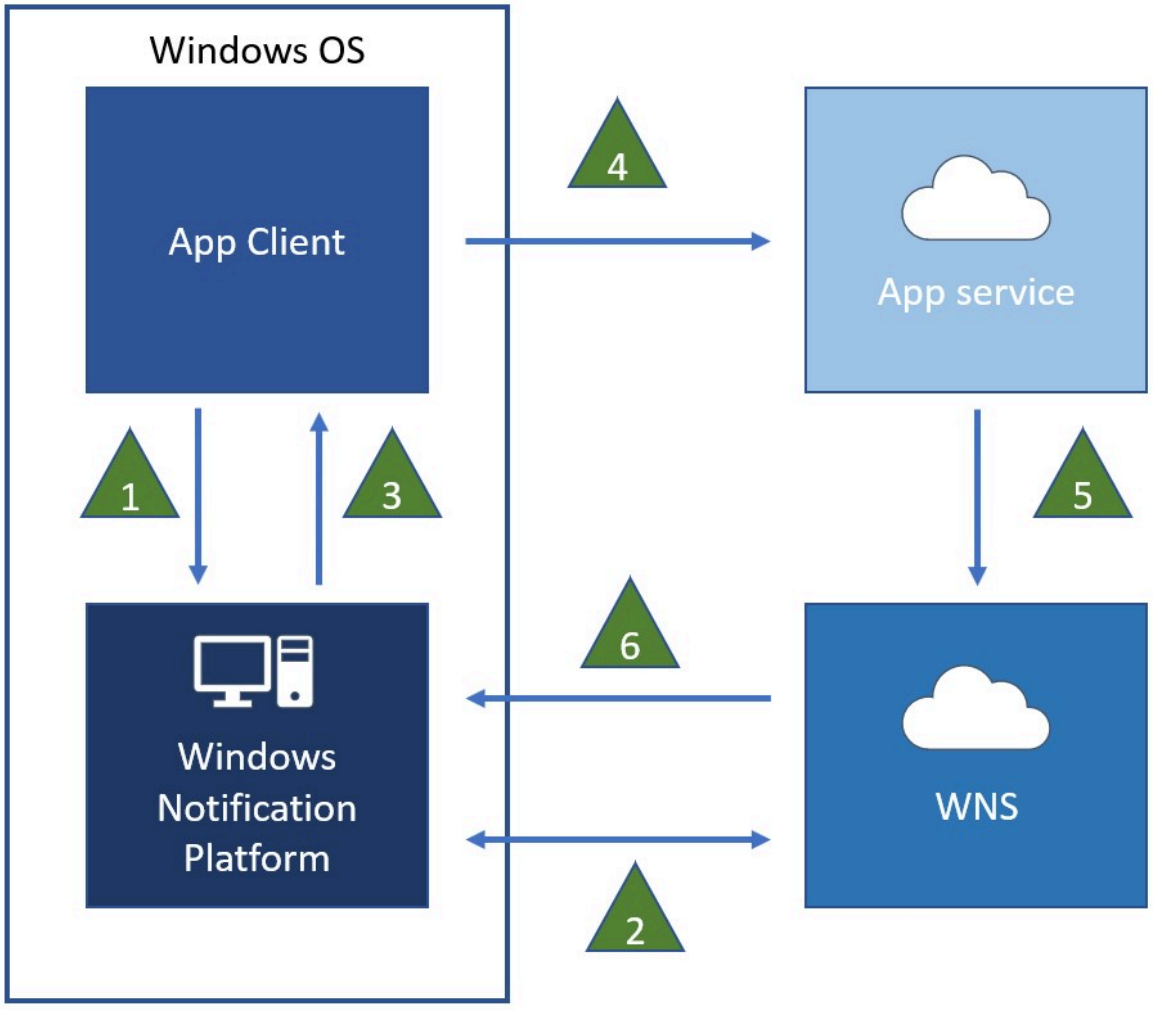
Article • 11/27/2024

The Windows Push Notification Services (WNS) enable third-party developers to send toast, tile, badge, and raw updates from their own cloud service. This provides a mechanism to deliver new updates to your users in a power-efficient and dependable way.

How it works

The following diagram shows the complete data flow for sending a push notification. It involves these steps:

1. Your app requests a push notification channel from WNS.
2. Windows asks WNS to create a notification channel. This channel is returned to the calling device in the form of a Uniform Resource Identifier (URI).
3. The notification channel URI is returned by WNS to your app.
4. Your app sends the URI to your own cloud service. You then store the URI on your own cloud service so that you can access the URI when you send notifications. The URI is an interface between your own app and your own service; it's your responsibility to implement this interface with safe and secure web standards.
5. When your cloud service has an update to send, it notifies WNS using the channel URI. This is done by issuing an HTTP POST request, including the notification payload, over Secure Sockets Layer (SSL). This step requires authentication.
6. WNS receives the request and routes the notification to the appropriate device.



Registering your app and receiving the credentials for your cloud service

Before you can send notifications using WNS, your app must be registered with the Store Dashboard, as described here.

Requesting a notification channel

When an app that is capable of receiving push notifications runs, it must first request a notification channel through the **CreatePushNotificationChannelForApplicationAsync**. For a full discussion and example code, see How to request, create, and save a notification channel. This API returns a channel URI that is uniquely linked to the calling application and its tile, and through which all notification types can be sent.

After the app has successfully created a channel URI, it sends it to its cloud service, together with any app-specific metadata that should be associated with this URI.

Important notes

- We do not guarantee that the notification channel URI for an app will always remain the same. We advise that the app requests a new channel every time it runs and updates its service when the URI changes. The developer should never modify the channel URI and should consider it as a black-box string. At this time, channel URIs expire after 30 days. If your Windows 10 app will periodically renew its channel in the background then you can download the Push and periodic notifications sample for Windows 8.1 and re-use its source code and/or the pattern it demonstrates.
- The interface between the cloud service and the client app is implemented by you, the developer. We recommend that the app go through an authentication process with its own service and transmit data over a secure protocol such as HTTPS.
- It is important that the cloud service always ensures that the channel URI uses the domain "notify.windows.com". The service should never push notifications to a channel on any other domain. If the callback for your app is ever compromised, a malicious attacker could submit a channel URI to spoof WNS. Without inspecting the domain, your cloud service could potentially disclose information to this attacker unknowingly. The subdomain of the channel URI is subject to change and should not be considered when validating the channel URI.
- If your cloud service attempts to deliver a notification to an expired channel, WNS will return response code 410. In response to that code, your service should no longer attempt to send notifications to that URI.

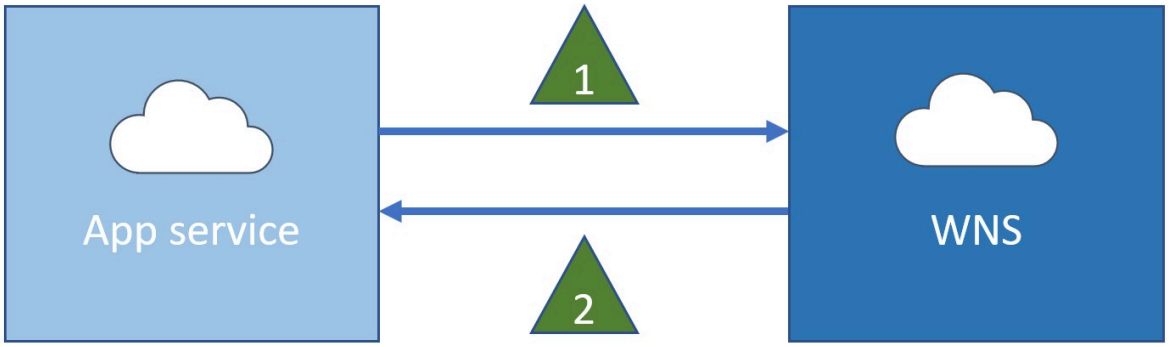
Authenticating your cloud service

To send a notification, the cloud service must be authenticated through WNS. The first step in this process occurs when you register your app with the Microsoft Store Dashboard. During the registration process, your app is given a Package security identifier (SID) and a secret key. This information is used by your cloud service to authenticate with WNS.

The WNS authentication scheme is implemented using the client credentials profile from the OAuth 2.0 protocol. The cloud service authenticates with WNS by providing its credentials (Package SID and secret key). In return, it receives an access token. This access token allows a cloud service to send a notification. The token is required with every notification request sent to the WNS.

At a high level, the information chain is as follows:

1. The cloud service sends its credentials to WNS over HTTPS following the OAuth 2.0 protocol. This authenticates the service with WNS.
2. WNS returns an access token if the authentication was successful. This access token is used in all subsequent notification requests until it expires.



In the authentication with WNS, the cloud service submits an HTTP request over Secure Sockets Layer (SSL). The parameters are supplied in the "application/x-www-for-urlencoded" format. Supply your Package SID in the "client_id" field and your secret key in the "client_secret" field as shown in the following example. For syntax details, see the access token request reference.

> **Note**
>
> This is just an example, not cut-and-paste code that you can successfully use in your own code.

HTTP

```
POST /accesstoken.srf HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: https://login.live.com
Content-Length: 211

grant_type=client_credentials&client_id=ms-app%3a%2f%2fS-1-15-2-2972962901-2322836549-3722629029-1345238579-3987825745-2155616079-650196962&client_secret=Vex8L9WOFZuj95euaLrvSH7XyoDhLJc7&scope=notify.windows.com
```

The WNS authenticates the cloud service and, if successful, sends a response of "200 OK". The access token is returned in the parameters included in the body of the HTTP response, using the "application/json" media type. After your service has received the access token, you are ready to send notifications.

The following example shows a successful authentication response, including the access token. For syntax details, see Push notification service request and response headers.

HTTP

```
HTTP/1.1 200 OK
Cache-Control: no-store
Content-Length: 422
Content-Type: application/json

{
    "access_token":"EgAcAQMAAAAALYAAY/c+Huwi3Fv4Ck10UrKNmtxRO6Njk2MgA=",
    "token_type":"bearer"
}
```

Important notes

- The OAuth 2.0 protocol supported in this procedure follows draft version V16.
- The OAuth Request for Comments (RFC) uses the term "client" to refer to the cloud service.
- There might be changes to this procedure when the OAuth draft is finalized.
- The access token can be reused for multiple notification requests. This allows the cloud service to authenticate just once to send many notifications. However, when the access token expires, the cloud service must authenticate again to receive a new access token.

Sending a notification

Using the channel URI, the cloud service can send a notification whenever it has an update for the user.

The access token described above can be reused for multiple notification requests; the cloud server is not required to request a new access token for every notification. If the access token has expired, the notification request will return an error. We recommended that you do not try to re-send your notification more than once if the access token is rejected. If you encounter this error, you will need to request a new access token and resend the notification. For the exact error code, see Push notification response codes.

1. The cloud service makes an HTTP POST to the channel URI. This request must be made over SSL and contains the necessary headers and the notification payload. The authorization header must include the acquired access token for authorization.

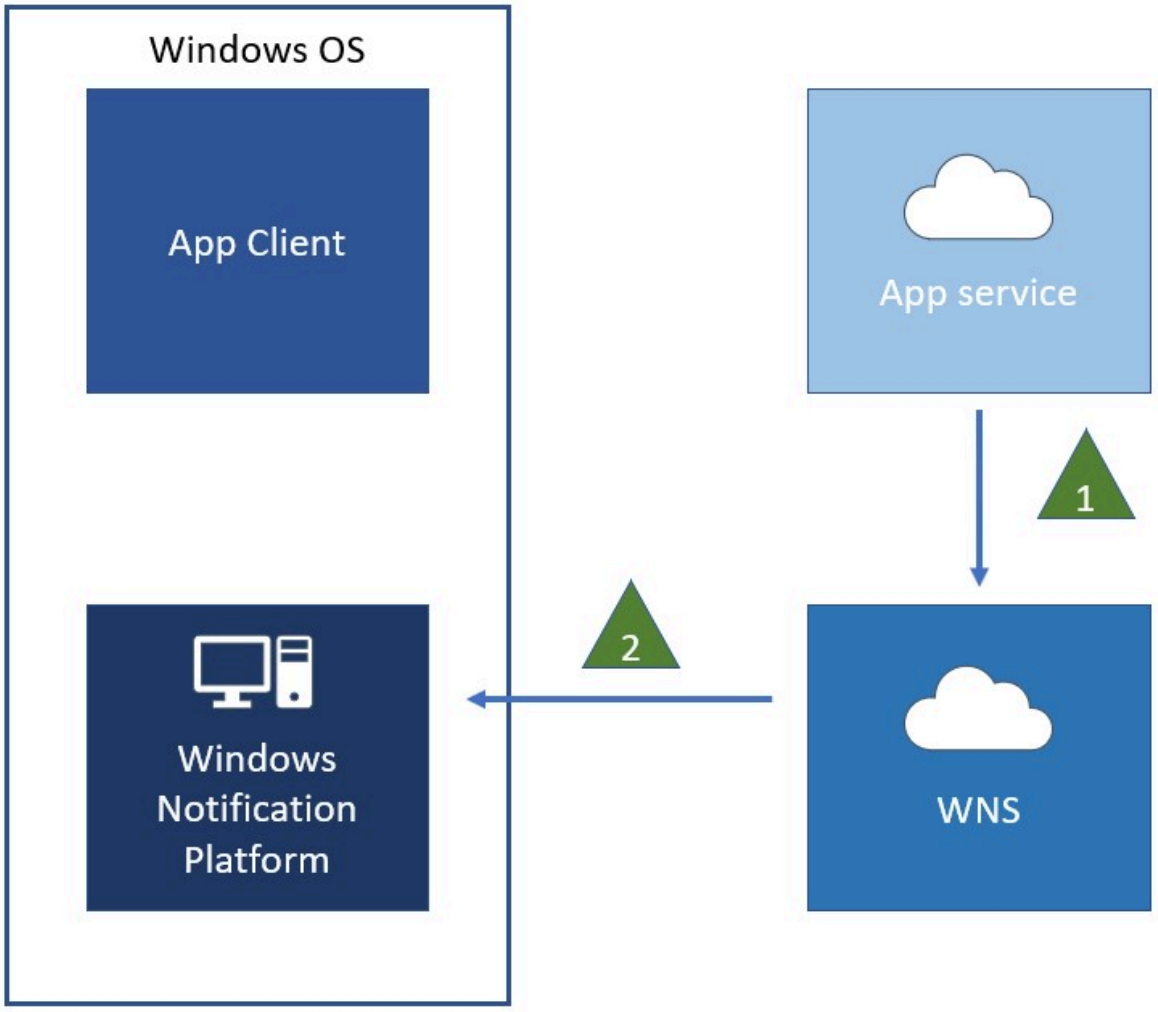
An example request is shown here. For syntax details, see Push notification response codes.

For details on composing the notification payload, see Quickstart: Sending a push notification. The payload of a tile, toast, or badge push notification is supplied as XML content that adheres to their respective defined Adaptive tiles schema or Legacy tiles schema. The payload of a raw notification does not have a specified structure. It is strictly app-defined.

```HTTP
 POST https://cloud.notify.windows.com/?
token=AQE%bU%2fSjZOCvRjjpILow%3d%3d HTTP/1.1
 Content-Type: text/xml
 X-WNS-Type: wns/tile
 Authorization: Bearer
EgAcAQMAAAAALYAAY/c+Huwi3Fv4Ck10UrKNmtxRO6Njk2MgA=
 Host: cloud.notify.windows.com
 Content-Length: 24

 <body>
 ....
```

2. WNS responds to indicate that the notification has been received and will be delivered at the next available opportunity. However, WNS does not provide end-to-end confirmation that your notification has been received by the device or application.

This diagram illustrates the data flow:



Important notes

- WNS does not guarantee the reliability or latency of a notification.
- Notifications should never include confidential, sensitive, or personal data.
- To send a notification, the cloud service must first authenticate with WNS and receive an access token.
- An access token only allows a cloud service to send notifications to the single app for which the token was created. One access token cannot be used to send notifications across multiple apps. Therefore, if your cloud service supports multiple apps, it must provide the correct access token for the app when pushing a notification to each channel URI.
- When the device is offline, by default WNS will store one of each notification type (tile, badge, toast) for each channel URI and no raw notifications.
- In scenarios where the notification content is personalized to the user, WNS recommends that the cloud service immediately send those updates when those are received. Examples of this scenario include social media feed updates, instant communication invitations, new message notifications, or alerts. As an alternative, you can have scenarios in which the same generic update is frequently delivered to

a large subset of your users; for example, weather, stock, and news updates. WNS guidelines specify that the frequency of these updates should be at most one every 30 minutes. The end user or WNS may determine more frequent routine updates to be abusive.
- Windows Notification Platform maintains a periodic data connection with WNS to keep the socket alive and healthy. If there are no applications requesting or using notification channels then the socket will not be created.

Expiration of tile and badge notifications

By default, tile and badge notifications expire three days after being downloaded. When a notification expires, the content is removed from the tile or queue and is no longer shown to the user. It's a best practice to set an expiration (using a time that makes sense for your app) on all tile and badge notifications so that your tile's content doesn't persist longer than it is relevant. An explicit expiration time is essential for content with a defined lifespan. This also assures the removal of stale content if your cloud service stops sending notifications, or if the user disconnects from the network for an extended period.

Your cloud service can set an expiration for each notification by setting the X-WNS-TTL HTTP header to specify the time (in seconds) that your notification will remain valid after it is sent. For more information, see Push notification service request and response headers.

For example, during a stock market's active trading day, you can set the expiration for a stock price update to twice that of your sending interval (such as one hour after receipt if you are sending notifications every half-hour). As another example, a news app might determine that one day is an appropriate expiration time for a daily news tile update.

Push notifications and battery saver

Battery saver extends battery life by limiting background activity on the device. Windows 10 lets the user set battery saver to turn on automatically when the battery drops below a specified threshold. When battery saver is on, the receipt of push notifications is disabled to save energy. But there are a couple exceptions to this. The following Windows 10 battery saver settings (found in the **Settings** app) allow your app to receive push notifications even when battery saver is on.

- **Allow push notifications from any app while in battery saver**: This setting lets all apps receive push notifications while battery saver is on. Note that this setting

applies only to Windows 10 for desktop editions (Home, Pro, Enterprise, and Education).

- **Always allowed**: This setting lets specific apps run in the background while battery saver is on - including receiving push notifications. This list is maintained manually by the user.

There is no way to check the state of these two settings, but you can check the state of battery saver. In Windows 10, use the **EnergySaverStatus** property to check battery saver state. Your app can also use the **EnergySaverStatusChanged** event to listen for changes to battery saver.

If your app depends heavily on push notifications, we recommend notifying users that they may not receive notifications while battery saver is on and to make it easy for them to adjust **battery saver settings**. Using the battery saver settings URI scheme in Windows 10, `ms-settings:batterysaver-settings`, you can provide a convenient link to the Settings app.

> **Tip**
>
> When notifying the user about battery saver settings, we recommend providing a way to suppress the message in the future. For example, the `dontAskMeAgainBox` checkbox in the following example persists the user's preference in **LocalSettings**.

Here's an example of how to check whether battery saver is turned on in Windows 10. This example notifies the user and launches the Settings app to **battery saver settings**. The `dontAskAgainSetting` lets the user suppress the message if they don't want to be notified again.

C#

```
using System;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Navigation;
using Windows.System;
using Windows.System.Power;
...
...
async public void CheckForEnergySaving()
{
   //Get reminder preference from LocalSettings
   bool dontAskAgain;
   var localSettings =
Windows.Storage.ApplicationData.Current.LocalSettings;
   object dontAskSetting = localSettings.Values["dontAskAgainSetting"];
```

```
    if (dontAskSetting == null)
    {  // Setting does not exist
       dontAskAgain = false;
    }
    else
    {  // Retrieve setting value
       dontAskAgain = Convert.ToBoolean(dontAskSetting);
    }

    // Check if battery saver is on and that it's okay to raise dialog
    if ((PowerManager.EnergySaverStatus == EnergySaverStatus.On)
          && (dontAskAgain == false))
    {
       // Check dialog results
       ContentDialogResult dialogResult = await saveEnergyDialog.ShowAsync();
       if (dialogResult == ContentDialogResult.Primary)
       {
          // Launch battery saver settings (settings are available only when
a battery is present)
          await Launcher.LaunchUriAsync(new Uri("ms-settings:batterysaver-
settings"));
       }

       // Save reminder preference
       if (dontAskAgainBox.IsChecked == true)
       {  // Don't raise dialog again
          localSettings.Values["dontAskAgainSetting"] = "true";
       }
    }
}
```

C++/WinRT

```
#include <winrt/Windows.Foundation.h>
#include <winrt/Windows.Storage.h>
#include <winrt/Windows.System.h>
#include <winrt/Windows.System.Power.h>
#include <winrt/Windows.UI.Xaml.h>
#include <winrt/Windows.UI.Xaml.Controls.h>
#include <winrt/Windows.UI.Xaml.Navigation.h>
using namespace winrt;
using namespace winrt::Windows::Foundation;
using namespace winrt::Windows::Storage;
using namespace winrt::Windows::System;
using namespace winrt::Windows::System::Power;
using namespace winrt::Windows::UI::Xaml;
using namespace winrt::Windows::UI::Xaml::Controls;
using namespace winrt::Windows::UI::Xaml::Navigation;
...
winrt::fire_and_forget CheckForEnergySaving()
{
    // Get reminder preference from LocalSettings.
    bool dontAskAgain{ false };
```

```
    auto localSettings = ApplicationData::Current().LocalSettings();
    IInspectable dontAskSetting =
localSettings.Values().Lookup(L"dontAskAgainSetting");
    if (!dontAskSetting)
    {
        // Setting doesn't exist.
        dontAskAgain = false;
    }
    else
    {
        // Retrieve setting value
        dontAskAgain = winrt::unbox_value<bool>(dontAskSetting);
    }

    // Check whether battery saver is on, and whether it's okay to raise
dialog.
    if ((PowerManager::EnergySaverStatus() == EnergySaverStatus::On) &&
(!dontAskAgain))
    {
        // Check dialog results.
        ContentDialogResult dialogResult = co_await
saveEnergyDialog().ShowAsync();
        if (dialogResult == ContentDialogResult::Primary)
        {
            // Launch battery saver settings
            // (settings are available only when a battery is present).
            co_await Launcher::LaunchUriAsync(Uri(L"ms-
settings:batterysaver-settings"));
        }

        // Save reminder preference.
        if (dontAskAgainBox().IsChecked())
        {
            // Don't raise the dialog again.
            localSettings.Values().Insert(L"dontAskAgainSetting",
winrt::box_value(true));
        }
    }
}
```

This is the XAML for the ContentDialog featured in this example.

XAML

```
<ContentDialog x:Name="saveEnergyDialog"
               PrimaryButtonText="Open battery saver settings"
               SecondaryButtonText="Ignore"
               Title="Battery saver is on.">
   <StackPanel>
      <TextBlock TextWrapping="WrapWholeWords">
         <LineBreak/><Run>Battery saver is on and you may
          not receive push notifications.</Run><LineBreak/>
         <LineBreak/><Run>You can choose to allow this app to work normally
```

```
            while in battery saver, including receiving push notifications.
</Run>
            <LineBreak/>
        </TextBlock>
        <CheckBox x:Name="dontAskAgainBox" Content="OK, got it."/>
    </StackPanel>
</ContentDialog>
```

Related topics

- Send a local tile notification
- Quickstart: Sending a push notification
- How to update a badge through push notifications
- How to request, create, and save a notification channel
- How to intercept notifications for running applications
- How to authenticate with the Windows Push Notification Service (WNS)
- Push notification service request and response headers
- Guidelines and checklist for push notifications
- Raw notifications

Quickstart: Sending a push notification (XAML)

Article • 10/25/2021

Your cloud server can send a push notification to your app through the Windows Push Notification Services (WNS). This procedure applies to tile, toast, badge, and raw push notifications.

Objective: To create and send a tile, toast, badge, or raw push notification.

Prerequisites

To understand this topic or to use the code it provides, you will need:

- A familiarity with HTTP communications.
- An authenticated cloud server. For more information, see How to authenticate with the Windows Push Notification Service (WNS).
- A registered channel over which your cloud server can communicate with your app. For more information, see How to request, create, and save a notification channel.
- An existing tile for your app, defined in your app's manifest, to receive the notification (unless you're sending a raw notification). For more information, see Quickstart: Creating a default tile using the Microsoft Visual Studio manifest editor.
- A familiarity with XML and its manipulation through Document Object Model (DOM) APIs.
- In the case of raw notifications, your app must be configured to receive raw notifications. For more information, see Quickstart: Intercepting push notifications for running apps and Quickstart: Creating and registering a raw notification background task.

Instructions

1. Include the necessary namespace references

The examples given in this topic can be used as-is, but require that your code include these namespace references:

```
C#
```

```
using System.Net;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Json;
using System.Web;
using System.Text;
```

2. Create an HTTP POST request

The `uri` parameter is the channel Uniform Resource Identifier (URI) requested by the app and passed to the cloud server. For more information, see How to request, create, and save a notification channel.

C#

```
HttpWebRequest request = HttpWebRequest.Create(uri) as HttpWebRequest;
request.Method = "POST";
```

3. Add the required headers

There are four required headers that must be included in all push notifications: X-WNS-Type, Content-Type, Content-Length, and Authorization.

- The X-WNS-Type header specifies whether this is a tile, toast, badge, or raw notification.
- The Content-Type is set depending on the value of the X-WNS-Type.
- The Content-Length gives the size of the included notification payload.
- The Authorization header specifies the authentication credential that allows you to send a push notification to this user over this channel.

The *accessToken* parameter of the Authorization header specifies the access token, stored on the server, that was received from WNS when the cloud server requested authentication. Without the access token, your notification will be rejected.

For a complete list of possible headers, see Push notification service request and response headers.

C#

```
request.Headers.Add("X-WNS-Type", notificationType);
request.ContentType = contentType;
request.Headers.Add("Authorization", String.Format("Bearer {0}",
accessToken.AccessToken));
```

4. Add the prepared content

As far as the HTTP request is concerned, the XML content of the notification is a data blob in the request body. For instance, no verification is made that the XML matches the X-WNS-Type specification. The content is specified as an XML payload and here is added to the request as a stream of bytes.

C#

```
byte[] contentInBytes = Encoding.UTF8.GetBytes(xml);

using (Stream requestStream = request.GetRequestStream())
    requestStream.Write(contentInBytes, 0, contentInBytes.Length);
```

5. Listen for a response from WNS that acknowledges receipt of the notification

> **Note**
>
> You will never receive a delivery confirmation for a notification, just an acknowledgment that it was received by WNS.

C#

```
using (HttpWebResponse webResponse = (HttpWebResponse)request.GetResponse())
    return webResponse.StatusCode.ToString();
```

6. Handle WNS response codes

There are many response codes that your app service can receive when it sends a notification. Some of these response codes are more common than others and can be easily dealt with in a catch block.

C#

```
catch (WebException webException)
{
    HttpStatusCode status =
((HttpWebResponse)webException.Response).StatusCode;
```

HttpStatusCode.Unauthorized: The access token you presented has expired. Get a new one and then try sending your notification again. Because your cached access token

expires after 24 hours, you can expect to get this response from WNS at least once a day. We recommend that you implement a maximum retry policy.

```
C#
```

```
    if (status == HttpStatusCode.Unauthorized)
    {
        GetAccessToken(secret, sid);
        return PostToWns(uri, xml, secret, sid, notificationType,
contentType);
    }
```

HttpStatusCode.Gone / HttpStatusCode.NotFound: The channel URI is no longer valid. Remove this channel from your database to prevent further attempts to send notification to it. The next time this user launches your app, request a new WNS channel. Your app should detect that its channel has changed, which should trigger the app to send the new channel URI to your app server. For more information, see How to request, create, and save a notification channel.

```
C#
```

```
    else if (status == HttpStatusCode.Gone || status ==
HttpStatusCode.NotFound)
    {
        return "";
    }
```

HttpStatusCode.NotAcceptable: This channel is being throttled by WNS. Implement a retry strategy that exponentially reduces the amount of notifications being sent in order to prevent being throttled again. Also, rethink scenarios that are causing your notifications to be throttled. You will provide a richer user experience by limiting the notifications you send to those that add true value.

```
C#
```

```
    else if (status == HttpStatusCode.NotAcceptable)
    {
        return "";
    }
```

Other response codes: WNS responded with a less common response code. Log this code to assist in debugging. See Push notification service request and response headers for a full list of WNS response codes.

```
C#
```

```
    else
    {
        string[] debugOutput = {
                                        status.ToString(),
                                        webException.Response.Headers["X-WNS-
Debug-Trace"],
                                        webException.Response.Headers["X-WNS-
Error-Description"],
                                        webException.Response.Headers["X-WNS-Msg-
ID"],
                                        webException.Response.Headers["X-WNS-
Status"]
                                    };
        return string.Join(" | ", debugOutput);
    }
```

7. Encapsulate the code into a single function

The following example packages the code given in the preceding steps into a single function. This function composes the HTTP POST request that contains a notification to be sent to WNS. By changing the value of the *type* parameter and adjusting additional headers, this code can be used for toast, tile, badge, or raw push notifications. You can use this function as part of your cloud server code.

Note that the error handling in this function includes the situation where the access token has expired. In this case, it calls another cloud server function that re-authenticates with WNS to obtain a new access token. It then makes a new call to the original function.

C#

```
// Post to WNS
public string PostToWns(string secret, string sid, string uri, string xml,
string notificationType, string contentType)
{
    try
    {
        // You should cache this access token.
        var accessToken = GetAccessToken(secret, sid);

        byte[] contentInBytes = Encoding.UTF8.GetBytes(xml);

        HttpWebRequest request = HttpWebRequest.Create(uri) as
HttpWebRequest;
        request.Method = "POST";
        request.Headers.Add("X-WNS-Type", notificationType);
        request.ContentType = contentType;
        request.Headers.Add("Authorization", String.Format("Bearer {0}",
```

```
accessToken.AccessToken));

        using (Stream requestStream = request.GetRequestStream())
            requestStream.Write(contentInBytes, 0, contentInBytes.Length);

        using (HttpWebResponse webResponse =
(HttpWebResponse)request.GetResponse())
            return webResponse.StatusCode.ToString();
    }

    catch (WebException webException)
    {
        HttpStatusCode status =
((HttpWebResponse)webException.Response).StatusCode;

        if (status == HttpStatusCode.Unauthorized)
        {
            // The access token you presented has expired. Get a new one and
then try sending
            // your notification again.

            // Because your cached access token expires after 24 hours, you
can expect to get
            // this response from WNS at least once a day.

            GetAccessToken(secret, sid);

            // We recommend that you implement a maximum retry policy.
            return PostToWns(uri, xml, secret, sid, notificationType,
contentType);
        }
        else if (status == HttpStatusCode.Gone || status ==
HttpStatusCode.NotFound)
        {
            // The channel URI is no longer valid.

            // Remove this channel from your database to prevent further
attempts
            // to send notifications to it.

            // The next time that this user launches your app, request a new
WNS channel.
            // Your app should detect that its channel has changed, which
should trigger
            // the app to send the new channel URI to your app server.

            return "";
        }
        else if (status == HttpStatusCode.NotAcceptable)
        {
            // This channel is being throttled by WNS.

            // Implement a retry strategy that exponentially reduces the
amount of
            // notifications being sent in order to prevent being throttled
```

```
again.

            // Also, consider the scenarios that are causing your
notifications to be throttled.
            // You will provide a richer user experience by limiting the
notifications you send
            // to those that add true value.

            return "";
        }
        else
        {
            // WNS responded with a less common error. Log this error to
assist in debugging.

            // You can see a full list of WNS response codes here:
            //
https://msdn.microsoft.com/library/windows/apps/hh868245.aspx#wnsresponsecod
es

            string[] debugOutput = {
                                        status.ToString(),
                                        webException.Response.Headers["X-WNS-
Debug-Trace"],
                                        webException.Response.Headers["X-WNS-
Error-Description"],
                                        webException.Response.Headers["X-WNS-
Msg-ID"],
                                        webException.Response.Headers["X-WNS-
Status"]
                                    };
            return string.Join(" | ", debugOutput);
        }
    }

    catch (Exception ex)
    {
        return "EXCEPTION: " + ex.Message;
    }
}

// Authorization
[DataContract]
public class OAuthToken
{
    [DataMember(Name = "access_token")]
    public string AccessToken { get; set; }
    [DataMember(Name = "token_type")]
    public string TokenType { get; set; }
}

private OAuthToken GetOAuthTokenFromJson(string jsonString)
{
    using (var ms = new MemoryStream(Encoding.Unicode.GetBytes(jsonString)))
    {
```

```
        var ser = new DataContractJsonSerializer(typeof(OAuthToken));
        var oAuthToken = (OAuthToken)ser.ReadObject(ms);
        return oAuthToken;
    }
}

protected OAuthToken GetAccessToken(string secret, string sid)
{
    var urlEncodedSecret = HttpUtility.UrlEncode(secret);
    var urlEncodedSid = HttpUtility.UrlEncode(sid);

    var body = String.Format("grant_type=client_credentials&client_id=
{0}&client_secret={1}&scope=notify.windows.com",
                             urlEncodedSid,
                             urlEncodedSecret);

    string response;
    using (var client = new WebClient())
    {
        client.Headers.Add("Content-Type", "application/x-www-form-
urlencoded");
        response =
client.UploadString("https://login.live.com/accesstoken.srf", body);
    }
    return GetOAuthTokenFromJson(response);
}
```

The following shows example content for an HTTP POST request for a toast push notification.

C#

```
POST https://db3.notify.windows.com/?
token=AgUAAADCQmTg7OMlCg%2fK0K8rBPcBqHuy%2b1rTSNPMuIzF6BtvpRdT7DM4j%2fs%2bNN
m8z5l1QKZMtyjByKW5uXqb9V7hIAeA3i8FoKR%2f49ZnGgyUkAhzix%2fuSuasL3jalk7562F4Bp
w%3d HTTP/1.1
Authorization: Bearer EgAaAQMAAAAEgAAACoAAPzCGedIbQb9vRfPF2Lxy3K//QZB79mLTgK
X-WNS-RequestForStatus: true
X-WNS-Type: wns/toast
Content-Type: text/xml
Host: db3.notify.windows.com
Content-Length: 196

<toast launch="">
  <visual lang="en-US">
    <binding template="ToastImageAndText01">
      <image id="1" src="World" />
      <text id="1">Hello</text>
    </binding>
  </visual>
</toast>
```

The following shows an example HTTP response, sent to the cloud server by WNS in response to the HTTP POST request.

```
C#

HTTP/1.1 200 OK
Content-Length: 0
X-WNS-DEVICECONNECTIONSTATUS: connected
X-WNS-STATUS: received
X-WNS-MSG-ID: 3CE38FF109E03A74
X-WNS-DEBUG-TRACE: DB3WNS4011534
```

Summary

In this Quickstart, you composed an HTTP POST request to send to WNS. WNS, in turn, delivers the notification to your app. By this point, you have registered your app, authenticated your cloud server with WNS, created XML content to define your notification, and sent that notification from your server to your app.

Related topics

- How to request, create, and save a notification channel
- Push notification service request and response headers
- Windows Push Notification Services (WNS) overview
- Get started with Mobile Services
- Push and periodic notifications sample

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

WNS Notification Priorities

Article • 03/17/2023

By setting a notification's priority with a simple header to WNS POST messages, you can control how notifications are delivered in battery sensitive situations.

Power on Windows

As more users are working only on battery powered devices, minimizing power usage has become a standard requirement for all apps. If apps consume more energy than the value they provide, users might uninstall the apps. While the Windows operating system reduces power usage on the battery where possible, it is the app's responsibility to work efficiently.

WNS priorities is one way to move non-critical work off the battery. The WNS priorities tell the system which notifications should be delivered instantly and which can wait until the device is plugged into a power source. With these hints, the system can deliver the notifications the exact time they are the most valuable to both the user and the app.

Power modes on the device

Every Windows device operates through a variety of power modes (battery, battery saver, and charge), and users expect different behaviors from apps in different power modes. When the device is on, all notifications should be delivered. In battery saver mode, only the most important notifications should be delivered. While the device is plugged in, sync or non-time critical operations can be completed.

Windows does not know which notifications are important to any user or app, so the system relies totally on apps to set the right priority for their notifications.

Priorities

There are four priorities available for an app to use when sending push notifications. The priority is set on individual notifications, allowing you to choose which notifications need to be delivered instantly (for example, an IM message) and which ones can wait (for example, contact photo updates).

The priorities are:

Expand table

Priority	User Override	Description	Example
High	Yes – user can block all notifications from an app OR can prevent an app from being throttled in battery saver mode.	The most important notifications that must be delivered right away in any circumstance when the device can receive notifications. Things like VoIP calls or critical alerts that should wake the device fall into this category.	VoIP calls, time-critical alerts
Medium	Yes – user can block all notifications from an app OR can prevent an app from being throttled in battery saver mode.	These are things that are not as important, things that don't need to happen right away, but users would be annoyed if they are not running in the background.	Secondary Email account sync, live tile updates.
Low	Yes – user can block all notifications from an app OR can prevent an app from being throttled in battery saver mode.	Notifications that only make sense when the user is using the device or when background activity makes sense. These are cached and not processed until the user signs in or plugs in their device.	Contact status (online/offline)

Note that many apps will have notifications of different priority throughout their lifecycle. Since the priority is set on a per-notification basis, this isn't an issue. A VoIP app can send a high priority notification for an incoming call and then follow it up with a low priority one when a contact comes online.

Setting the priority

Setting the priority on the notification request is done through an additional header on the POST request, `X-WNS-PRIORITY`. This is an integer value between 1 and 4 which maps to a priority:

Expand table

Priority Name	X-WNS-PRIORITY Value	Default for:
High	1	Toasts
Medium	2	Tiles and Badges
Low	3	Raw

To be backward compatible, setting a priority is not required. In case an app doesn't set the priority of their notifications, the system will provide a default priority. The defaults are shown in the chart above and match the behavior of existing versions of Windows.

Detailed listing of desktop behavior

If you are shipping your app across many different SKUs of Windows, it is normally best to follow the chart in the above section.

More specific recommended behaviors for each priority are listed below. This is not a guarantee that each device will work exactly according to the chart. OEMs are free to configure the behavior differently, but most are close to this chart.

Expand table

Device State	PRIORITY: High	PRIORITY: Medium	PRIORITY: Low	PRIORITY: Very Low
Screen On OR plugged in	Deliver	Deliver	Deliver	Deliver
Screen Off AND on battery	Deliver	If user exempted: deliver Else: cache	If user exempted: deliver Else: cache *	Cache
Battery Saver enabled	If user exempted: deliver Else: cache	If user exempted: deliver Else: cache	If user exempted: deliver Else: cache	Cache
On battery + battery saver enabled + screen off	If user exempted: deliver Else: cache	If user exempted: deliver Else: cache	If user exempted: deliver Else: cache	Cache

Note that low priority notifications will be delivered by default for screen off and battery only for Windows Phone based devices. This is to maintain compatibility with preexisting MPNS policy. Also note that the fourth and fifth rows are the same, just calling out different scenarios.

To exempt an app in battery saver, users must go to the "Battery Usage by App" in Settings and select "Allow the app to run background tasks." This user selection exempts the app from battery saver for high, medium, and low priority notifications. You can also call BackgroundExecutionManager API to programmatically ask for the user's permission.

Related topics

- Windows Push Notification Services (WNS) overview
- Requesting permission to run in the background

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Enterprise Firewall Configurations to Support WNS Traffic

Article • 10/04/2021

Background

Many enterprises use firewalls to block unwanted network traffic and ports; unfortunately, this can also block important things like Windows Notification Service communications. This means all notifications sent through WNS will be dropped under certain network configurations. To avoid this, network admins can add the list of approved WNS FQDNs or VIPs to their exemption list to allow the WNS traffic to pass through the firewall.

Proxy Support

> **Note**
>
> WNS Push Notifications *may not* support proxies. There are many different proxy setups and environments, and not all configurations are guaranteed work with WNS. For best results, the connection to WNS must be a direct connection, however, VPN interfaces can be used.

We welcome any feedback about proxy support. Please direct your feedback to https://aka.ms/windowsappsdk and file an issue for tracking interest in proxy support for WNS. Feel free to add the "area-Notifications" label to your issue for quicker visibility with the notifications team.

What information should be added to the allowlist

Below is a list that contains the FQDNs, VIPs, and IP address ranges used by the Windows Notification Service.

> **Important**

We strongly suggest that you allow list by FQDN, because these will not change. If you allow list by FQDN, you do not need to also allow the IP address ranges.

> **ⓘ Important**
>
> The IP address ranges will change periodically; because of this, they are not included on this page. If you want to see the list of IP ranges, you can download the file from Download Center: **Windows Notification Service (WNS) VIP and IP Ranges**. Please check back regularly to make sure you have the most up-to-date information.

FQDNs, VIPs, IPs, and Ports

Regardless of the method you choose from below, you'll need to allow network traffic to the listed destinations through **port 443**. Each of the elements in the following XML document is explained in the table that follows it (in Terms and Notations). The IP ranges were intentionally left out of this document to encourage you to use only the FQDNs as the FQDNs will remain constant. However, you can download the XML file containing the complete list from Download Center: Windows Notification Service (WNS) VIP and IP Ranges. New VIPs or IP ranges will be **effective one week after they are uploaded**.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<WNSPublicIpAddresses xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!-- This file contains the FQDNs, VIPs, and IP address ranges used by
the Windows Notification Service. A new text file will be uploaded every
time a new VIP or IP range is released in production.  Please copy the below
information and perform the necessary changes on your site. Endpoints in
CloudService nodes are used for cloud services to send notifications to WNS.
Endpoints in Client nodes are used by devices to receive notifications from
WNS. -->
    <CloudServiceDNS>
    <DNS FQDN="*.notify.windows.com"/>
    </CloudServiceDNS>
    <ClientDNS>
        <DNS FQDN="*.wns.windows.com"/>
        <DNS FQDN="*.notify.live.net"/>
    </ClientDNS>
    <CloudServiceIPs>
        <IpRange Subnet=""/>
        <!-- See the file in Download Center for the complete list of IP
ranges -->
```

```
    </CloudServiceIPs>
    <ClientIPsIPv4>
        <IpRange Subnet=""/>
        <!-- See the file in Download Center for the complete list of IP
ranges -->
    </ClientIPsIPv4>
    <IdentityServiceDNS>
        <DNS FQDN="login.microsoftonline.com"/>
        <DNS FQDN="login.live.com"/>
    </IdentityServiceDNS>
</WNSPublicIpAddresses>
```

Terms and notations

Below are explanations on the notations and elements used in the above XML snippet.

Expand table

Term	Explanation
Dot-decimal notation (i.e. 64.4.28.0/26)	Dot-decimal notation is a way to describe the range of IP addresses. For example, 64.4.28.0/26 means the first 26 bits of 64.4.28.0 are constant, while the last 6 bits are variable. In this case, the IPv4 range is 64.4.28.0 - 64.4.28.63.
ClientDNS	These are the Fully-Qualified Domain Name (FQDN) filters for the client devices (Windows PCs, desktops) receiving notifications from WNS. These must be allowed through the firewall in order for WNS clients to use the WNS Functionality. It is recommended to allow-list by the FQDNs instead of the IP/VIP ranges, since these will never change.
ClientIPsIPv4	These are the IPv4 addresses of the servers accessed by client devices (Windows PCs, desktops) receiving notifications from WNS.
CloudServiceDNS	These are the Fully-Qualified Domain Name (FQDN) filters for the WNS servers your cloud service will talk to send notifications to WNS. These must be allowed through the firewall in order for services to send WNS notifications. It is recommended to allow-list by the FQDNs instead of the IP/VIP ranges, since these will never change.
CloudServiceIPs	CloudServiceIPs are the IPv4 addresses of the servers used for cloud services to send notifications to WNS

Microsoft Push Notifications Service (MPNS) public IP ranges

If you are using the legacy notification service, MPNS, the IP address ranges that you will need to add to the allow list are available from Download Center: Microsoft Push Notifications Service (MPNS) Public IP ranges.

Related topics

- Quickstart: Sending a push notification
- How to request, create, and save a notification channel
- How to intercept notifications for running applications
- How to authenticate with the Windows Push Notification Service (WNS)
- Push notification service request and response headers
- Guidelines and checklist for push notifications

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Code generated by the push notification wizard

Article • 06/24/2021

By using a wizard in Visual Studio, you can generate push notifications from a mobile service that was created with Azure Mobile Services. The Visual Studio wizard generates code to help you get started. This topic explains how the wizard modifies your project, what the generated code does, how to use this code, and what you can do next to get the most out of push notifications. See Windows Push Notification Services (WNS) overview.

How the wizard modifies your project

The push notification wizard modifies your project in the following ways:

- Adds a reference to Mobile Services Managed Client (MobileServicesManagedClient.dll). Not applicable to JavaScript projects.
- Adds a file in a subfolder under services, and names the file push.register.cs, push.register.vb, push.register.cpp, or push.register.js.
- Creates a channels table on the database server for the mobile service. The table contains information that's required to send push notifications to app instances.
- Creates scripts for four functions: delete, insert, read and update.
- Creates a script with a custom API, notifyallusers.js, which sends a push notification to all clients.
- Adds a declaration to your App.xaml.cs, App.xaml.vb, or App.xaml.cpp file, or adds a declaration to a new file, service.js, for JavaScript projects. The declaration declares a MobileServiceClient object, which contains the information that's required to connect to the mobile service. You can access this MobileServiceClient object, which is named *MyServiceName*Client, from any page in your app by using the name App.*MyServiceName*Client.

The services.js file contains the following code:

JavaScript

```
var <mobile-service-name>Client = new
Microsoft.WindowsAzure.MobileServices.MobileServiceClient(
                "https://<mobile-service-name>.azure-mobile.net/",
                "<your client secret>");
```

Registration for push notifications

In push.register.*, the UploadChannel method registers the device to receive push notifications. The Store tracks installed instances of your app and provides the push notification channel. See PushNotificationChannelManager.

The client code is similar for both the JavaScript backend and the .NET backend. By default, when you add push notifications for a JavaScript backend service, a sample call to notifyAllUsers custom API is inserted into the UploadChannel method.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using Microsoft.WindowsAzure.MobileServices;
using Newtonsoft.Json.Linq;

namespace App2
{
    internal class mymobileservice1234Push
    {
        public async static void UploadChannel()
        {
            var channel = await
Windows.Networking.PushNotifications.PushNotificationChannelManager.CreatePu
shNotificationChannelForApplicationAsync();

            try
            {
                await
App.mymobileservice1234Client.GetPush().RegisterNativeAsync(channel.Uri);
                await
App.mymobileservice1234Client.InvokeApiAsync("notifyAllUsers");
            }
            catch (Exception exception)
            {
                HandleRegisterException(exception);
            }
        }

        private static void HandleRegisterException(Exception exception)
        {

        }
    }
}
```

VB

```
Imports Microsoft.WindowsAzure.MobileServices
Imports Newtonsoft.Json.Linq

Friend Class mymobileservice1234Push
    Public Shared Async Sub UploadChannel()
        Dim channel = Await
Windows.Networking.PushNotifications.PushNotificationChannelManager.CreatePu
shNotificationChannelForApplicationAsync()

        Try
            Await
App.mymobileservice1234Client.GetPush().RegisterNativeAsync(channel.Uri)
            Await
App.mymobileservice1234Client.GetPush().RegisterNativeAsync(channel.Uri, New
String() {"tag1", "tag2"})
            Await
App.mymobileservice1234Client.InvokeApiAsync("notifyAllUsers")
        Catch exception As Exception
            HandleRegisterException(exception)
        End Try
    End Sub

    Private Shared Sub HandleRegisterException(exception As Exception)

    End Sub
End Class
```

C++

```
#include "pch.h"
#include "services\mobile
services\mymobileservice1234\mymobileservice1234Push.h"

using namespace AzureMobileHelper;

using namespace web;
using namespace concurrency;

using namespace Windows::Networking::PushNotifications;

void mymobileservice1234Push::UploadChannel()
{

create_task(PushNotificationChannelManager::CreatePushNotificationChannelFor
ApplicationAsync()).
    then([] (PushNotificationChannel^ newChannel)
    {
        return
mymobileservice1234MobileService::GetClient().get_push().register_native(new
Channel->Uri->Data());
    }).then([]()
```

```
    {
        return
mymobileservice1234MobileService::GetClient().invoke_api(L"notifyAllUsers");
    }).then([](task<json::value> result)
    {
        try
        {
            result.wait();
        }
        catch(...)
        {
            HandleExceptionsComingFromTheServer();
        }
    });
}

void mymobileservice1234Push::HandleExceptionsComingFromTheServer()
{
}
```

JavaScript

```
(function () {
    "use strict";

    var app = WinJS.Application;
    var activation = Windows.ApplicationModel.Activation;

    app.addEventListener("activated", function (args) {
        if (args.detail.kind == activation.ActivationKind.launch) {

Windows.Networking.PushNotifications.PushNotificationChannelManager.createPu
shNotificationChannelForApplicationAsync()
                .then(function (channel) {

mymobileserviceclient1234Client.push.registerNative(channel.Uri, new
Array("tag1", "tag2"))
                    return
mymobileservice1234Client.push.registerNative(channel.uri);
                })
                .done(function (registration) {
                    return
mymobileservice1234Client.invokeApi("notifyAllUsers");
                }, function (error) {
                    // Error

                });
        }
    });
})();
```

Push notification tags provide a way to restrict notifications to a subset of clients. You can use the for the registerNative method (or RegisterNativeAsync) method to either register for all push notifications without specifying tags, or you can register with tags by providing the second argument, an array of tags. If you register with one or more tags, you only receive notifications that match those tags.

Server-side scripts (JavaScript backend only)

For mobile services that use the JavaScript backend, the server-side scripts run when delete, insert, read, or update operations occur. The scripts don't implement these operations, but they run when a call from the client to the Windows Mobile REST API triggers these events. The scripts then pass control onto the operations themselves by calling request.execute or request.respond to issue a response to the calling context. See Azure Mobile Services REST API Reference.

A variety of functions are available in the server-side script. See Register table operations in Azure Mobile Services. For a reference to all available functions, see Mobile Services server script reference.

The following custom API code in Notifyallusers.js is also created:

JavaScript

```
exports.post = function(request, response) {
    response.send(statusCodes.OK,{ message : 'Hello World!' })

    // The following call is for illustration purpose only
    // The call and function body should be moved to a script in your app
    // where you want to send a notification
    sendNotifications(request);
};

// The following code should be moved to appropriate script in your app
where notification is sent
function sendNotifications(request) {
    var payload = '<?xml version="1.0" encoding="utf-8"?><toast><visual>
<binding template="ToastText01">' +
        '<text id="1">Sample Toast</text></binding></visual></toast>';
    var push = request.service.push;
    push.wns.send(null,
        payload,
        'wns/toast', {
            success: function (pushResponse) {
                console.log("Sent push:", pushResponse);
            }
        });
}
```

The sendNotifications function sends a single notification as a toast notification. You can also use other types of push notifications.

Tip For information about how to get help while editing scripts, see Enabling IntelliSense for server-side JavaScript.

Push notification types

Windows supports notifications that aren't push notifications. For general information about notifications, see Choosing a notification delivery method.

Toast notifications are easy to use, and you can review an example in the Insert.js code on the channel's table that's generated for you. If you plan to use tile or badge notifications, you must create an XML template for the tile and badge, and you must specify the encoding of packaged information in the template. See Working with tiles, badges, and toast notifications.

Because Windows responds to push notifications, it can handle most of these notifications when the app isn't running. For example, a push notification could let a user know when a new mail message is available even when the local mail app isn't running. Windows handles a toast notification by displaying a message, such as the first line of a text message. Windows handles a tile or badge notification by updating an app's live tile to reflect the number of new mail messages. In this way, you can prompt users of your app to check it for new information. Your app can receive raw notifications when it's running, and you can use them to send data to your app. If your app isn't running, you can set up a background task to monitor push notifications.

You should use push notifications according to the guidelines for Windows apps, because those notifications use up a user's resources and can be distracting if overused. See Guidelines and checklist for push notifications.

If you're updating live tiles with push notifications, you should also follow the guidelines in Guidelines and checklist for tiles and badges.

Next steps

Using the Windows Push Notification Services (WNS)

You can call Windows Push Notification Services (WNS) directly if Mobile Services doesn't provide enough flexibility, if you want to write your server code in C# or Visual

Basic, or if you already have a cloud service and you want to send push notifications from it. By calling WNS directly, you can send push notifications from your own cloud service, such as a worker role that monitors data from a database or another web service. Your cloud service must authenticate with WNS to send push notifications to your apps. See How to authenticate with the Windows Push Notification Service (JavaScript) or (C#/C++/VB).

You can also send push notifications by running a scheduled task in your mobile service. See Schedule recurring jobs in Mobile Services.

Warning Once you've run the push notification wizard once, don't run the wizard a second time to add registration code for another mobile service. Running the wizard more than once per project generates code that results in overlapping calls to the **CreatePushNotificationChannelForApplicationAsync** method, which leads to a runtime exception. If you want to register for push notifications for more than one mobile service, run the wizard once and then rewrite the registration code to ensure that calls to **CreatePushNotificationChannelForApplicationAsync** do not run at the same time. For example, you can accomplish this by moving the wizard-generated code in push.register.* (including the call to **CreatePushNotificationChannelForApplicationAsync**) outside of the OnLaunched event, but the specifics of this will depend on your app's architecture.

Related topics

- Windows Push Notification Services (WNS) overview
- Raw notification overview
- Connecting to Windows Azure Mobile Services (JavaScript)
- Connecting to Windows Azure Mobile Services (C#/C++/VB)
- Quickstart: Adding push notifications for a mobile service (JavaScript)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

more information, see our contributor guide.

Open a documentation issue

Provide product feedback

How to request, create, and save a notification channel

Article • 10/25/2021

You can open a channel Uniform Resource Identifier (URI) over which your app can receive push notifications. You can then send the channel to your server which uses it to send push notifications, and close it when you no longer need it. A channel is a unique address that represents a single user on a single device, for a specific app or secondary tile.

You should request a new channel each time that your app is launched, and update the cloud server when the URI changes. For more details, see Remarks.

> **Important**
>
> Notification channels automatically expire after 30 days.

What you need to know

Technologies

- Windows Runtime

Prerequisites

- Familiarity with push notification and Windows Push Notification Services (WNS) concepts, requirements, and operation. These are discussed in the Windows Push Notification Services (WNS) overview.

Instructions

Step 1: Add namespace declarations

Windows.UI.Notifications includes the toast APIs.

```
C#
```

```
using Windows.UI.Notifications;
using Windows.Data.Xml.Dom;
using Windows.Networking.PushNotifications;
```

Step 2: Request a channel URI

This example requests a channel URI. The request is made to the Notification Client Platform, which in turn requests the channel URI from WNS. When the request is complete, the returned value is a PushNotificationChannel object that contains the URI.

C#

```
PushNotificationChannel channel = null;

try
{
    channel = await
PushNotificationChannelManager.CreatePushNotificationChannelForApplicationAs
ync();
}

catch (Exception ex)
{
    // Could not create a channel.
}
```

Step 3: Send the channel URI to your server

The channel URI is packaged in an HTTP POST request and sent to the server.

> **ⓘ Important**
>
> You should send this information to your server in a secure manner. You should require the app to authenticate itself with the server when it transmits the channel URI. Encrypt the information and use a secure protocol such as HTTPS.

C#

```
String serverUrl = "http://www.contoso.com";

// Create the web request.
HttpWebRequest webRequest =
(HttpWebRequest)HttpWebRequest.Create(serverUrl);
webRequest.Method = "POST";
```

```
webRequest.ContentType = "application/x-www-form-urlencoded";
byte[] channelUriInBytes = System.Text.Encoding.UTF8.GetBytes("ChannelUri="
+ channel.Uri);

// Write the channel URI to the request stream.
Stream requestStream = await webRequest.GetRequestStreamAsync();
requestStream.Write(channelUriInBytes, 0, channelUriInBytes.Length);

try
{
    // Get the response from the server.
    WebResponse response = await webRequest.GetResponseAsync();
    StreamReader requestReader = new
StreamReader(response.GetResponseStream());
    String webResponse = requestReader.ReadToEnd();
}

catch (Exception ex)
{
    // Could not send channel URI to server.
}
```

Remarks

Requesting channels

You should request a new channel each time your app is invoked, by using the following logic:

1. Request a channel.
2. Compare the new channel with your previous channel. If the channel is the same, you don't need to take any further action. Note that this requires your app to store the channel locally each time the app successfully sends it to your service, so that you have the channel to compare against later.
3. If the channel has changed, send the new channel to your web service. Include error handling logic that always sends a new channel in the following cases:

    - Your app has never sent a channel to the web service before.
    - Your app's last attempt to send the channel to the web service was not successful.

Different calls to the CreatePushNotificationChannelForApplicationAsync method do not always return a different channel. If the channel has not changed since the last call, your app should conserve effort and Internet traffic by not resending that same channel to your service. An app can have multiple valid channel URIs at the same time. Because

each unique channel remains valid until it expires, there is no harm in requesting a new channel because it does not affect the expiration time of any previous channels.

By requesting a new channel each time your app is invoked, you maximize your chances of always having access to a valid channel. This is particularly important if it is vital to your tile or toast scenario that content always be live. If you are concerned that a user might not run your app more than once every 30 days, you can implement a background task to execute your channel request code on a regular basis.

Handling errors in channel requests

The call to the **CreatePushNotificationChannelForApplicationAsync** method can fail if the Internet is not available. To handle this, add retry logic to the code shown in step 2. We recommend three attempts with a 10-second delay between each unsuccessful attempt. If all three attempts fail, your app should wait until the next time the user launches it to try again.

Closing channels

Your app can immediately stop the delivery of notifications on all channels by calling the **PushNotificationChannel.Close** method. While it will not be common for your app to do so, there might be certain scenarios in which you want to stop all notification delivery to your app. For instance, if your app has the concept of user accounts and a user logs out of that app, it is reasonable to expect that the tile no longer shows that user's personal information. To successfully clear the tile of content and stop the delivery of notifications, you should do the following:

1. Stop all tile updates by calling the **PushNotificationChannel.Close** method on any of your notification channels that are delivering tile, toast, badge or raw notifications to a user. Calling the **Close** method ensures that no further notifications for that user can be delivered to the client.
2. Clear the contents of the tile by calling the **TileUpdater.Clear** method to remove the previous user's data from the tile.

Push notification service request and response headers (Windows Runtime apps)

Article • 05/03/2024

This topic describes the service-to-service web APIs and protocols required to send a push notification.

See the Windows Push Notification Services (WNS) overview for a conceptual discussion of push notification and WNS concepts, requirements, and operation.

Requesting and receiving an access token

This section describes the request and response parameters involved when you authenticate with the WNS.

Access token request

An HTTP request is sent to WNS to authenticate the cloud service and retrieve an access token in return. The request is issued to `https://login.live.com/accesstoken.srf` using Secure Sockets Layer (SSL).

Access token request parameters

The cloud service submits these required parameters in the HTTP request body, using the "application/x-www-form-urlencoded" format. You must ensure that all parameters are URL encoded.

Expand table

Parameter	Required	Description
grant_type	TRUE	Must be set to `client_credentials`.
client_id	TRUE	Package security identifier (SID) for your cloud service as assigned when you registered your app with the Microsoft Store.
client_secret	TRUE	Secret key for your cloud service as assigned when you registered your app with the Microsoft Store.

Parameter	Required	Description
scope	TRUE	Must be set to `notify.windows.com`

Access token response

WNS authenticates the cloud service and, if successful, responds with a "200 OK", including the access token. Otherwise, WNS responds with an appropriate HTTP error code as described in the OAuth 2.0 protocol draft.

Access token response parameters

An access token is returned in the HTTP response if the cloud service successfully authenticated. This access token can be used in notification requests until it expires. The HTTP response uses the "application/json" media type.

Expand table

Parameter	Required	Description
access_token	TRUE	The access token that the cloud service will use when it sends a notification.
token_type	FALSE	Always returned as `bearer`.

Response code

Expand table

HTTP response code	Description
200 OK	The request was successful.
400 Bad Request	The authentication failed. See the OAuth draft Request for Comments (RFC) for response parameters.

Example

The following shows an example of a successful authentication response:

```
JSON
```

```
HTTP/1.1 200 OK
Cache-Control: no-store
Content-Length: 422
Content-Type: application/json

{
    "access_token":"EgAcAQMAAAAALYAAY/c+Huwi3Fv4Ck10UrKNmtxRO6Njk2MgA=",
    "token_type":"bearer",
    "expires_in": 86400
}
```

Sending a notification request and receiving a response

This section describes the headers involved in an HTTP request to WNS to deliver a notification and those involved in the reply.

- Send notification request
- Send notification response
- Unsupported HTTP features

Send notification request

When sending a notification request, the calling app makes an HTTP request over SSL, addressed to the channel Uniform Resource Identifier (URI). "Content-Length" is a standard HTTP header that must be specified in the request. All other standard headers are either optional or not supported.

In addition, the custom request headers listed here can be used in the notification request. Some headers are required while others are optional.

Request parameters

Expand table

Header name	Required	Description
Authorization	TRUE	Standard HTTP authorization header used to authenticate your notification request. Your cloud service provides its access token in this header.
Content-Type	TRUE	Standard HTTP authorization header. For toast, tile, and badge notifications, this header must be set to `text/xml`. For raw

Header name	Required	Description
		notifications, this header must be set to `application/octet-stream`.
Content-Length	TRUE	Standard HTTP authorization header to denote the size of the request payload.
X-WNS-Type	TRUE	Defines the notification type in the payload: tile, toast, badge, or raw.
X-WNS-Cache-Policy	FALSE	Enables or disables notification caching. This header applies only to tile, badge, and raw notifications.
X-WNS-RequestForStatus	FALSE	Requests device status and WNS connection status in the notification response.
X-WNS-Tag	FALSE	String used to provide a notification with an identifying label, used for tiles that support the notification queue. This header applies only to tile notifications.
X-WNS-TTL	FALSE	Integer value, expressed in seconds, that specifies the time to live (TTL).
MS-CV	FALSE	Correlation Vector value used for your request.

Important notes

- Content-Length and Content-Type are the only standard HTTP headers that are included in the notification delivered to the client, regardless of whether other standard headers were included in the request.
- All other standard HTTP headers are either ignored or return an error if the functionality is not supported.
- Starting in February 2023, WNS will cache only one tile notification when the device is offline.

Authorization

The authorization header is used to specify the credentials of the calling party, following the OAuth 2.0 authorization method for bearer tokens.

The syntax consists of a string literal "Bearer", followed by a space, followed by your access token. This access token is retrieved by issuing the access token request described above. The same access token can be used on subsequent notification requests until it expires.

This header is required.

```
JSON

Authorization: Bearer <access-token>
```

X-WNS-Type

These are the notification types supported by WNS. This header indicates the type of notification and how WNS should handle it. After the notification reaches the client, the actual payload is validated against this specified type. This header is required.

```
JSON

X-WNS-Type: wns/toast | wns/badge | wns/tile | wns/raw
```

Expand table

Value	Description
wns/badge	A notification to create a badge overlay on the tile. The Content-Type header included in the notification request must be set to `text/xml`.
wns/tile	A notification to update the tile content. The Content-Type header included in the notification request must be set to `text/xml`.
wns/toast	A notification to raise a toast on the client. The Content-Type header included in the notification request must be set to `text/xml`.
wns/raw	A notification which can contain a custom payload and is delivered directly to the app. The Content-Type header included in the notification request must be set to `application/octet-stream`.

X-WNS-Cache-Policy

When the notification target device is offline, WNS will cache one badge, one tile, and one toast notification for each channel URI. By default, raw notifications are not cached, but if raw notification caching is enabled, one raw notification is cached. Items are not held in the cache indefinitely and will be dropped after a reasonable period of time. Otherwise, the cached content is delivered when the device next comes online.

```
JSON
```

```
X-WNS-Cache-Policy: cache | no-cache
```

Expand table

Value	Description
cache	Default. Notifications will be cached if the user is offline. This is the default setting for tile and badge notifications.
no-cache	The notification will not be cached if the user is offline. This is the default setting for raw notifications.

X-WNS-RequestForStatus

Specifies whether the response should include the device status and WNS connection status. This header is optional.

JSON

```
X-WNS-RequestForStatus: true | false
```

Expand table

Value	Description
true	Return the device status and notification status in the response.
false	Default. Do not return the device status and notification status.

X-WNS-Tag

Assigns a tag label to a notification. The tag is used in the replacement policy of the tile in the notification queue when the app has opted for notification cycling. If a notification with this tag already exists in the queue, a new notification with the same tag takes its place.

> **Note**
>
> This header is optional and used only when sending tile notifications.

JSON

```
X-WNS-Tag: <string value>
```

Expand table

Value	Description
string value	An alphanumeric string of no more than 16 characters.

X-WNS-TTL

Specifies the TTL (expiration time) for a notification. This is not typically needed, but can be used if you want to ensure that your notifications are not displayed later than a certain time. The TTL is specified in seconds and is relative to the time that WNS receives the request. When a TTL is specified, the device will not display the notification after that time. Note that this could result in the notification never being shown at all if the TTL is too short. In general, an expiration time will be measured in at least minutes.

This header is optional. If no value is specified, the notification does not expire and will be replaced under the normal notification replacement scheme.

X-WNS-TTL: `<integer value>`

Expand table

Value	Description
integer value	The life span of the notification, in seconds, after WNS receives the request.

X-WNS-SuppressPopup

> **Note**
>
> For Windows Phone Store apps, you have the option to suppress a toast notification's UI, instead sending the notification directly to the action center. This lets your notification be delivered silently, a potentially superior option for less urgent notifications. This header is optional and only used on Windows Phone channels. If you include this header on a Windows channel, your notification will be dropped and you will receive an error response from WNS.

X-WNS-SuppressPopup: true | false

Expand table

Value	Description
true	Send the toast notification directly to the action center; do not raise the toast UI.
false	Default. Raise the toast UI as well as adding the notification to the action center.

X-WNS-Group

> **Note**
>
> The action center for Windows Phone Store apps can display multiple toast notifications with the same tag only if they are labelled as belonging to different groups. For example, consider a recipe book app. Each recipe would be identified by a tag. A toast that contains a comment on that recipe would have the recipe's tag, but a comment group label. A toast that contains a rating for that recipe would again have that recipe's tag, but would have a rating group label. Those different group labels would allow both toast notifications to be shown at once in the action center. This header is optional.

X-WNS-Group: `<string value>`

Expand table

Value	Description
string value	An alphanumeric string of no more than 16 characters.

X-WNS-Match

> **Note**
>
> Used with the HTTP DELETE method to remove a specific toast, a set of toasts (by either tag or group), or all toasts from the action center for Windows Phone Store apps. This header can specify a group, a tag, or both. This header is required in an HTTP DELETE notification request. Any payload included with this notification request is ignored.

X-WNS-Match: type:wns/toast;group=`<string value>`;tag=`<string value>` | type:wns/toast;group=`<string value>` | type:wns/toast;tag=`<string value>` | type:wns/toast;all

Expand table

Value	Description
type:wns/toast;group=`<string value>`;tag=`<string value>`	Remove a single notification labelled with both the specified tag and group.
type:wns/toast;group=`<string value>`	Remove all notifications labelled with the specified group.
type:wns/toast;tag=`<string value>`	Remove all notifications labelled with the specified tag.
type:wns/toast;all	Clear all of your app's notifications from the action center.

Send notification response

After WNS processes the notification request, it sends an HTTP message in response. This section discusses the parameters and headers that can be found in that response.

Response parameters

Expand table

Header name	Required	Description
X-WNS-Debug-Trace	FALSE	Debugging information that should be logged to help troubleshoot issues when reporting a problem.
X-WNS-DeviceConnectionStatus	FALSE	The device status, returned only if requested in the notification request through the X-WNS-RequestForStatus header.
X-WNS-Error-Description	FALSE	A human-readable error string that should be logged to help with debugging.
X-WNS-Msg-ID	FALSE	A unique identifier for the notification, used for debugging purposes. When reporting a problem, this information should be logged to help in troubleshooting.

Header name	Required	Description
X-WNS-Status	FALSE	Indicates whether WNS successfully received and processed the notification. When reporting a problem, this information should be logged to help in troubleshooting.
MS-CV	FALSE	Debugging information that should be logged to help troubleshoot issues when reporting a problem.

X-WNS-Debug-Trace

This header returns useful debugging information as a string. We recommend that this header be logged to help developers debug issues. This header, together with the X-WNS-Msg-ID header and MS-CV, are required when reporting an issue to WNS.

X-WNS-Debug-Trace: `<string value>`

Expand table

Value	Description
string value	An alphanumeric string.

X-WNS-DeviceConnectionStatus

This header returns the device status to the calling application, if requested in the X-WNS-RequestForStatus header of the notification request.

X-WNS-DeviceConnectionStatus: connected | disconnected | tempdisconnected

Expand table

Value	Description
connected	The device is online and connected to WNS.
disconnected	The device is offline and not connected to WNS.
tempconnected (deprecated)	The device temporarily lost connection to WNS, for instance when a 3G connection is dropped or the wireless switch on a laptop is thrown. It is seen by the Notification Client Platform as a temporary interruption rather than an intentional disconnection.

X-WNS-Error-Description

This header provides a human-readable error string that should be logged to help with debugging.

X-WNS-Error-Description: `<string value>`

Expand table

Value	Description
string value	An alphanumeric string.

X-WNS-Msg-ID

This header is used to provide the caller with an identifier for the notification. We recommended that this header be logged to help debug issues. This header, together with the X-WNS-Debug-Trace and MS-CV, are required when reporting an issue to WNS.

X-WNS-Msg-ID: `<string value>`

Expand table

Value	Description
string value	An alphanumeric string of no more than 16 characters.

X-WNS-Status

This header describes how WNS handled the notification request. This can be used rather than interpreting response codes as success or failure.

X-WNS-Status: received | dropped | channelthrottled

Expand table

Value	Description
received	The notification was received and processed by WNS. **Note:**This does not guarantee that the device received the notification.
dropped	The notification was explicitly dropped because of an error or because the client has explicitly rejected these notifications. Toast notifications will also be

Value	Description
	dropped if the device is offline.
channelthrottled	The notification was dropped because the app server exceeded the rate limit for this specific channel.

MS-CV

This header provides a Correlation Vector related to the request which is primarily used for debugging. If a CV is provided as part of the request then WNS will use this value, else WNS will generate and respond back with a CV. This header, together with the X-WNS-Debug-Trace and X-WNS-Msg-ID header, are required when reporting an issue to WNS.

> **Important**
>
> Please generate a new CV for each push notification request if you are providing your own CV.

MS-CV: `<string value>`

Expand table

Value	Description
string value	Follows the Correlation Vector standard

Response codes

Each HTTP message contains one of these response codes. WNS recommends that developers log the response code for use in debugging. When developers report an issue to WNS, they are required to provide response codes and header information.

Expand table

HTTP response code	Description	Recommended action
200 OK	The notification was accepted by WNS.	None required.

HTTP response code	Description	Recommended action
400 Bad Request	One or more headers were specified incorrectly or conflict with another header.	Log the details of your request. Inspect your request and compare against this documentation.
401 Unauthorized	The cloud service did not present a valid authentication ticket. The OAuth ticket may be invalid.	Request a valid access token by authenticating your cloud service using the access token request.
403 Forbidden	The cloud service is not authorized to send a notification to this URI even though they are authenticated.	The access token provided in the request does not match the credentials of the app that requested the channel URI. Ensure that your package name in your app's manifest matches the cloud service credentials given to your app in the Dashboard.
404 Not Found	The channel URI is not valid or is not recognized by WNS.	Log the details of your request. Do not send further notifications to this channel; notifications to this address will fail.
405 Method Not Allowed	Invalid method (GET, CREATE); only POST (Windows or Windows Phone) or DELETE (Windows Phone only) is allowed.	Log the details of your request. Switch to using HTTP POST.
406 Not Acceptable	The cloud service exceeded its throttle limit.	Please send your request after the Retry-After header value in the response
410 Gone	The channel expired.	Log the details of your request. Do not send further notifications to this channel. Have your app request a new channel URI.
410 Domain Blocked	The sending domain has been blocked by WNS.	Do not send further notifications to this channel. The sending domain has been blocked by WNS for abusing push notifications.
413 Request Entity Too Large	The notification payload exceeds the 5000 byte size limit.	Log the details of your request. Inspect the payload to ensure it is within the size limitations.
500 Internal Server Error	An internal failure caused notification delivery to fail.	Log the details of your request. Report this issue through the developer forums.
503 Service Unavailable	The server is currently unavailable.	Log the details of your request. Report this issue through the developer forums. If the Retry-After header is observed then please send your

HTTP response code	Description	Recommended action
		request after the Retry-After header value in the response.

Unsupported HTTP features

The WNS Web Interface supports HTTP 1.1 but does not support the following features:

- Chunking
- Pipelining (POST is not idempotent)
- Although supported, developers should disable Expect-100 as that introduces latency when sending a notification.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Troubleshooting WNS push notifications

Article • 10/25/2021

This topic discusses initial troubleshooting steps you should take when you encounter problems with tile, toast, and badge notifications, including the various notification methods: local, push, periodic, and scheduled notifications.

Troubleshooting specific errors

This section addresses some common errors you might encounter while working with push notifications.

- Check your event logs
- Push notification receives a "200 OK" response, but does not display
- Push notification returns a code other than "200 OK"
- Errors when attempting to create a push notification channel

Check your event logs

If tile or toast push notifications are not displaying as expected, have a look at the event logs.

- **If the notification is received but not shown**: Launch the Event Viewer and examine the *Microsoft-Windows-TWinUI/Operational* log under Applications and Services\Microsoft\Windows\Apps.
- **If the notification is not received at all**: Launch the Event Viewer and examine the Operational log under Applications and Services\Microsoft\Windows\PushNotifications-Platform.

Push notification receives a "200 OK" response, but does not display

If Windows Push Notification Services (WNS) returns a "200 OK" response, then it will deliver the notification to the client if the client is online. If you have verified that the client is online but not displaying the notification, walk through these steps:

- **Cause**: XML errors in the notification content.

  Fix: Verify the basic XML syntax and make sure that your XML is complete and correct. Some common points of failure in the XML content include the following:

    - Case sensitivity. Tag names, attribute names, and attribute values are all case sensitive. Be sure that your XML has the correct casing.
    - A **binding** element must be provided for each supported tile format. You should provide a **binding** element for each of the tile sizes you support in each notification you send.
    - Text strings should not contain reserved XML characters. For example, you cannot italicize tile or toast strings by including and tags. If you intend to show the literal characters "", they should be properly escaped. For more information on escape characters in XML, see XML Character Entities and XAML.
    - The values supplied for the *lang* attributes must conform to the ITEF BCP 47 specification.
    - XML strings sent through push notifications should use the UTF-8 encoding.
    - If you include an **image** element in your XML payload with a non-empty src attribute, you must be sure to include a reference to a valid image or the notification will be dropped.

- **Cause**: Improper use of push notification API parameters

  Fix: See the API documentation in the **Windows.Networking.PushNotifications** namespace for specifics.

- **Cause**: Header type does not match notification content. If the X-WNS-Type header is not set to a value—tile, badge, or toast—that corresponds to the notification template specified in the payload, the notification will not be displayed. This mismatch will cause an error on the client and the notification will be dropped.

  Fix: Refer to Push notification service request and response headers to ensure that your app server is using the correct value for the X-WNS-Type header.

- **Cause**: The time to live (TTL) value, set in the X-WNS-TTL header, is too small.

  Fix: Provide a larger TTL value, being aware that the value is given in seconds.

If you still do not see your notification displayed after addressing the items in the previous steps, see the troubleshooting steps for local notifications in the Local tile notification is not displayed section of this topic for further suggestions.

Push notification returns a code other than "200 OK"

If WNS doesn't return "200 OK", your notification will not be delivered to the client. If the return code is in the 400s, then you, as a developer, should be able to fix the issue.

> **Note**
>
> For errors not specifically listed here, see **COM Error Codes (WPN, MBN, P2P, Bluetooth)**.

- Notification request returns "400 Bad Request"
- Notification request returns "401 Unauthorized"
- Notification request returns "401 Unauthorized", token is expired
- Notification request returns "403 Forbidden"
- Notification request returns "404 Not Found"
- Notification request returns "406 Not Acceptable"
- Notification request returns "410 Gone"

Notification request returns "400 Bad Request"

- **Cause**: The use of one or more WNS headers could be incorrect or the HTTP request was invalid.

  Fix: Refer to Push notification service request and response headers to ensure that your app server is using all custom headers as described.

Notification request returns "401 Unauthorized"

- **Cause**: Your app server must use the correct Package Security Identifier (Package SID) and secret key given to you when registered your app. If you have recently changed your secret key in the Windows Store Dashboard, you will also need to update your app server.

  Fix: Visit the Windows StoreDashboard to verify your Package SID and secret.

Notification request returns "401 Unauthorized", token is expired

- **Cause**: An access token has a finite lifetime. If you send a notification with an expired access token, your app server's credentials are invalid and the notification cannot be sent.

  Fix: Request a new access token from WNS by authenticating with WNS using your Package Security Identifier (Package SID) and secret key. For more information, see the Windows Push Notification Services (WNS) overview

Notification request returns "403 Forbidden"

- **Cause**: This error occurs when the access token that you presented does not match the credentials required to send notifications to the corresponding channel URL. Every app must be registered with the Windows Store to receive credentials for its app server. For each app, only the credentials provided by the Windows Store can be used to send notifications to that app and they can be used only for that particular app.

  Fix: Log into the Windows Store Dashboard with your developer account. Select your app and click "Advanced Features" -> "Manage your cloud service settings". Select "Identifying your app" to read instructions on updating your app manifest to match your cloud service credentials.

Notification request returns "404 Not Found"

- **Cause**: This error typically means that the channel URL is not formed correctly. The channel URL must never be tampered with or modified when you send a notification to WNS. The channel URL should always be treated as an opaque string—you never need to examine or even know its content.

  Fix: Verify that your code is not modifying the channel URL either by changing one or more of its characters or changing its encoding.

Notification request returns "406 Not Acceptable"

- **Cause**: WNS has protective policies in place to prevent malicious apps from negatively impacting the service for other users and developers. An excessive number of notifications in too short a time period can result in WNS explicitly dropping notifications.

  Fix: Review your notification frequency to see if it can be decreased or optimized to produce a better user experience.

Notification request returns "410 Gone"

- **Cause**: The channel URL has expired. No further notifications can be sent until your app runs and requests a new channel URL.

  Fix: Your Windows Store app should request a channel URL each time it is launched. The channel URL that it is assigned is not guaranteed to remain the

same. If the URL has changed, the client should update the information on its cloud server.

Errors when attempting to create a push notification channel

- Creating a notification channel results in an ERROR_NO_NETWORK error
- Creating a notification channel results in an WPN_E_CLOUD_INCAPABLE error
- Creating a notification channel results in an WPN_E_INVALID_APP error

> ⓘ **Note**
>
> For errors not specifically listed here, see **COM Error Codes (WPN, MBN, P2P, Bluetooth)**.

Creating a notification channel results in an ERROR_NO_NETWORK error

- **Cause**: WNS requires an Internet connection to create a notification channel.

  Fix: Check your Internet connectivity.

Creating a notification channel results in an WPN_E_CLOUD_INCAPABLE error

- **Cause**: Your app has not declared the Internet capability in its app manifest (package.appxmanifest).

  Fix: Ensure that your app manifest has declared Internet capability. In the Visual Studio manifest editor, you will find this option under the Capabilities tab as **Internet (Client)**. For more information, see **Capabilities**.

Creating a notification channel results in an WPN_E_INVALID_APP error

- **Cause**: Your app must use a valid package name. If you have not received one yet, you can get it through the Windows Store portal under "Advanced Features".

  Fix: For details on retrieving a Package Security Identifier (PKSID) for your Windows Store app, see Windows Push Notification Services (WNS) overview.

Reporting an issue

If you have tried the solutions suggested in this topic and have not resolved your issue, post a message on the Microsoft forums to discuss it with both Microsoft developers and other interested parties.

For push notifications, in addition to a description of the problem, you might be asked to provide your channel URL and an example of the response you received from WNS, including both the HTTP error codes and HTTP headers. There are specific headers that your app server should be logging when reporting an issue. For more information, see Push notification service request and response headers.

Raw notification overview

Article • 03/17/2023

Raw notifications are short, general purpose push notifications. They are strictly instructional and do not include a UI component. As with other push notifications, the Windows Push Notification Services (WNS) feature delivers raw notifications from your cloud service to your app.

You can use raw notifications for a variety of purposes, including to trigger your app to run a background task if the user has given the app permission to do so. By using WNS to communicate with your app, you can avoid the processing overhead of creating persistent socket connections, sending HTTP GET messages, and other service-to-app connections.

> **Important**
>
> To understand raw notifications, it's best to be familiar with the concepts discussed in the Windows Push Notification Services (WNS) overview.

As with toast, tile, and badge push notifications, a raw notification is pushed from your app's cloud service over an assigned channel Uniform Resource Identifier (URI) to WNS. WNS, in turn, delivers the notification to the device and user account associated with that channel. Unlike other push notifications, raw notifications don't have a specified format. The content of the payload is entirely app-defined.

As an illustration of an app that could benefit from raw notifications, let's look at a theoretical document collaboration app. Consider two users who are editing the same document at the same time. The cloud service, which hosts the shared document, could use raw notifications to notify each user when changes are made by the other user. The raw notifications would not necessarily contain the changes to the document, but instead would signal each user's copy of the app to contact the central location and sync the available changes. By using raw notifications, the app and its cloud service can save the overhead of maintaining persistent connections the entire time the document is open.

How raw notifications work

All raw notifications are push notifications. Therefore, the setup required to send and receive push notifications applies to raw notifications as well:

- You must have a valid WNS channel to send raw notifications. For more information about acquiring a push notification channel, see How to request, create, and save a notification channel.
- You must include the **Internet** capability in your app's manifest. In the Microsoft Visual Studio manifest editor, you will find this option under the **Capabilities** tab as **Internet (Client)**. For more information, see **Capabilities**.

The body of the notification is in an app-defined format. The client receives the data as a null-terminated string (**HSTRING**) that only needs to be understood by the app.

If the client is offline, raw notifications will be cached by WNS only if the X-WNS-Cache-Policy header is included in the notification. However, only one raw notification will be cached and delivered once the device comes back online.

There are only three possible paths for a raw notification to take on the client: they will be delivered to your running app through a notification delivery event, sent to a background task, or dropped. Therefore, if the client is offline and WNS attempts to deliver a raw notification, the notification is dropped.

Creating a raw notification

Sending a raw notification is similar to sending a tile, toast, or badge push notification, with these differences:

- The HTTP Content-Type header must be set to "application/octet-stream".
- The HTTP X-WNS-Type header must be set to "wns/raw".
- The notification body can contain any string payload smaller than 5 KB in size, but must not be an empty string.

Raw notifications are intended to be used as short messages that trigger your app to take an action, such as to directly contact the service to sync a larger amount of data or to make a local state modification based on the notification content. Note that WNS push notifications cannot be guaranteed to be delivered, so your app and cloud service must account for the possibility that the raw notification might not reach the client, such as when the client is offline.

For more information on sending push notifications, see Quickstart: Sending a push notification.

Receiving a raw notification

There are two avenues through which your app can be receive raw notifications:

- Through notification delivery events while your application is running.
- Through background tasks triggered by the raw notification if your app is enabled to run background tasks.

An app can use both mechanisms to receive raw notifications. If an app implements both the notification delivery event handler and background tasks that are triggered by raw notifications, the notification delivery event will take priority when the app is running.

- If the app is running, the notification delivery event will take priority over the background task and the app will have the first opportunity to process the notification.
- The notification delivery event handler can specify, by setting the event's **PushNotificationReceivedEventArgs.Cancel** property to **true**, that the raw notification should not be passed to its background task once the handler exits. If the **Cancel** property is set to **false** or is not set (the default value is **false**), the raw notification will trigger the background task after the notification delivery event handler has done its work.

Notification delivery events

Your app can use a notification delivery event (**PushNotificationReceived**) to receive raw notifications while the app is in use. When the cloud service sends a raw notification, the running app can receive it by handling the notification delivery event on the channel URI.

If your app is not running and does not use background tasks), any raw notification sent to that app is dropped by WNS on receipt. To avoid wasting your cloud service's resources, you should consider implementing logic on the service to track whether the app is active. There are two sources of this information: an app can explicitly tell the service that it's ready to start receiving notifications, and WNS can tell the service when to stop.

- **The app notifies the cloud service**: The app can contact its service to let it know that the app is running in the foreground. The disadvantage of this approach is that the app can end up contacting your service very frequently. However, it has the advantage that the service will always know when the app is ready to receive incoming raw notifications. Another advantage is that when the app contacts its

service, the service then knows to send raw notifications to the specific instance of that app rather than broadcast.

- **The cloud service responds to WNS response messages** : Your app service can use the X-WNS-NotificationStatus and X-WNS-DeviceConnectionStatus information returned by WNS to determine when to stop sending raw notifications to the app. When your service sends a notification to a channel as an HTTP POST, it can receive one of these messages in the response:
  - **X-WNS-NotificationStatus: dropped**: This indicates that the notification was not received by the client. It's a safe assumption that the **dropped** response is caused by your app no longer being in the foreground on the user's device.
  - **X-WNS-DeviceConnectionStatus: disconnected** or **X-WNS-DeviceConnectionStatus: tempconnected**: This indicates that the Windows client no longer has a connection to WNS. Note that to receive this message from WNS, you have to ask for it by setting the X-WNS-RequestForStatus header in the notification's HTTP POST.

  Your app's cloud service can use the information in these status messages to cease communication attempts through raw notifications. The service can resume sending raw notifications once it is contacted by the app, when the app switches back into the foreground.

  Note that you should not rely on X-WNS-NotificationStatus to determine whether the notification was successfully delivered to the client.

  For more information, see Push notification service request and response headers

Background tasks triggered by raw notifications

> **ⓘ Important**
>
> Before using raw notification background tasks, an app must be granted background access via **BackgroundExecutionManager.RequestAccessAsync**.

Your background task must be registered with a **PushNotificationTrigger**. If it is not registered, the task will not run when a raw notification is received.

A background task that is triggered by a raw notification enables your app's cloud service to contact your app, even when the app is not running (though it might trigger it to run). This happens without the app having to maintain a continuous connection. Raw

notifications are the only notification type that can trigger background tasks. However, while toast, tile, and badge push notifications cannot trigger background tasks, background tasks triggered by raw notifications can update tiles and invoke toast notifications through local API calls.

As an illustration of how background tasks that are triggered by raw notifications work, let's consider an app used to read e-books. First, a user purchases a book online, possibly on another device. In response, the app's cloud service can send a raw notification to each of the user's devices, with a payload that states that the book was purchased and the app should download it. The app then directly contacts the app's cloud service to begin a background download of the new book so that later, when the user launches the app, the book is already there and ready for reading.

To use a raw notification to trigger a background task, your app must:

1. Request permission to run tasks in the background (which the user can revoke at any time) by using **BackgroundExecutionManager.RequestAccessAsync**.
2. Implement the background task. For more information, see Support your app with background tasks

Your background task is then invoked in response to the **PushNotificationTrigger**, each time a raw notification is received for your app. Your background task interprets the raw notification's app-specific payload and acts on it.

For each app, only one background task can run at a time. If a background task is triggered for an app for which a background task is already running, the first background task must complete before the new one is run.

Other resources

You can learn more by downloading the Raw notifications sample for Windows 8.1, and the Push and periodic notifications sample for Windows 8.1, and re-using their source code in your Windows 10 app.

Related topics

- Guidelines for raw notifications
- Quickstart: Creating and registering a raw notification background task
- Quickstart: Intercepting push notifications for running apps
- **RawNotification**
- **BackgroundExecutionManager.RequestAccessAsync**

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

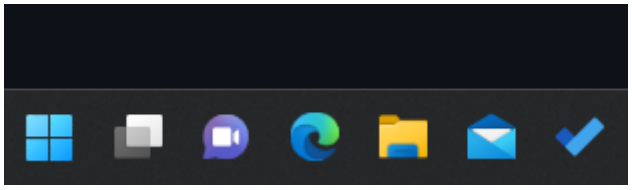
Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Pin your app to the taskbar

Article • 02/27/2024

Now you can programmatically request users to pin your Win32 or UWP app to the taskbar, similar to how you can pin your app to the Start menu. And you can check whether your app is currently pinned, and whether the taskbar allows pinning.



> **ⓘ Important**
>
> **Requires Fall Creators Update**: You must target SDK 16299 and be running build 16299 or later to use the taskbar APIs.

> **Important APIs**: TaskbarManager class

When should you ask the user to pin your app to the taskbar?

The TaskbarManager class lets you ask the user to pin your app to the taskbar; the user must approve the request. You put a lot of effort into building a stellar app, and now you have the opportunity to ask the user to pin it to taskbar. But before we dive into the code, here are some things to keep in mind as you are designing your experience:

- **Do** craft a non-disruptive and easily dismissible UX in your app with a clear call to action. Avoid using dialogs and flyouts for this purpose. Accessible pinning icons or similar UX is recommended but not required.
- **Do** ensure your app has value to the user before asking the user to pin it.
- **Don't** ask a user to pin your app if the tile is already pinned or the device doesn't support it. (This article explains how to determine whether pinning is supported.)
- **Don't** repeatedly ask the user to pin your app (they will probably get annoyed).
- **Don't** call the pin API without explicit user interaction or when your app is minimized/not open. Your app must be in the foreground for the process to work.
- **Don't** use installers to call the API.

Limited Access Feature (LAF) approval

> **ⓘ Important**
>
> The Taskbar Pinning is a Limited Access Feature (see LimitedAccessFeatures class). For more information or to request an unlock token, please use the LAF Access Token Request Form.

1. Check whether the required APIs exist

UWP

If your app supports older versions of Windows 10, you need to check whether the `TaskbarManager` class is available. You can use the ApiInformation.IsTypePresent method to perform this check. If the `TaskbarManager` class isn't available, avoid executing any calls to the APIs.

C#

```
if (ApiInformation.IsTypePresent("Windows.UI.Shell.TaskbarManager"))
{
    // Taskbar APIs exist!
}

else
{
    // Older version of Windows, no taskbar APIs
}
```

Win32

If you want to use `TaskbarManager` from your WIn32 desktop app, then you'll need to check whether desktop app support is present. You can look for the `ITaskbarManagerDesktopAppSupportStatics` marker interface on the `TaskbarManager` activation factory to perform this check. If this interface is not available, then you won't be able to use `TaskbarManager` from your desktop app.

C++

```
if
(winrt::try_get_activation_factory<winrt::Windows::UI::Shell::TaskbarManager
, winrt::Windows::UI::Shell::ITaskbarManagerDesktopAppSupportStatics>())
{
    // TaskbarManager desktop app support is available.
```

```
}
else
{
    // TaskbarManager desktop app support is not available.
}
```

2. Check whether taskbar is present and allows pinning

Windows apps can run on a wide variety of devices; not all of them support the taskbar. Right now, only Desktop devices support the taskbar. Additionally, apps may request pinning, but pinning may not be allowed at any given time. It is suggested that apps check whether pinning is allowed before UX is surfaced to prevent confusing users.

Even if the taskbar is available, a group policy on the user's machine might disable taskbar pinning. So, before you attempt to pin your app, you need to check whether pinning to the taskbar is supported. The TaskbarManager.IsPinningAllowed property returns `true` if the taskbar is present and allows pinning.

C#

```
// Check if taskbar allows pinning, apps may request pinning, but pinning
may not be allowed at any given time. It is suggested that apps check
whether pinning is allowed before a UX is surfaced in order to prevent
confusing users.

bool isPinningAllowed = TaskbarManager.GetDefault().IsPinningAllowed;
```

ⓘ Important

There are also requirements that must be met at the time the call is actually made for the pin request to be allowed:

- App is in foreground
- App has a Start menu entry

These requirements won't result in an exception if not met, the pin request will just be denied. `IsPinningAllowed` can be called to determine if a pin request (prompt) will be allowed.

ⓘ Note

If you don't want to pin your app to the taskbar and just want to find out whether the taskbar is available, use the TaskbarManager.IsSupported property.

3. Check whether your app is currently pinned to the taskbar

Obviously, there's no point in asking the user to let you pin the app to the taskbar if it's already pinned there. You can use the TaskbarManager.IsCurrentAppPinnedAsync method to check whether the app is already pinned before asking the user.

C#

```
// Check whether your app is currently pinned
bool isPinned = await TaskbarManager.GetDefault().IsCurrentAppPinnedAsync();

if (isPinned)
{
    // The app is already pinned--no point in asking to pin it again!
}
else
{
    //The app is not pinned.
}
```

4. Pin your app

If the taskbar is present and pinning is allowed and your app currently isn't pinned, you might want to show a subtle tip to let users know that they can pin your app. For example, you might show a pin icon somewhere in your UI that the user can click.

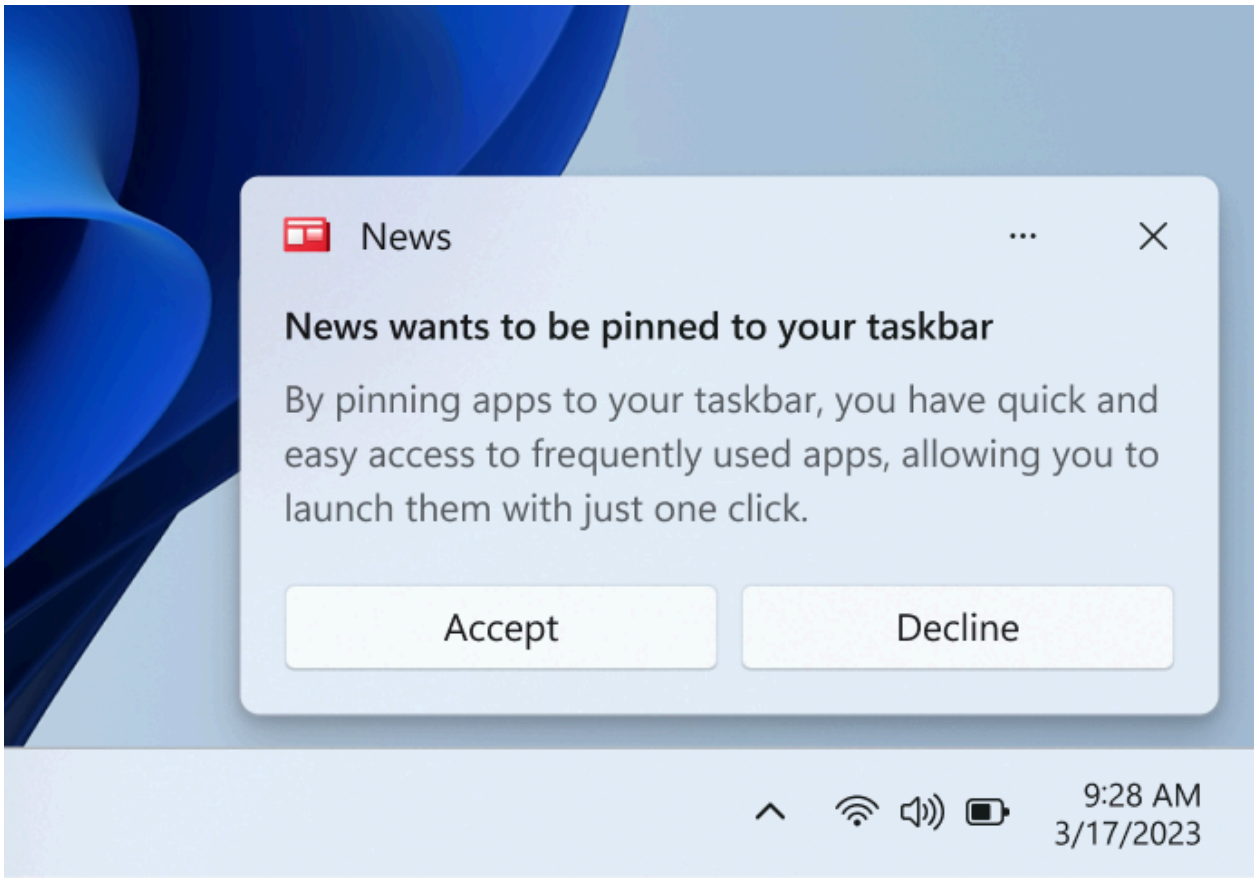
If the user clicks your pin suggestion UI, you would then call the TaskbarManager.RequestPinCurrentAppAsync method. This method displays a dialog that asks the user to confirm that they want your app pinned to the taskbar.

ⓘ Important

This must be called from a foreground UI thread, otherwise an exception will be thrown.

C#

```
// Request to be pinned to the taskbar.
bool isPinned = await
TaskbarManager.GetDefault().RequestPinCurrentAppAsync();
```



This method returns a boolean value that indicates whether your app is now pinned to the taskbar. If your app was already pinned, the method immediately returns `true` without showing the dialog to the user. If the user clicks "no" on the dialog, or pinning your app to the taskbar isn't allowed, the method returns `false`. Otherwise, the user clicked yes and the app was pinned, and the API will return `true`.

Resources

- Full UWP code sample on GitHub
- Full desktop code sample on GitHub
- TaskbarManager class
- Pin an app to the Start menu

Collaborate with us on GitHub

Windows developer feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Notifications Visualizer

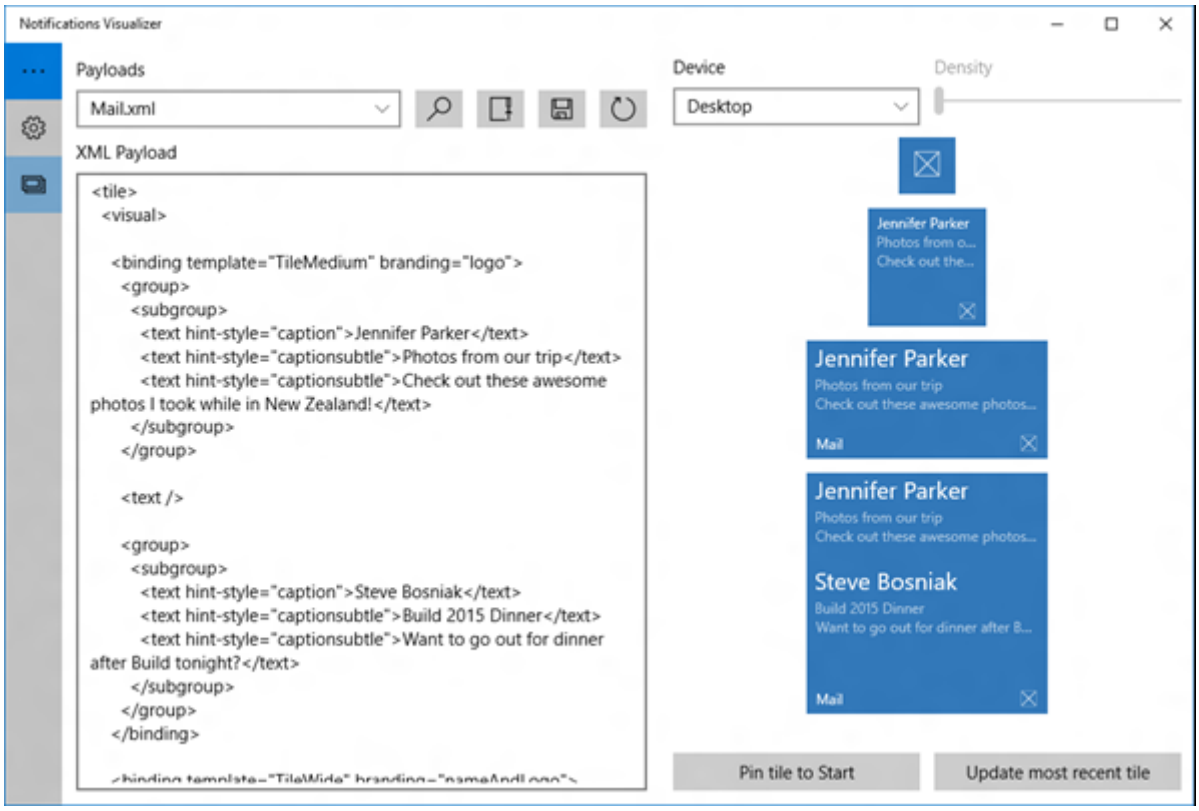
Article • 06/24/2021

Notifications Visualizer is a new Windows app in the Store that helps developers design adaptive Live Tiles and interactive toast notifications for Windows 10.

Overview

Notifications Visualizer provides instant visual previews of your tile and toast notification as you edit the XML payload, similar to Visual Studio's XAML editor/design view. The app also checks for errors, which ensures that you create a valid tile or toast notification payload.

This screenshot from the app shows the XML payload and how tile sizes appear on a selected device:



With Notifications Visualizer, you can create and test adaptive tile and toast payloads without having to edit and deploy your own app. Once you've created a payload with ideal visual results, you can integrate that into your app. See Send a local tile notification and Send a local toast to learn more.

Note Notifications Visualizer's simulation of the Windows Start menu and toast notifications isn't always completely accurate, and it doesn't support some advanced payload properties. When you have the tile or toast you want, test it by pinning the tile or popping the toast to verify that it appears as you intend.

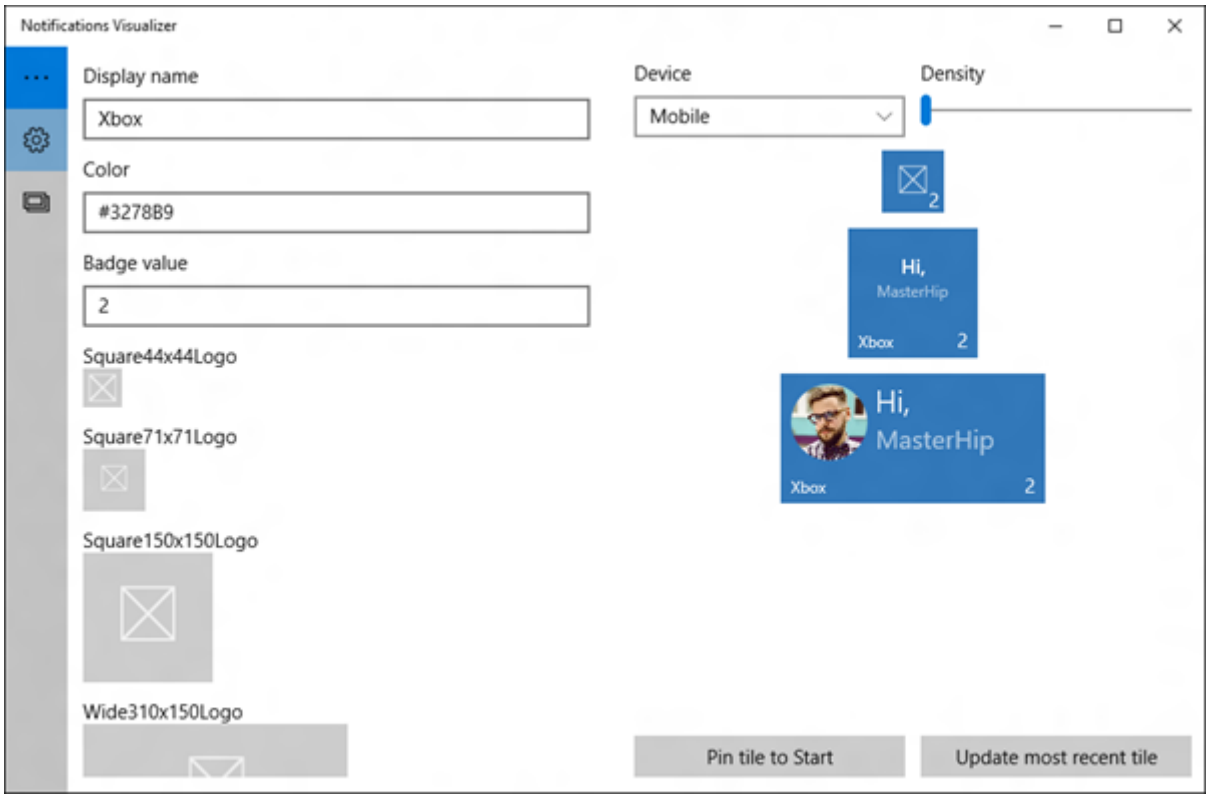
Features

Notifications Visualizer comes with a number of sample payloads to showcase what's possible with adaptive Live Tiles and interactive toasts to help you get started. You can experiment with all the different text options, groups/subgroups, background images, and you can see how the tile adapts to different devices and screens. Once you've made changes, you can save your updated payload to a file for future use.

The editor provides real-time errors and warnings. For example, if your payload is greater than 5 KB (a platform limitation), Notifications Visualizer warns you that your payload exceeds that limit. It gives you warnings for incorrect attribute names or values, which helps you debug visual issues.

You can control tile properties like display name, color, logos, ShowName, and badge value. These options help you instantly understand how your tile properties and tile notification payloads interact, and the results they produce.

This screenshot from the app shows the tile editor:



Related topics

- Get Notifications Visualizer in the Store
- Create adaptive tiles
- Interactive toasts

Choose a notification delivery method

Article • 06/24/2021

This article covers the four notification options—local, scheduled, periodic, and push—that deliver tile and badge updates and toast notification content. A tile or a toast notification can get information to your user even when the user is not directly engaged with your app. The nature and content of your app and the information that you want to deliver can help you determine which notification method or methods is best for your scenario.

Notification delivery methods overview

There are four mechanisms that an app can use to deliver a notification:

- **Local**
- **Scheduled**
- **Periodic**
- **Push**

This table summarizes the notification delivery types.

Expand table

Delivery method	Use with	Description	Examples
Local	Tile, Badge, Toast	A set of API calls that send notifications while your app is running, directly updating the tile or badge, or sending a toast notification.	• A music app updates its tile to show what's "Now Playing". • A game app updates its tile with the user's high score when the user leaves the game. • A badge whose glyph indicates that there's new info int the app is cleared when the app is activated.

Delivery method	Use with	Description	Examples
Scheduled	Tile, Toast	A set of API calls that schedule a notification in advance, to update at the time you specify.	• A calendar app sets a toast notification reminder for an upcoming meeting.
Periodic	Tile, Badge	Notifications that update tiles and badges regularly at a fixed time interval by polling a cloud service for new content.	• A weather app updates its tile, which shows the forecast, at 30-minute intervals. • A "daily deals" site updates its deal-of-the-day every morning. • A tile that displays the days until an event updates the displayed countdown each day at midnight.
Push	Tile, Badge, Toast, Raw	Notifications sent from a cloud server, even if your app isn't running.	• A shopping app sends a toast notification to let a user know about a sale on an item that they're watching. • A news app updates its tile with breaking news as it happens. • A sports app keeps its tile up-to-date during an ongoing game. • A communication

Delivery method	Use with	Description	Examples
			app provides alerts about incoming messages or phone calls.

Local notifications

Updating the app tile or badge or raising a toast notification while the app is running is the simplest of the notification delivery mechanisms; it only requires local API calls. Every app can have useful or interesting information to show on the tile, even if that content only changes after the user launches and interacts with the app. Local notifications are also a good way to keep the app tile current, even if you also use one of the other notification mechanisms. For instance, a photo app tile could show photos from a recently added album.

We recommended that your app update its tile locally on first launch, or at least immediately after the user makes a change that your app would normally reflect on the tile. That update isn't seen until the user leaves the app, but by making that change while the app is being used ensures that the tile is already up-to-date when the user departs.

While the API calls are local, the notifications can reference web images. If the web image is not available for download, is corrupted, or doesn't meet the image specifications, tiles and toast respond differently:

- Tiles: The update is not shown
- Toast: The notification is displayed, but your image is dropped

By default, local toast notifications expire in three days, and local tile notifications never expire. We recommend overriding these defaults with an explicit expiration time that makes sense for your notifications (toasts have a max of three days).

For more information, see these topics:

- Send a local tile notification
- Send a local toast notification
- Windows app notifications code samples

Scheduled notifications

Scheduled notifications are the subset of local notifications that can specify the precise time when a tile should be updated or a toast notification should be shown. Scheduled notifications are ideal in situations where the content to be updated is known in advance, such as a meeting invitation. If you don't have advance knowledge of the notification content, you should use a push or periodic notification.

Note that scheduled notifications cannot be used for badge notifications; badge notifications are best served by local, periodic, or push notifications.

By default, scheduled notifications expire three days from the time they are delivered. You can override this default expiration time on scheduled tile notifications, but you cannot override the expiration time on scheduled toasts.

For more information, see these topics:

- Scheduling a toast notification
- Windows app notifications code samples

Periodic notifications

Periodic notifications give you live tile updates with a minimal cloud service and client investment. They are also an excellent method of distributing the same content to a wide audience. Your client code specifies the URL of a cloud location that Windows polls for tile or badge updates, and how often the location should be polled. At each polling interval, Windows contacts the URL to download the specified XML content and display it on the tile.

Periodic notifications require the app to host a cloud service, and this service will be polled at the specified interval by all users who have the app installed. Note that periodic updates cannot be used for toast notifications; toast notifications are best served by scheduled or push notifications.

By default, periodic notifications expire three days from the time polling occurs. If needed, you can override this default with an explicit expiration time.

For more information, see these topics:

- Periodic notification overview
- Windows app notifications code samples

Push notifications

Push notifications are ideal to communicate real-time data or data that is personalized for your user. Push notifications are used for content that is generated at unpredictable times, such as breaking news, social network updates, or instant messages. Push notifications are also useful in situations where the data is time-sensitive in a way that would not suit periodic notifications, such as sports scores during a game.

Push notifications require a cloud service that manages push notification channels and chooses when and to whom to send notifications.

By default, push notifications expire three days from the time they are received by the device. If needed, you can override this default with an explicit expiration time (toasts have a max of three days).

For more information, see:

- Windows Push Notification Services (WNS) overview
- Guidelines for push notifications
- Windows app notifications code samples

Related topics

- Send a local tile notification
- Send a local toast notification
- Guidelines for push notifications
- Guidelines for toast notifications
- Periodic notification overview
- Windows Push Notification Services (WNS) overview
- Windows app notifications code samples on GitHub

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Choosing the right push notification channel type

Article • 08/03/2022

This article covers the three types of Windows push notification channels (primary, secondary, and alternate) that help you deliver content to your app.

(For details on how to create push notifications, see the Windows Push Notification Services (WNS) overview.)

Types of push channels

There are three types of push channels that can be used to send notifications to a Windows app. They are:

Primary channel - the "traditional" push channel. Can be used by any app in the store to send toast, tile, raw, or badge notifications. Learn more here.

Secondary tile channel - used to push tile updates to a secondary tile. Can only be used to send tile or badge notifications to a secondary tile pinned on the user's start screen

Alternate channel - **No longer supported** - A new type of channel added in the Creators Update. It allows for raw notifications to be sent to any Windows app, including those which aren't registered in the Store. Again, note that the alternate channel is no longer supported.

> ⓘ **Note**
>
> No matter which push channel you use, once your app is running on the device, it will always be able to send local toast, tile, or badge notifications. It can send local notifications from the foreground app processes or from a background task.

Primary channels

These are the most commonly used channels on Windows right now, and are good for almost any scenario where your app is going to be distributed through the Microsoft Store. They allow you to send all types of notifications to the app.

What do primary channels enable?

- **Sending tile or badge updates to the primary tile.** If the user has chosen to pin your tile to the start screen, this is your chance to show off. Send updates with useful information or reminders of experiences within your app.
- **Sending toast notifications.** Toast notifications are a chance to get some information in front of the user immediately. They are painted by the shell over top of most apps, and live in the action

center so the user can go back and interact with them later.
- **Sending raw notifications to trigger a background task.** Sometimes you want to do some work on behalf of the user based on a notification. Raw notifications allow your app's background tasks to run
- **Message encryption in transit provided by Windows using TLS.** Messages are encrypted on the wire both coming into WNS and going to the user's device.

Limitations of primary channels

- Requires using the WNS REST API to push notifications, which isn't standard across device vendors.
- Only one channel can be created per app
- Requires your app to be registered in the Microsoft Store

Creating a primary channel

```
C#

PushNotificationChannel channel =
    await
PushNotificationChannelManager.CreatePushNotificationChannelForApplicationAsync();
```

Secondary tile channels

These are channels that can be used to push tile and badge updates to a secondary tile. These are used by apps to notify users of interesting actions or information that they can interact with in the app, such as new messages in a group chat or an updated sports score.

What do secondary tile channels enable?

- Sending tile or badge notifications to secondary tiles. Secondary tiles are a great way to pull users back into your app. They are a deep link to information they care about, and putting relevant information on the tiles helps to bring them back again and again.
- Separation of channels (and expiries) between various tiles. This allows you to separate the logic in the backend between the various types of secondary tiles that a user might pin to their start screen.
- Message encryption in transit provided by Windows using TLS. Messages are encrypted on the wire both coming into WNS and going to the user's device.

Limitations of secondary tile channels

- No toast or raw notifications allowed. Toast or raw notifications sent to a secondary tile are ignored by the system.
- Requires your app to be registered in the Microsoft Store

Creating a secondary tile channel

```
C#

PushNotificationChannel channel =
    await
PushNotificationChannelManager.CreatePushNotificationChannelForSecondaryTileAsync(tileId
);
```

Alternate channels

Alternate channels enable apps to send push notifications without registering to the Microsoft Store or creating push channels outside of the primary one used for the app.

> ⓘ **Note**
>
> As of July 1, 2021 applications that want to use the web push channel or the alternate channel to send browser based notifications through WNS will need to be onboarded to the Microsoft Store.

What do alternate channels enable?

- Send raw push notifications to a Windows running on any Windows device. Alternate channels only allow for raw notifications (however you can still wake up a background task to locally show toast or tile notifications).
- Allows apps to create multiple raw push channels for different features within the app. An app can create up to 1000 alternate channels, and each one is valid for 30 days. Each of these channels can be managed or revoked separately by the app.
- Alternate push channels can be created without registering an app with the Microsoft Store. If you app is going to be installed on devices without registering it in the Microsoft Store, it will still be able to receive push notifications.
- Servers can push notifications using the W3C standard REST APIs and VAPID protocol. Alternate channels use the W3C standard protocol, this allows you to simplify the server logic that needs to be maintained.
- Full, end-to-end, message encryption. While the primary channel provides encryption while in transit, if you want to be extra secure, alternate channels enable your app to pass through encryption headers to protect a message.

Limitations of alternate channels

- Your app's server cannot send push toast, tile, or badge type notifications. You can only send push raw notifications. Your app is still able to send local notifications from your background task.
- Requires a different REST API than either primary or secondary tile channels. Using the standard W3C REST API means that your app will need to have different logic for sending push toast or

tile updates

Creating an alternate channel

```
C#

PushNotificationChannel webChannel =
    await
PushNotificationChannelManager.GetDefault().CreateRawPushNotificationChannelWithAlternat
eKeyForApplicationAsync(applicationServerKey, appChannelId);
```

Channel type comparison

Here is a quick comparison between the different types of channels:

Expand table

Type	Push toast?	Push tile/badge?	Push raw notifications?	Authentication	API	Store registration required?	Channels	Encryption
Primary	Yes	Yes - primary tile only	Yes	OAuth	WNS REST API	Yes	One per app	In Transit
Secondary Tile	No	Yes - secondary tile only	No	OAuth	WNS REST API	Yes	One per secondary tile	In Transit
Alternate	No	No	Yes	VAPID	WebPush W3C Standard	No	1,000 per app	In transit + end to end encryption possible with header pass through (requires app code)

Choosing the right channel

In general, we recommend using the primary channel in your app, with a few exceptions:

1. If you are pushing a tile update to a secondary tile, use the secondary tile push channel.
2. If you are passing out channels to other services (such as in the case of a browser) use the alternate channel.
3. If you are creating an app that won't be listed in the Windows store (such as an LOB app) use an alternate channel.

4. If you have existing web push code on your server you wish to reuse or have a need for multiple channels in your backend service, use alternate channels.

Related articles

- Send a local tile notification
- Adaptive and interactive toast notifications
- Quickstart: Sending a push notification
- How to update a badge through push notifications
- How to request, create, and save a notification channel
- How to intercept notifications for running applications
- How to authenticate with the Windows Push Notification Service (WNS)
- Push notification service request and response headers
- Guidelines and checklist for push notifications
- Raw notifications

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Periodic notification overview

Article • 06/24/2021

Periodic notifications, which are also called polled notifications, update tiles and badges at a fixed interval by downloading content from a cloud service. To use periodic notifications, your client app code needs to provide two pieces of information:

- The Uniform Resource Identifier (URI) of a web location for Windows to poll for tile or badge updates for your app
- How often that URI should be polled

Periodic notifications enable your app to get live tile updates with minimal cloud service and client investment. Periodic notifications are a good delivery method for distributing the same content to a wide audience.

Note You can learn more by downloading the Push and periodic notifications sample for Windows 8.1 and re-using its source code in your Windows 10 app.

How it works

Periodic notifications require that your app hosts a cloud service. The service will be polled periodically by all users who have the app installed. At each polling interval, such as once an hour, Windows sends an HTTP GET request to the URI, downloads the requested tile or badge content (as XML) that is supplied in response to the request, and displays the content on the app's tile.

Note that periodic updates cannot be used with toast notifications. Toast is best delivered through scheduled or push notifications.

URI location and XML content

Any valid HTTP or HTTPS web address can be used as the URI to be polled.

The cloud server's response includes the downloaded content. The content returned from the URI must conform to the Tile or Badge XML schema specification, and must be UTF-8 encoded. You can use defined HTTP headers to specify the expiration time or tag for the notification.

Polling Behavior

Call one of these methods to begin polling:

- **StartPeriodicUpdate** (Tile)
- **StartPeriodicUpdate** (Badge)
- **StartPeriodicUpdateBatch** (Tile)

When you call one of these methods, the URI is immediately polled and the tile or badge is updated with the received contents. After this initial poll, Windows continues to provide updates at the requested interval. Polling continues until you explicitly stop it (with **TileUpdater.StopPeriodicUpdate**), your app is uninstalled, or, in the case of a secondary tile, the tile is removed. Otherwise, Windows continues to poll for updates to your tile or badge even if your app is never launched again.

The recurrence interval

You specify the recurrence interval as a parameter of the methods listed above. Note that while Windows makes a best effort to poll as requested, the interval is not precise. The requested poll interval can be delayed by up to 15 minutes at the discretion of Windows.

The start time

You optionally can specify a particular time of day to begin polling. Consider an app that changes its tile content just once a day. In such a case, we recommend that you poll close to the time that you update your cloud service. For example, if a daily shopping site publishes the day's offers at 8 AM, poll for new tile content shortly after 8 AM.

If you provide a start time, the first call to the method polls for content immediately. Then, regular polling starts within 15 minutes of the provided start time.

Automatic retry behavior

The URI is polled only if the device is online. If the network is available but the URI cannot be contacted for any reason, this iteration of the polling interval is skipped, and the URI will be polled again at the next interval. If the device is in an off, sleep, or hibernated state when a polling interval is reached, the URI is polled when the device returns from its off or sleep state.

Handling app updates

If you release an app update that changes your polling URI, you should add a daily time trigger background task which calls StartPeriodicUpdate with the new URI to ensure your tiles are using the new URI. Otherwise, if users receive your app update but don't launch your app, their tiles will still be using the old URI, which may fail to display if the URI is now invalid or if the returned payload references local images that no longer exist.

Expiration of tile and badge notifications

By default, periodic tile and badge notifications expire three days from the time they are downloaded. When a notification expires, the content is removed from the badge, tile, or queue and is no longer shown to the user. It is a best practice to set an explicit expiration time on all periodic tile and badge notifications, using a time that makes sense for your app or notification, to ensure that the content does not persist longer than it is relevant. An explicit expiration time is essential for content with a defined life span. It also assures the removal of stale content if your cloud service becomes unreachable, or if the user disconnects from the network for an extended period of time.

Your cloud service sets an expiration date and time for a notification by including the X-WNS-Expires HTTP header in the response payload. The X-WNS-Expires HTTP header conforms to the HTTP-date format. For more information, see **StartPeriodicUpdate** or **StartPeriodicUpdateBatch**.

For example, during a stock market's active trading day, you can set the expiration for a stock price update to twice that of your polling interval (such as one hour after receipt if you are polling every half-hour). As another example, a news app might determine that one day is an appropriate expiration time for a daily news tile update.

Periodic notifications in the notification queue

You can use periodic tile updates with notification cycling. By default, a tile on the Start screen shows the content of a single notification until it is replaced by a new notification. When you enable cycling, up to five notifications are maintained in a queue and the tile cycles through them.

If the queue has reached its capacity of five notifications, the next new notification replaces the oldest notification in the queue. However, by setting tags on your notifications, you can affect the queue's replacement policy. A tag is an app-specific, case-insensitive string of up to 16 alphanumeric characters, specified in the X-WNS-Tag HTTP header in the response payload. Windows compares the tag of an incoming notification with the tags of all notifications already in the queue. If a match is found, the new notification replaces the queued notification with the same tag. If no match is

found, the default replacement rule is applied and the new notification replaces the oldest notification in the queue.

You can use notification queuing and tagging to implement a variety of rich notification scenarios. For example, a stock app could send five notifications, each about a different stock and each tagged with a stock name. This prevents the queue from ever containing two notifications for the same stock, the older of which is out of date.

For more information, see Using the notification queue.

Enabling the notification queue

To implement a notification queue, first enable the queue for your tile (see How to use the notification queue with local notifications). The call to enable the queue needs to be done only once in your app's lifetime, but there is no harm in calling it each time your app is launched.

Polling for more than one notification at a time

You must provide a unique URI for each notification that you'd like Windows to download for your tile. By using the **StartPeriodicUpdateBatch** method, you can provide up to five URIs at once for use with the notification queue. Each URI is polled for a single notification payload, at or near the same time. Each polled URI can return its own expiration and tag value.

Related topics

- Guidelines for periodic notifications
- How to set up periodic notifications for badges
- How to set up periodic notifications for tiles

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Input and interactions

Article • 10/04/2021

UWP apps automatically handle a wide variety of inputs and run on a variety of devices —there's nothing extra you need to do to enable touch input, for example. But there are times when you might want to optimize your app for certain types of input or devices. For example, if you're creating a painting app, you might want to customize the way you handle pen input.

The design and coding instructions in this section help you customize your UWP app for specific types of inputs.

Input primer

Familiarize yourself with each input device type and its behaviors, capabilities, and limitations when paired with certain form factors.

Gaze input

Track a user's gaze based on the location and movement of their eyes and head.

Input

Identify input devices
Pointer
Pen and Windows Ink
Touch
Mouse
Keyboard
Gamepad and remote control
Touchpad
Surface Dial
Multiple inputs
Input injection
Custom text input

Interactions

Drag and drop
Panning
Rotation
Selecting text and images
Targeting
Visual feedback

Speech and AI

Speech
Cortana

Interaction primer

Article • 10/24/2022

User interactions in the Windows app are a combination of input and output sources (such as mouse, keyboard, pen, touch, touchpad, speech, **Cortana**, controller, gesture, gaze, and so on), along with various modes or modifiers that enable extended experiences (including mouse wheel and buttons, pen eraser and barrel buttons, touch keyboard, and background app services).

UWP uses a "smart" contextual interaction system that, in most cases, eliminates the need to individually handle the unique types of input received by your app. This includes handling touch, touchpad, mouse, and pen input as a generic pointer type to support static gestures such as tap or press-and-hold, manipulation gestures such as slide for panning, or rendering digital ink.

Familiarize yourself with each input device type and its behaviors, capabilities, and limitations when paired with certain form factors. This can help you decide whether the platform controls and affordances are sufficient for your app, or require you to provide customized interaction experiences.

Gaze

For **Windows 10 April 2018 Update**, we introduced support for Gaze input using eye and head tracking input devices.

> **Note**
>
> Support for eye tracking hardware was introduced in **Windows 10 Fall Creators Update** along with **Eye control**, a built-in feature that lets you use your eyes to control the on-screen pointer, type with the on-screen keyboard, and communicate with people using text-to-speech.

Device support

- Tablet
- PCs and laptops

Typical usage

Track a user's gaze, attention, and presence based on the location and movement of their eyes. This powerful new way to use and interact with Windows apps is especially useful as an assistive technology for users with neuro-muscular diseases (such as ALS) and other disabilities involving impaired muscle or nerve functions. Gaze input also provides compelling opportunities for both gaming (including target acquisition and tracking) and traditional productivity applications, kiosks, and other interactive scenarios where traditional input devices (keyboard, mouse, touch) are not available, or where it might be useful/helpful to free up the user's hands for other tasks (such as holding shopping bags).

More info

Gaze interactions and eye tracking

Surface Dial

For **Windows 10 Anniversary Update**, we introduced the Windows Wheel category of input device. The Surface Dial is the first in this class of device.

Device support

- Tablet
- PCs and laptops

Typical usage

With a form factor based on a rotate action (or gesture), the Surface Dial is intended as a secondary, multi-modal input device that complements or modifies input from a primary device. In most cases, the device is manipulated by a user's non-dominant hand while they perform a task with their dominant hand (such as inking with a pen).

More info

Surface Dial design guidelines

Cortana

In Windows 10, **Cortana** extensibility lets you handle voice commands from a user and launch your application to carry out a single action.

Device support

- Phones and phablets
- Tablet
- PCs and laptops
- Surface Hub
- IoT
- Xbox
- HoloLens

Typical usage

A voice command is a single utterance, defined in a Voice Command Definition (VCD) file, directed at an installed app through **Cortana**. The app can be launched in the foreground or background, depending on the level and complexity of the interaction. For instance, voice commands that require additional context or user input are best handled in the foreground, while basic commands can be handled in the background.

Integrating the basic functionality of your app, and providing a central entry point for the user to accomplish most of the tasks without opening your app directly, lets **Cortana** become a liaison between your app and the user. In many cases, this can save the user significant time and effort. For more info, see Cortana design guidelines.

More info

Cortana design guidelines

Speech

Speech is an effective and natural way for people to interact with applications. It's an easy and accurate way to communicate with applications, and lets people be productive and stay informed in a variety of situations.

Speech can complement or, in many cases, be the primary input type, depending on the user's device. For example, devices such as HoloLens and Xbox do not support traditional input types (aside from a software keyboard in specific scenarios). Instead, they rely on speech input and output (often combined with other non-traditional input types such as gaze and gesture) for most user interactions.

Text-to-speech (also known as TTS, or speech synthesis) is used to inform or direct the user.

Device support

- Phones and phablets
- Tablet
- PCs and laptops
- Surface Hub
- IoT
- Xbox
- HoloLens

Typical usage

There are three modes of Speech interaction:

Natural language

Natural language is how we verbally interact with people on a regular basis. Our speech varies from person to person and situation to situation, and is generally understood. When it's not, we often use different words and word order to get the same idea across.

Natural language interactions with an app are similar: we speak to the app through our device as if it were a person and expect it to understand and react accordingly.

Natural language is the most advanced mode of speech interaction, and can be implemented and exposed through **Cortana**.

Command and control

Command and control is the use of verbal commands to activate controls and functionality such as clicking a button or selecting a menu item.

As command and control is critical to a successful user experience, a single input type is generally not recommended. Speech is typically one of several input options for a user based on their preferences or hardware capabilities.

Dictation

The most basic speech input method. Each utterance is converted to text.

Dictation is typically used when an app doesn't need to understand meaning or intent.

More info

Speech design guidelines

Pen

A pen (or stylus) can serve as a pixel precise pointing device, like a mouse, and is the optimal device for digital ink input.

Note There are two types of pen devices: active and passive.

- Passive pens do not contain electronics, and effectively emulate touch input from a finger. They require a basic device display that recognizes input based on contact pressure. Because users often rest their hand as they write on the input surface, input data can become polluted due to unsuccessful palm rejection.
- Active pens contain electronics and can work with complex device displays to provide much more extensive input data (including hover, or proximity data) to the system and your app. Palm rejection is much more robust.

When we refer to pen devices here, we are referring to active pens that provide rich input data and are used primarily for precise ink and pointing interactions.

Device support

- Phones and phablets
- Tablet
- PCs and laptops
- Surface Hub
- IoT

Typical usage

The Windows ink platform, together with a pen, provides a natural way to create handwritten notes, drawings, and annotations. The platform supports capturing ink data from digitizer input, generating ink data, rendering that data as ink strokes on the output device, managing the ink data, and performing handwriting recognition. In addition to capturing the spatial movements of the pen as the user writes or draws, your app can also collect info such as pressure, shape, color, and opacity, to offer user experiences that closely resemble drawing on paper with a pen, pencil, or brush.

Where pen and touch input diverge is the ability for touch to emulate direct manipulation of UI elements on the screen through physical gestures performed on those objects (such as swiping, sliding, dragging, rotating, and so on).

You should provide pen-specific UI commands, or affordances, to support these interactions. For example, use previous and next (or + and -) buttons to let users flip through pages of content, or rotate, resize, and zoom objects.

More info

Pen design guidelines

Touch

With touch, physical gestures from one or more fingers can be used to either emulate the direct manipulation of UI elements (such as panning, rotating, resizing, or moving), as an alternative input method (similar to mouse or pen), or as a complementary input method (to modify aspects of other input, such as smudging an ink stroke drawn with a pen). Tactile experiences such as this can provide more natural, real-world sensations for users as they interact with elements on a screen.

Device support

- Phones and phablets
- Tablet
- PCs and laptops
- Surface Hub
- IoT

Typical usage

Support for touch input can vary significantly, depending on the device.

Some devices don't support touch at all, some devices support a single touch contact, while others support multi-touch (two or more contacts).

Most devices that support multi-touch input, typically recognize ten unique, concurrent contacts.

Surface Hub devices recognize 100 unique, concurrent touch contacts.

In general, touch is:

- Single user, unless being used with a Microsoft Team device like Surface Hub, where collaboration is emphasized.
- Not constrained to device orientation.
- Used for all interactions, including text input (touch keyboard) and inking (app-configured).

More info

Touch design guidelines

Touchpad

A touchpad combines both indirect multi-touch input with the precision input of a pointing device, such as a mouse. This combination makes the touchpad suited to both a touch-optimized UI and the smaller targets of productivity apps.

Device support

- PCs and laptops
- IoT

Typical usage

Touchpads typically support a set of touch gestures that provide support similar to touch for direct manipulation of objects and UI.

Because of this convergence of interaction experiences supported by touchpads, we recommend also providing mouse-style UI commands or affordances rather than relying solely on support for touch input. Provide touchpad-specific UI commands, or affordances, to support these interactions.

You should provide mouse-specific UI commands, or affordances, to support these interactions. For example, use previous and next (or + and -) buttons to let users flip through pages of content, or rotate, resize, and zoom objects.

More info

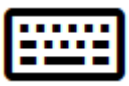
Touchpad design guidelines

Keyboard

A keyboard is the primary input device for text, and is often indispensable to people with certain disabilities or users who consider it a faster and more efficient way to interact with an app.

Device support

- Phones and phablets
- Tablet
- PCs and laptops
- Surface Hub
- IoT
- Xbox
- HoloLens



Typical usage

Users can interact with Universal Windows apps through a hardware keyboard and two software keyboards: the On-Screen Keyboard (OSK) and the touch keyboard.

The OSK is a visual, software keyboard that you can use instead of the physical keyboard to type and enter data using touch, mouse, pen/stylus or other pointing device (a touch screen is not required). The OSK is provided for systems that don't have a physical keyboard, or for users whose mobility impairments prevent them from using traditional

physical input devices. The OSK emulates most, if not all, the functionality of a hardware keyboard.

The touch keyboard is a visual, software keyboard used for text entry with touch input. The touch keyboard is not a replacement for the OSK as it is used for text input only (it doesn't emulate the hardware keyboard) and appears only when a text field or other editable text control gets focus. The touch keyboard does not support app or system commands.

Note The OSK has priority over the touch keyboard, which won't be shown if the OSK is present.

In general, a keyboard is:

- Single user.
- Not constrained to device orientation.
- Used for text input, navigation, gameplay, and accessibility.
- Always available, either proactively or reactively.

More info

Keyboard design guidelines

Mouse

A mouse is best suited for productivity apps and high-density UI where user interactions require pixel-level precision for targeting and commanding.

Device support

- Phones and phablets
- Tablet
- PCs and laptops
- Surface Hub
- IoT

Typical usage

Mouse input can be modified with the addition of various keyboard keys (Ctrl, Shift, Alt, and so on). These keys can be combined with the left mouse button, the right mouse button, the wheel button, and the X buttons for an expanded mouse-optimized command set. (Some Microsoft mouse devices have two additional buttons, referred to as X buttons, typically used to navigate back and forward in Web browsers).

Similar to pen, where mouse and touch input diverge is the ability for touch to emulate direct manipulation of UI elements on the screen through physical gestures performed on those objects (such as swiping, sliding, dragging, rotating, and so on).

You should provide mouse-specific UI commands, or affordances, to support these interactions. For example, use previous and next (or + and -) buttons to let users flip through pages of content, or rotate, resize, and zoom objects.

More info

Mouse design guidelines

Gesture

A gesture is any form of user movement that is recognized as input for controlling or interacting with an application. Gestures take many forms, from simply using a hand to target something on the screen, to specific, learned patterns of movement, to long stretches of continuous movement using the entire body. Be careful when designing custom gestures, as their meaning can vary depending on locale and culture.

Device support

- PCs and laptops
- IoT
- Xbox
- HoloLens

Typical usage

Static gesture events are fired after an interaction is complete.

- Static gesture events include Tapped, DoubleTapped, RightTapped, and Holding.

Manipulation gesture events indicate an ongoing interaction. They start firing when the user touches an element and continue until the user lifts their finger(s), or the manipulation is canceled.

- Manipulation events include multi-touch interactions such as zooming, panning, or rotating, and interactions that use inertia and velocity data such as dragging. (The information provided by the manipulation events doesn't identify the interaction, but rather provides data such as position, translation delta, and velocity.)

- Pointer events such as PointerPressed and PointerMoved provide low-level details for each touch contact, including pointer motion and the ability to distinguish press and release events.

Because of the convergence of interaction experiences supported by Windows, we recommend also providing mouse-style UI commands or affordances rather than relying solely on support for touch input. For example, use previous and next (or + and -) buttons to let users flip through pages of content, or rotate, resize, and zoom objects.

Gamepad/Controller

The gamepad/controller is a highly specialized device typically dedicated to playing games. However, it is also used for to emulate basic keyboard input and provides a UI navigation experience very similar to the keyboard.

Device support

- PCs and laptops
- IoT
- Xbox

Typical usage

Playing games and interacting with a specialized console.

Multiple inputs

Accommodating as many users and devices as possible and designing your apps to work with as many input types (gesture, speech, touch, touchpad, mouse, and keyboard) as possible maximizes flexibility, usability, and accessibility.

Device support

- Phones and phablets
- Tablet
- PCs and laptops
- Surface Hub
- IoT
- Xbox
- HoloLens

Typical usage

Just as people use a combination of voice and gesture when communicating with each other, multiple types and modes of input can also be useful when interacting with an app. However, these combined interactions need to be as intuitive and natural as possible as they can also create a very confusing experience.

Handle pointer input

Article • 06/24/2021

Receive, process, and manage input data from pointing devices (such as touch, mouse, pen/stylus, and touchpad) in your Windows applications.

> **Important**
>
> Create custom interactions only if there is a clear, well-defined requirement and the interactions supported by the platform controls don't support your scenario.
> If you customize the interaction experiences in your Windows application, users expect them to be consistent, intuitive, and discoverable. For these reasons, we recommend that you model your custom interactions on those supported by the **platform controls**. The platform controls provide the full Windows app user interaction experience, including standard interactions, animated physics effects, visual feedback, and accessibility.

Important APIs

- Windows.Devices.Input
- Windows.UI.Input
- Windows.UI.Xaml.Input

Pointers

Most interaction experiences typically involve the user identifying the object they want to interact with by pointing at it through input devices such as touch, mouse, pen/stylus, and touchpad. Because the raw Human Interface Device (HID) data provided by these input devices includes many common properties, the data is promoted and consolidated into a unified input stack and exposed as device-agnostic pointer data. Your Windows applications can then consume this data without worrying about the input device being used.

> **Note**
>
> Device-specific info is also promoted from the raw HID data should your app require it.

Each input point (or contact) on the input stack is represented by a **Pointer** object exposed through the **PointerRoutedEventArgs** parameter in the various pointer event handlers. In the case of multi-pen or multi-touch input, each contact is treated as a unique input pointer.

Pointer events

Pointer events expose basic info such as input device type and detection state (in range or in contact), and extended info such as location, pressure, and contact geometry. In addition, specific device properties such as which mouse button a user pressed or whether the pen eraser tip is being used are also available. If your app needs to differentiate between input devices and their capabilities, see Identify input devices.

Windows apps can listen for the following pointer events:

> **Note**
>
> Constrain pointer input to a specific UI element by calling **CapturePointer** on that element within a pointer event handler. When a pointer is captured by an element, only that object receives pointer input events, even when the pointer moves outside the bounding area of the object. The **IsInContact** (mouse button pressed, touch or stylus in contact) must be true for **CapturePointer** to be successful.

Event	Description
PointerCanceled	Occurs when a pointer is canceled by the platform. This can occur in the following circumstances: • Touch pointers are canceled when a pen is detected within range of the input surface. • An active contact is not detected for more than 100 ms. • Monitor/display is changed (resolution, settings, multi-mon configuration). • The desktop is locked or the user has logged off. • The number of simultaneous contacts exceeded the number supported by the device.

Event	Description
PointerCaptureLost	Occurs when another UI element captures the pointer, the pointer was released, or another pointer was programmatically captured. **Note** There is no corresponding pointer capture event.
PointerEntered	Occurs when a pointer enters the bounding area of an element. This can happen in slightly different ways for touch, touchpad, mouse, and pen input. • Touch requires a finger contact to fire this event, either from a direct touch down on the element or from moving into the bounding area of the element. • Mouse and touchpad both have an on-screen cursor that is always visible and fires this event even if no mouse or touchpad button is pressed. • Like touch, pen fires this event with a direct pen down on the element or from moving into the bounding area of the element. However, pen also has a hover state (IsInRange) that, when true, fires this event.

Event	Description
PointerExited	Occurs when a pointer leaves the bounding area of an element. This can happen in slightly different ways for touch, touchpad, mouse, and pen input. • Touch requires a finger contact and fires this event when the pointer moves out of the bounding area of the element. • Mouse and touchpad both have an on-screen cursor that is always visible and fires this event even if no mouse or touchpad button is pressed. • Like touch, pen fires this event when moving out of the bounding area of the element. However, pen also has a hover state (IsInRange) that fires this event when the state changes from true to false.
PointerMoved	Occurs when a pointer changes coordinates, button state, pressure, tilt, or contact geometry (for example, width and height) within the bounding area of an element. This can happen in slightly different ways for touch, touchpad, mouse, and pen input. • Touch requires a finger contact and fires this event only when in contact within the bounding area of the element. • Mouse and touchpad both have an on-screen cursor that is always visible and fires this event even if no mouse or touchpad button is pressed. • Like touch, pen fires this event when in contact within the bounding area of the element. However, pen also has a hover state (IsInRange) that, when true and within the bounding area of the element, fires this event.

Event	Description
PointerPressed	Occurs when the pointer indicates a press action (such as a touch down, mouse button down, pen down, or touchpad button down) within the bounding area of an element. CapturePointer must be called from the handler for this event.
PointerReleased	Occurs when the pointer indicates a release action (such as a touch up, mouse button up, pen up, or touchpad button up) within the bounding area of an element or, if the pointer is captured, outside the bounding area.
PointerWheelChanged	Occurs when the mouse wheel is rotated. Mouse input is associated with a single pointer assigned when mouse input is first detected. Clicking a mouse button (left, wheel, or right) creates a secondary association between the pointer and that button through the PointerMoved event.

Pointer event example

Here are some code snippets from a basic pointer tracking app that show how to listen for and handle events for multiple pointers, and get various properties for the associated pointers.

Download this sample from Pointer input sample (basic)

Create the UI

For this example, we use a Rectangle (`Target`) as the object consuming pointer input. The color of the target changes when the pointer status changes.

Details for each pointer are displayed in a floating TextBlock that follows the pointer as it moves. The pointer events themselves are reported in the RichTextBlock to the right of the rectangle.

This is the Extensible Application Markup Language (XAML) for the UI in this example.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="*" />
        <ColumnDefinition Width="250"/>
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
        <RowDefinition Height="*" />
        <RowDefinition Height="320" />
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Canvas Name="Container"
            Grid.Column="0"
            Grid.Row="1"
            HorizontalAlignment="Center"
            VerticalAlignment="Center"
            Margin="245,0"
```

```
            Height="320"  Width="640">
        <Rectangle Name="Target"
                   Fill="#FF0000"
                   Stroke="Black"
                   StrokeThickness="0"
                   Height="320" Width="640" />
    </Canvas>
    <Grid Grid.Column="1" Grid.Row="0" Grid.RowSpan="3">
        <Grid.RowDefinitions>
            <RowDefinition Height="50" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <Button Name="buttonClear"
                Grid.Row="0"
                Content="Clear"
                Foreground="White"
                HorizontalAlignment="Stretch"
                VerticalAlignment="Stretch">
        </Button>
        <ScrollViewer Name="eventLogScrollViewer" Grid.Row="1"
                      VerticalScrollMode="Auto"
                      Background="Black">
            <RichTextBlock Name="eventLog"
                      TextWrapping="Wrap"
                      Foreground="#FFFFFF"
                      ScrollViewer.VerticalScrollBarVisibility="Visible"

ScrollViewer.HorizontalScrollBarVisibility="Disabled"
                      Grid.ColumnSpan="2">
            </RichTextBlock>
        </ScrollViewer>
    </Grid>
</Grid>
```

Listen for pointer events

In most cases, we recommend that you get pointer info through the **PointerRoutedEventArgs** of the event handler.

If the event argument doesn't expose the pointer details required, you can get access to extended **PointerPoint** info exposed through the **GetCurrentPoint** and **GetIntermediatePoints** methods of **PointerRoutedEventArgs**.

The following code sets up the global dictionary object for tracking each active pointer, and identifies the various pointer event listeners for the target object.

CSharp

```
// Dictionary to maintain information about each active pointer.
// An entry is added during PointerPressed/PointerEntered events and removed
```

```
// during PointerReleased/PointerCaptureLost/PointerCanceled/PointerExited
events.
Dictionary<uint, Windows.UI.Xaml.Input.Pointer> pointers;

public MainPage()
{
    this.InitializeComponent();

    // Initialize the dictionary.
    pointers = new Dictionary<uint, Windows.UI.Xaml.Input.Pointer>();

    // Declare the pointer event handlers.
    Target.PointerPressed +=
        new PointerEventHandler(Target_PointerPressed);
    Target.PointerEntered +=
        new PointerEventHandler(Target_PointerEntered);
    Target.PointerReleased +=
        new PointerEventHandler(Target_PointerReleased);
    Target.PointerExited +=
        new PointerEventHandler(Target_PointerExited);
    Target.PointerCanceled +=
        new PointerEventHandler(Target_PointerCanceled);
    Target.PointerCaptureLost +=
        new PointerEventHandler(Target_PointerCaptureLost);
    Target.PointerMoved +=
        new PointerEventHandler(Target_PointerMoved);
    Target.PointerWheelChanged +=
        new PointerEventHandler(Target_PointerWheelChanged);

    buttonClear.Click +=
        new RoutedEventHandler(ButtonClear_Click);
}
```

Handle pointer events

Next, we use UI feedback to demonstrate basic pointer event handlers.

- This handler manages the **PointerPressed** event. We add the event to the event log, add the pointer to the active pointer dictionary, and display the pointer details.

  > **Note**
  >
  > **PointerPressed** and **PointerReleased** events do not always occur in pairs. Your app should listen for and handle any event that might conclude a pointer down (such as **PointerExited**, **PointerCanceled**, and **PointerCaptureLost**).

```
C#
```

```
/// <summary>
/// The pointer pressed event handler.
/// PointerPressed and PointerReleased don't always occur in pairs.
/// Your app should listen for and handle any event that can conclude
/// a pointer down (PointerExited, PointerCanceled, PointerCaptureLost).
/// </summary>
/// <param name="sender">Source of the pointer event.</param>
/// <param name="e">Event args for the pointer routed event.</param>
void Target_PointerPressed(object sender, PointerRoutedEventArgs e)
{
    // Prevent most handlers along the event route from handling the same
event again.
    e.Handled = true;

    PointerPoint ptrPt = e.GetCurrentPoint(Target);

    // Update event log.
    UpdateEventLog("Down: " + ptrPt.PointerId);

    // Lock the pointer to the target.
    Target.CapturePointer(e.Pointer);

    // Update event log.
    UpdateEventLog("Pointer captured: " + ptrPt.PointerId);

    // Check if pointer exists in dictionary (ie, enter occurred prior to
press).
    if (!pointers.ContainsKey(ptrPt.PointerId))
    {
        // Add contact to dictionary.
        pointers[ptrPt.PointerId] = e.Pointer;
    }

    // Change background color of target when pointer contact detected.
    Target.Fill = new SolidColorBrush(Windows.UI.Colors.Green);

    // Display pointer details.
    CreateInfoPop(ptrPt);
}
```

- This handler manages the **PointerEntered** event. We add the event to the event log, add the pointer to the pointer collection, and display the pointer details.

C#

```
/// <summary>
/// The pointer entered event handler.
/// We do not capture the pointer on this event.
/// </summary>
/// <param name="sender">Source of the pointer event.</param>
/// <param name="e">Event args for the pointer routed event.</param>
private void Target_PointerEntered(object sender, PointerRoutedEventArgs e)
```

```
{
    // Prevent most handlers along the event route from handling the same
event again.
    e.Handled = true;

    PointerPoint ptrPt = e.GetCurrentPoint(Target);

    // Update event log.
    UpdateEventLog("Entered: " + ptrPt.PointerId);

    // Check if pointer already exists (if enter occurred prior to down).
    if (!pointers.ContainsKey(ptrPt.PointerId))
    {
        // Add contact to dictionary.
        pointers[ptrPt.PointerId] = e.Pointer;
    }

    if (pointers.Count == 0)
    {
        // Change background color of target when pointer contact detected.
        Target.Fill = new SolidColorBrush(Windows.UI.Colors.Blue);
    }

    // Display pointer details.
    CreateInfoPop(ptrPt);
}
```

- This handler manages the **PointerMoved** event. We add the event to the event log and update the pointer details.

> **ⓘ Important**
>
> Mouse input is associated with a single pointer assigned when mouse input is first detected. Clicking a mouse button (left, wheel, or right) creates a secondary association between the pointer and that button through the **PointerPressed** event. The **PointerReleased** event is fired only when that same mouse button is released (no other button can be associated with the pointer until this event is complete). Because of this exclusive association, other mouse button clicks are routed through the **PointerMoved** event.

C#

```
/// <summary>
/// The pointer moved event handler.
/// </summary>
/// <param name="sender">Source of the pointer event.</param>
/// <param name="e">Event args for the pointer routed event.</param>
private void Target_PointerMoved(object sender, PointerRoutedEventArgs e)
{
```

```
    // Prevent most handlers along the event route from handling the same
event again.
    e.Handled = true;

    PointerPoint ptrPt = e.GetCurrentPoint(Target);

    // Multiple, simultaneous mouse button inputs are processed here.
    // Mouse input is associated with a single pointer assigned when
    // mouse input is first detected.
    // Clicking additional mouse buttons (left, wheel, or right) during
    // the interaction creates secondary associations between those buttons
    // and the pointer through the pointer pressed event.
    // The pointer released event is fired only when the last mouse button
    // associated with the interaction (not necessarily the initial button)
    // is released.
    // Because of this exclusive association, other mouse button clicks are
    // routed through the pointer move event.
    if (ptrPt.PointerDevice.PointerDeviceType ==
Windows.Devices.Input.PointerDeviceType.Mouse)
    {
        if (ptrPt.Properties.IsLeftButtonPressed)
        {
            UpdateEventLog("Left button: " + ptrPt.PointerId);
        }
        if (ptrPt.Properties.IsMiddleButtonPressed)
        {
            UpdateEventLog("Wheel button: " + ptrPt.PointerId);
        }
        if (ptrPt.Properties.IsRightButtonPressed)
        {
            UpdateEventLog("Right button: " + ptrPt.PointerId);
        }
    }

    // Display pointer details.
    UpdateInfoPop(ptrPt);
}
```

- This handler manages the **PointerWheelChanged** event. We add the event to the event log, add the pointer to the pointer array (if necessary), and display the pointer details.

C#

```
/// <summary>
/// The pointer wheel event handler.
/// </summary>
/// <param name="sender">Source of the pointer event.</param>
/// <param name="e">Event args for the pointer routed event.</param>
private void Target_PointerWheelChanged(object sender,
PointerRoutedEventArgs e)
{
```

```
    // Prevent most handlers along the event route from handling the same
event again.
    e.Handled = true;

    PointerPoint ptrPt = e.GetCurrentPoint(Target);

    // Update event log.
    UpdateEventLog("Mouse wheel: " + ptrPt.PointerId);

    // Check if pointer already exists (for example, enter occurred prior to
wheel).
    if (!pointers.ContainsKey(ptrPt.PointerId))
    {
        // Add contact to dictionary.
        pointers[ptrPt.PointerId] = e.Pointer;
    }

    // Display pointer details.
    CreateInfoPop(ptrPt);
}
```

- This handler manages the **PointerReleased** event where contact with the digitizer is terminated. We add the event to the event log, remove the pointer from the pointer collection, and update the pointer details.

C#

```
/// <summary>
/// The pointer released event handler.
/// PointerPressed and PointerReleased don't always occur in pairs.
/// Your app should listen for and handle any event that can conclude
/// a pointer down (PointerExited, PointerCanceled, PointerCaptureLost).
/// </summary>
/// <param name="sender">Source of the pointer event.</param>
/// <param name="e">Event args for the pointer routed event.</param>
void Target_PointerReleased(object sender, PointerRoutedEventArgs e)
{
    // Prevent most handlers along the event route from handling the same
event again.
    e.Handled = true;

    PointerPoint ptrPt = e.GetCurrentPoint(Target);

    // Update event log.
    UpdateEventLog("Up: " + ptrPt.PointerId);

    // If event source is mouse or touchpad and the pointer is still
    // over the target, retain pointer and pointer details.
    // Return without removing pointer from pointers dictionary.
    // For this example, we assume a maximum of one mouse pointer.
    if (ptrPt.PointerDevice.PointerDeviceType !=
Windows.Devices.Input.PointerDeviceType.Mouse)
```

```
    {
        // Update target UI.
        Target.Fill = new SolidColorBrush(Windows.UI.Colors.Red);

        DestroyInfoPop(ptrPt);

        // Remove contact from dictionary.
        if (pointers.ContainsKey(ptrPt.PointerId))
        {
            pointers[ptrPt.PointerId] = null;
            pointers.Remove(ptrPt.PointerId);
        }

        // Release the pointer from the target.
        Target.ReleasePointerCapture(e.Pointer);

        // Update event log.
        UpdateEventLog("Pointer released: " + ptrPt.PointerId);
    }
    else
    {
        Target.Fill = new SolidColorBrush(Windows.UI.Colors.Blue);
    }
}
```

- This handler manages the **PointerExited** event (when contact with the digitizer is maintained). We add the event to the event log, remove the pointer from the pointer array, and update the pointer details.

C#

```
/// <summary>
/// The pointer exited event handler.
/// </summary>
/// <param name="sender">Source of the pointer event.</param>
/// <param name="e">Event args for the pointer routed event.</param>
private void Target_PointerExited(object sender, PointerRoutedEventArgs e)
{
    // Prevent most handlers along the event route from handling the same
event again.
    e.Handled = true;

    PointerPoint ptrPt = e.GetCurrentPoint(Target);

    // Update event log.
    UpdateEventLog("Pointer exited: " + ptrPt.PointerId);

    // Remove contact from dictionary.
    if (pointers.ContainsKey(ptrPt.PointerId))
    {
        pointers[ptrPt.PointerId] = null;
        pointers.Remove(ptrPt.PointerId);
```

```
    }

    if (pointers.Count == 0)
    {
        Target.Fill = new SolidColorBrush(Windows.UI.Colors.Red);
    }

    // Update the UI and pointer details.
    DestroyInfoPop(ptrPt);
}
```

- This handler manages the **PointerCanceled** event. We add the event to the event log, remove the pointer from the pointer array, and update the pointer details.

C#

```
/// <summary>
/// The pointer canceled event handler.
/// Fires for various reasons, including:
///    - Touch contact canceled by pen coming into range of the surface.
///    - The device doesn't report an active contact for more than 100ms.
///    - The desktop is locked or the user logged off.
///    - The number of simultaneous contacts exceeded the number supported
by the device.
/// </summary>
/// <param name="sender">Source of the pointer event.</param>
/// <param name="e">Event args for the pointer routed event.</param>
private void Target_PointerCanceled(object sender, PointerRoutedEventArgs e)
{
    // Prevent most handlers along the event route from handling the same
event again.
    e.Handled = true;

    PointerPoint ptrPt = e.GetCurrentPoint(Target);

    // Update event log.
    UpdateEventLog("Pointer canceled: " + ptrPt.PointerId);

    // Remove contact from dictionary.
    if (pointers.ContainsKey(ptrPt.PointerId))
    {
        pointers[ptrPt.PointerId] = null;
        pointers.Remove(ptrPt.PointerId);
    }

    if (pointers.Count == 0)
    {
        Target.Fill = new SolidColorBrush(Windows.UI.Colors.Black);
    }

    DestroyInfoPop(ptrPt);
}
```

- This handler manages the **PointerCaptureLost** event. We add the event to the event log, remove the pointer from the pointer array, and update the pointer details.

> **Note**
>
> **PointerCaptureLost** can occur instead of **PointerReleased**. Pointer capture can be lost for various reasons including user interaction, programmatic capture of another pointer, calling **PointerReleased**.

C#

```
/// <summary>
/// The pointer capture lost event handler.
/// Fires for various reasons, including:
///    - User interactions
///    - Programmatic capture of another pointer
///    - Captured pointer was deliberately released
// PointerCaptureLost can fire instead of PointerReleased.
/// </summary>
/// <param name="sender">Source of the pointer event.</param>
/// <param name="e">Event args for the pointer routed event.</param>
private void Target_PointerCaptureLost(object sender, PointerRoutedEventArgs
e)
{
    // Prevent most handlers along the event route from handling the same
event again.
    e.Handled = true;

    PointerPoint ptrPt = e.GetCurrentPoint(Target);

    // Update event log.
    UpdateEventLog("Pointer capture lost: " + ptrPt.PointerId);

    if (pointers.Count == 0)
    {
        Target.Fill = new SolidColorBrush(Windows.UI.Colors.Black);
    }

    // Remove contact from dictionary.
    if (pointers.ContainsKey(ptrPt.PointerId))
    {
        pointers[ptrPt.PointerId] = null;
        pointers.Remove(ptrPt.PointerId);
    }

    DestroyInfoPop(ptrPt);
}
```

Get pointer properties

As stated earlier, you must get most extended pointer info from a Windows.UI.Input.PointerPoint object obtained through the GetCurrentPoint and GetIntermediatePoints methods of PointerRoutedEventArgs. The following code snippets show how.

- First, we create a new TextBlock for each pointer.

C#

```
/// <summary>
/// Create the pointer info popup.
/// </summary>
/// <param name="ptrPt">Reference to the input pointer.</param>
void CreateInfoPop(PointerPoint ptrPt)
{
    TextBlock pointerDetails = new TextBlock();
    pointerDetails.Name = ptrPt.PointerId.ToString();
    pointerDetails.Foreground = new
SolidColorBrush(Windows.UI.Colors.White);
    pointerDetails.Text = QueryPointer(ptrPt);

    TranslateTransform x = new TranslateTransform();
    x.X = ptrPt.Position.X + 20;
    x.Y = ptrPt.Position.Y + 20;
    pointerDetails.RenderTransform = x;

    Container.Children.Add(pointerDetails);
}
```

- Then we provide a way to update the pointer info in an existing TextBlock associated with that pointer.

C#

```
/// <summary>
/// Update the pointer info popup.
/// </summary>
/// <param name="ptrPt">Reference to the input pointer.</param>
void UpdateInfoPop(PointerPoint ptrPt)
{
    foreach (var pointerDetails in Container.Children)
    {
        if (pointerDetails.GetType().ToString() ==
"Windows.UI.Xaml.Controls.TextBlock")
        {
            TextBlock textBlock = (TextBlock)pointerDetails;
            if (textBlock.Name == ptrPt.PointerId.ToString())
            {
```

```
                // To get pointer location details, we need extended pointer
info.
                // We get the pointer info through the getCurrentPoint
method
                // of the event argument.
                TranslateTransform x = new TranslateTransform();
                x.X = ptrPt.Position.X + 20;
                x.Y = ptrPt.Position.Y + 20;
                pointerDetails.RenderTransform = x;
                textBlock.Text = QueryPointer(ptrPt);
            }
        }
    }
}
```

- Finally, we query various pointer properties.

C#

```
/// <summary>
/// Get pointer details.
/// </summary>
/// <param name="ptrPt">Reference to the input pointer.</param>
/// <returns>A string composed of pointer details.</returns>
String QueryPointer(PointerPoint ptrPt)
{
    String details = "";

    switch (ptrPt.PointerDevice.PointerDeviceType)
    {
        case Windows.Devices.Input.PointerDeviceType.Mouse:
            details += "\nPointer type: mouse";
            break;
        case Windows.Devices.Input.PointerDeviceType.Pen:
            details += "\nPointer type: pen";
            if (ptrPt.IsInContact)
            {
                details += "\nPressure: " + ptrPt.Properties.Pressure;
                details += "\nrotation: " + ptrPt.Properties.Orientation;
                details += "\nTilt X: " + ptrPt.Properties.XTilt;
                details += "\nTilt Y: " + ptrPt.Properties.YTilt;
                details += "\nBarrel button pressed: " +
ptrPt.Properties.IsBarrelButtonPressed;
            }
            break;
        case Windows.Devices.Input.PointerDeviceType.Touch:
            details += "\nPointer type: touch";
            details += "\nrotation: " + ptrPt.Properties.Orientation;
            details += "\nTilt X: " + ptrPt.Properties.XTilt;
            details += "\nTilt Y: " + ptrPt.Properties.YTilt;
            break;
        default:
            details += "\nPointer type: n/a";
```

```
            break;
    }

    GeneralTransform gt = Target.TransformToVisual(this);
    Point screenPoint;

    screenPoint = gt.TransformPoint(new Point(ptrPt.Position.X,
ptrPt.Position.Y));
    details += "\nPointer Id: " + ptrPt.PointerId.ToString() +
        "\nPointer location (target): " + Math.Round(ptrPt.Position.X) + ",
" + Math.Round(ptrPt.Position.Y) +
        "\nPointer location (container): " + Math.Round(screenPoint.X) + ",
" + Math.Round(screenPoint.Y);

    return details;
}
```

Primary pointer

Some input devices, such as a touch digitizer or touchpad, support more than the typical single pointer of a mouse or a pen (in most cases as the Surface Hub supports two pen inputs).

Use the read-only **IsPrimary** property of the **PointerPointerProperties** class to identify and differentiate a single primary pointer (the primary pointer is always the first pointer detected during an input sequence).

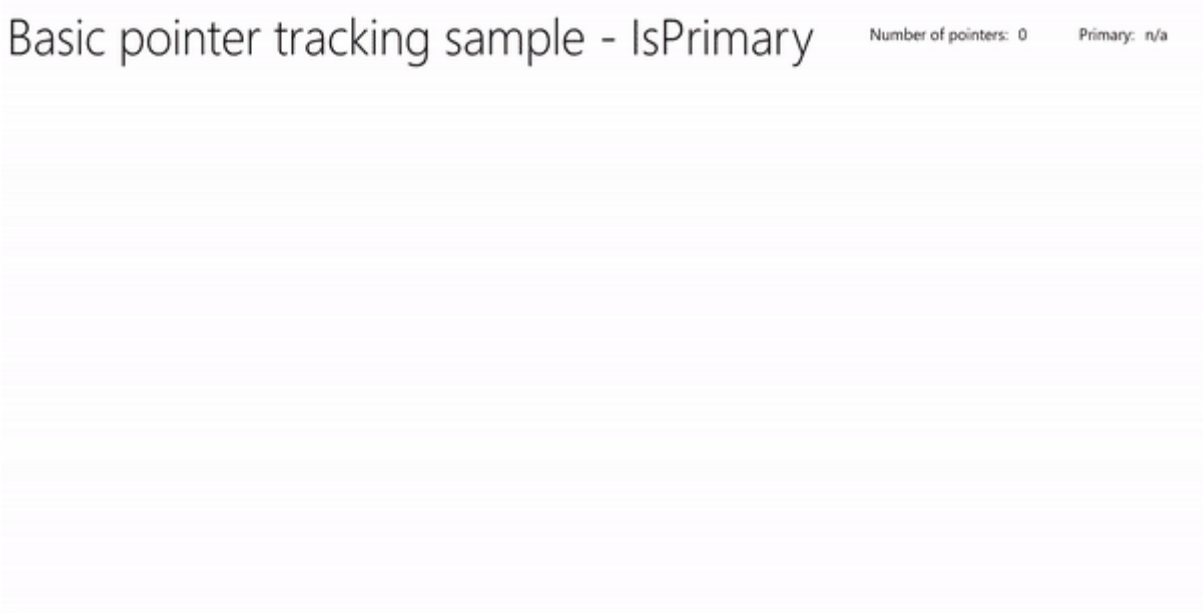
By identifying the primary pointer, you can use it to emulate mouse or pen input, customize interactions, or provide some other specific functionality or UI.

> **Note**
>
> If the primary pointer is released, canceled, or lost during an input sequence, a primary input pointer is not created until a new input sequence is initiated (an input sequence ends when all pointers have been released, canceled, or lost).

Primary pointer animation example

These code snippets show how you can provide special visual feedback to help a user differentiate between pointer inputs in your application.

This particular app uses both color and animation to highlight the primary pointer.



Download this sample from Pointer input sample (UserControl with animation)

Visual feedback

We define a **UserControl**, based on a XAML **Ellipse** object, that highlights where each pointer is on the canvas and uses a **Storyboard** to animate the ellipse that corresponds to the primary pointer.

Here's the XAML:

XAML

```
<UserControl
    x:Class="UWP_Pointers.PointerEllipse"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:UWP_Pointers"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="100"
    d:DesignWidth="100">

    <UserControl.Resources>
        <Style x:Key="EllipseStyle" TargetType="Ellipse">
            <Setter Property="Transitions">
                <Setter.Value>
                    <TransitionCollection>
                        <ContentThemeTransition/>
                    </TransitionCollection>
                </Setter.Value>
            </Setter>
        </Style>
```

```
        <Storyboard x:Name="myStoryboard">
            <!-- Animates the value of a Double property between
            two target values using linear interpolation over the
            specified Duration. -->
            <DoubleAnimation
              Storyboard.TargetName="ellipse"
              Storyboard.TargetProperty="(RenderTransform).
(ScaleTransform.ScaleY)"
              Duration="0:0:1"
              AutoReverse="True"
              RepeatBehavior="Forever" From="1.0" To="1.4">
            </DoubleAnimation>

            <!-- Animates the value of a Double property between
            two target values using linear interpolation over the
            specified Duration. -->
            <DoubleAnimation
              Storyboard.TargetName="ellipse"
              Storyboard.TargetProperty="(RenderTransform).
(ScaleTransform.ScaleX)"
              Duration="0:0:1"
              AutoReverse="True"
              RepeatBehavior="Forever" From="1.0" To="1.4">
            </DoubleAnimation>

            <!-- Animates the value of a Color property between
            two target values using linear interpolation over the
            specified Duration. -->
            <ColorAnimation
                Storyboard.TargetName="ellipse"
                EnableDependentAnimation="True"
                Storyboard.TargetProperty="(Fill).(SolidColorBrush.Color)"
                From="White" To="Red"  Duration="0:0:1"
                AutoReverse="True" RepeatBehavior="Forever"/>
        </Storyboard>
    </UserControl.Resources>

    <Grid x:Name="CompositionContainer">
        <Ellipse Name="ellipse"
        StrokeThickness="2"
        Width="{x:Bind Diameter}"
        Height="{x:Bind Diameter}"
        Style="{StaticResource EllipseStyle}" />
    </Grid>
</UserControl>
```

And here's the code-behind:

C#

```
using Windows.Foundation;
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
```

```
using Windows.UI.Xaml.Media;

// The User Control item template is documented at
// https://go.microsoft.com/fwlink/?LinkId=234236

namespace UWP_Pointers
{
    /// <summary>
    /// Pointer feedback object.
    /// </summary>
    public sealed partial class PointerEllipse : UserControl
    {
        // Reference to the application canvas.
        Canvas canvas;

        /// <summary>
        /// Ellipse UI for pointer feedback.
        /// </summary>
        /// <param name="c">The drawing canvas.</param>
        public PointerEllipse(Canvas c)
        {
            this.InitializeComponent();
            canvas = c;
        }

        /// <summary>
        /// Gets or sets the pointer Id to associate with the PointerEllipse
object.
        /// </summary>
        public uint PointerId
        {
            get { return (uint)GetValue(PointerIdProperty); }
            set { SetValue(PointerIdProperty, value); }
        }
        // Using a DependencyProperty as the backing store for PointerId.
        // This enables animation, styling, binding, etc...
        public static readonly DependencyProperty PointerIdProperty =
            DependencyProperty.Register("PointerId", typeof(uint),
                typeof(PointerEllipse), new PropertyMetadata(null));


        /// <summary>
        /// Gets or sets whether the associated pointer is Primary.
        /// </summary>
        public bool PrimaryPointer
        {
            get { return (bool)GetValue(PrimaryPointerProperty); }
            set
            {
                SetValue(PrimaryPointerProperty, value);
            }
        }
        // Using a DependencyProperty as the backing store for
PrimaryPointer.
        // This enables animation, styling, binding, etc...
```

```
        public static readonly DependencyProperty PrimaryPointerProperty =
            DependencyProperty.Register("PrimaryPointer", typeof(bool),
                typeof(PointerEllipse), new PropertyMetadata(false));


        /// <summary>
        /// Gets or sets the ellipse style based on whether the pointer is
Primary.
        /// </summary>
        public bool PrimaryEllipse
        {
            get { return (bool)GetValue(PrimaryEllipseProperty); }
            set
            {
                SetValue(PrimaryEllipseProperty, value);
                if (value)
                {
                    SolidColorBrush fillBrush =

(SolidColorBrush)Application.Current.Resources["PrimaryFillBrush"];
                    SolidColorBrush strokeBrush =

(SolidColorBrush)Application.Current.Resources["PrimaryStrokeBrush"];

                    ellipse.Fill = fillBrush;
                    ellipse.Stroke = strokeBrush;
                    ellipse.RenderTransform = new CompositeTransform();
                    ellipse.RenderTransformOrigin = new Point(.5, .5);
                    myStoryboard.Begin();
                }
                else
                {
                    SolidColorBrush fillBrush =

(SolidColorBrush)Application.Current.Resources["SecondaryFillBrush"];
                    SolidColorBrush strokeBrush =

(SolidColorBrush)Application.Current.Resources["SecondaryStrokeBrush"];
                    ellipse.Fill = fillBrush;
                    ellipse.Stroke = strokeBrush;
                }
            }
        }
        // Using a DependencyProperty as the backing store for
PrimaryEllipse.
        // This enables animation, styling, binding, etc...
        public static readonly DependencyProperty PrimaryEllipseProperty =
            DependencyProperty.Register("PrimaryEllipse",
                typeof(bool), typeof(PointerEllipse), new
PropertyMetadata(false));


        /// <summary>
        /// Gets or sets the diameter of the PointerEllipse object.
        /// </summary>
```

```
        public int Diameter
        {
            get { return (int)GetValue(DiameterProperty); }
            set { SetValue(DiameterProperty, value); }
        }
        // Using a DependencyProperty as the backing store for Diameter.
This enables animation, styling, binding, etc...
        public static readonly DependencyProperty DiameterProperty =
            DependencyProperty.Register("Diameter", typeof(int),
                typeof(PointerEllipse), new PropertyMetadata(120));
    }
}
```

Create the UI

The UI in this example is limited to the input **Canvas** where we track any pointers and render the pointer indicators and primary pointer animation (if applicable), along with a header bar containing a pointer counter and a primary pointer identifier.

Here's the MainPage.xaml:

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel x:Name="HeaderPanel"
                Orientation="Horizontal"
                Grid.Row="0">
        <StackPanel.Transitions>
            <TransitionCollection>
                <AddDeleteThemeTransition/>
            </TransitionCollection>
        </StackPanel.Transitions>
        <TextBlock x:Name="Header"
                   Text="Basic pointer tracking sample - IsPrimary"
                   Style="{ThemeResource HeaderTextBlockStyle}"
                   Margin="10,0,0,0" />
        <TextBlock x:Name="PointerCounterLabel"
                   VerticalAlignment="Center"
                   Style="{ThemeResource BodyTextBlockStyle}"
                   Text="Number of pointers: "
                   Margin="50,0,0,0"/>
        <TextBlock x:Name="PointerCounter"
                   VerticalAlignment="Center"
                   Style="{ThemeResource BodyTextBlockStyle}"
                   Text="0"
                   Margin="10,0,0,0"/>
        <TextBlock x:Name="PointerPrimaryLabel"
```

```
                    VerticalAlignment="Center"
                    Style="{ThemeResource BodyTextBlockStyle}"
                    Text="Primary: "
                    Margin="50,0,0,0"/>
        <TextBlock x:Name="PointerPrimary"
                    VerticalAlignment="Center"
                    Style="{ThemeResource BodyTextBlockStyle}"
                    Text="n/a"
                    Margin="10,0,0,0"/>
    </StackPanel>

    <Grid Grid.Row="1">
        <!--The canvas where we render the pointer UI.-->
        <Canvas x:Name="pointerCanvas"/>
    </Grid>
</Grid>
```

Handle pointer events

Finally, we define our basic pointer event handlers in the MainPage.xaml.cs code-behind. We won't reproduce the code here as the basics were covered in the previous example, but you can download the working sample from Pointer input sample (UserControl with animation).

Related articles

Topic samples

- Pointer input sample (basic)
- Pointer input sample (UserControl with animation)

Other samples

- Basic input sample
- Low latency input sample
- User interaction mode sample
- Focus visuals sample

Archive samples

- Input: XAML user input events sample
- Input: Device capabilities sample
- Input: Manipulations and gestures sample

- Input: Touch hit testing sample
- XAML scrolling, panning, and zooming sample
- Input: Simplified ink sample

Gaze interactions and eye tracking in Windows apps

Article • 05/24/2022

Provide support for tracking a user's gaze, attention, and presence based on the location and movement of their eyes.

> **Note**
>
> For gaze input in Windows Mixed Reality, see [Gaze]/windows/mixed-reality/mrtk-unity/features/input/gaze).

Important APIs: Windows.Devices.Input.Preview, GazeDevicePreview, GazePointPreview, GazeInputSourcePreview

Overview

Gaze input is a powerful way to interact and use Windows applications that is especially useful as an assistive technology for users with neuro-muscular diseases (such as ALS) and other disabilities involving impaired muscle or nerve functions.

In addition, gaze input offers equally compelling opportunities for both gaming (including target acquisition and tracking) and traditional productivity applications, kiosks, and other interactive scenarios where traditional input devices (keyboard, mouse, touch) are not available, or where it might be useful/helpful to free up the user's hands for other tasks (such as holding shopping bags).

> **Note**

Support for eye tracking hardware was introduced in **Windows 10 Fall Creators Update** along with Eye control, a built-in feature that lets you use your eyes to control the on-screen pointer, type with the on-screen keyboard, and communicate with people using text-to-speech. A set of Windows Runtime APIs (Windows.Devices.Input.Preview) for building applications that can interact with eye tracking hardware is available with **Windows 10 April 2018 Update (Version 1803, build 17134)** and newer.

Privacy

Due to the potentially sensitive personal data collected by eye tracking devices, you are required to declare the `gazeInput` capability in the app manifest of your application (see the following **Setup** section). When declared, Windows automatically prompts users with a consent dialog (when the app is first run), where the user must grant permission for the app to communicate with the eye-tracking device and access this data.

In addition, if your app collects, stores, or transfers eye tracking data, you must describe this in your app's privacy statement and follow all other requirements for **Personal Information** in the App Developer Agreement and the Microsoft Store Policies.

Setup

To use the gaze input APIs in your Windows app you'll need to:

- Specify the `gazeInput` capability in the app manifest.

  Open the **Package.appxmanifest** file with the Visual Studio manifest designer, or add the capability manually by selecting **View code**, and inserting the following `DeviceCapability` into the `Capabilities` node:

  XAML

  ```
  <Capabilities>
     <DeviceCapability Name="gazeInput" />
  </Capabilities>
  ```

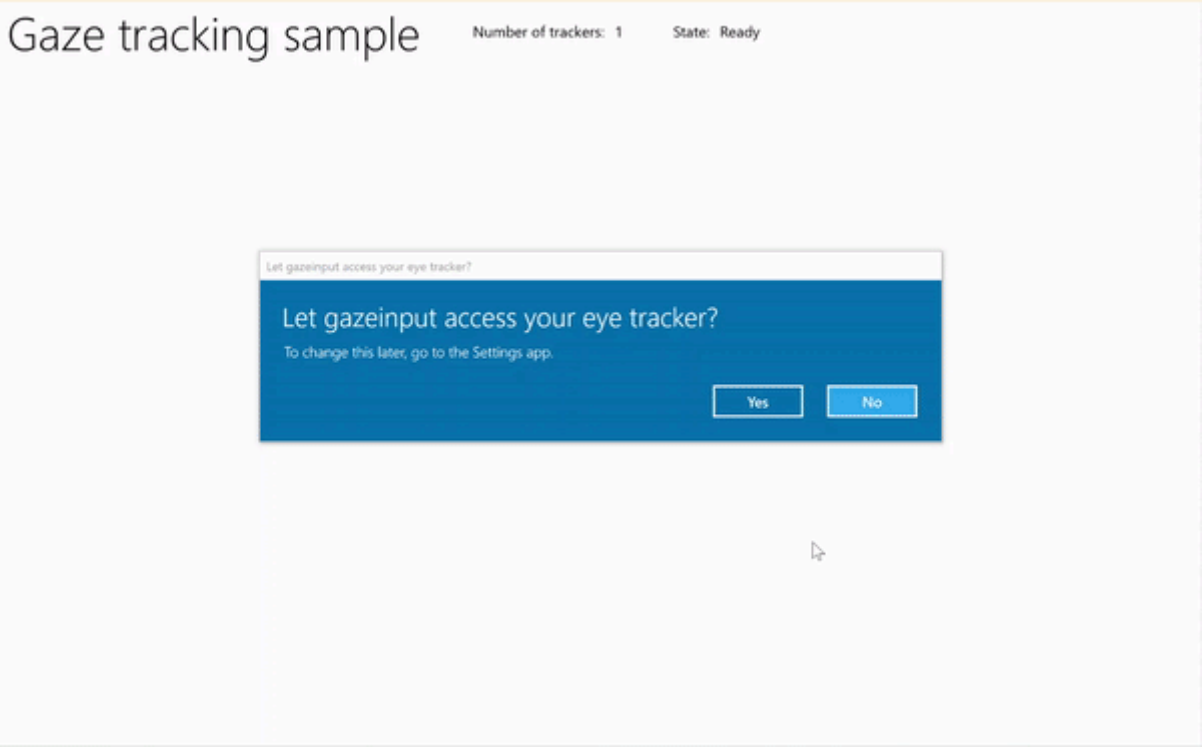
- A Windows-compatible eye-tracking device connected to your system (either built-in or peripheral) and turned on.

  See Get started with eye control in Windows 10 for a list of supported eye-tracking devices.

Basic eye tracking

In this example, we demonstrate how to track the user's gaze within a Windows app and use a timing function with basic hit testing to indicate how well they can maintain their gaze focus on a specific element.

A small ellipse is used to show where the gaze point is within the application viewport, while a RadialProgressBar from the Windows Community Toolkit is placed randomly on the canvas. When gaze focus is detected on the progress bar, a timer is started and the progress bar is randomly relocated on the canvas when the progress bar reaches 100%.



Gaze tracking with timer sample

Download this sample from Gaze input sample (basic)

1. First, we set up the UI (MainPage.xaml).

XAML

```
<Page
    x:Class="gazeinput.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:gazeinput"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    xmlns:controls="using:Microsoft.Toolkit.Uwp.UI.Controls"
    mc:Ignorable="d">
```

```
    <Grid Background="{ThemeResource
ApplicationPageBackgroundThemeBrush}">
        <Grid x:Name="containerGrid">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>
            <StackPanel x:Name="HeaderPanel"
                    Orientation="Horizontal"
                    Grid.Row="0">
                <StackPanel.Transitions>
                    <TransitionCollection>
                        <AddDeleteThemeTransition/>
                    </TransitionCollection>
                </StackPanel.Transitions>
                <TextBlock x:Name="Header"
                       Text="Gaze tracking sample"
                       Style="{ThemeResource HeaderTextBlockStyle}"
                       Margin="10,0,0,0" />
                <TextBlock x:Name="TrackerCounterLabel"
                       VerticalAlignment="Center"
                       Style="{ThemeResource BodyTextBlockStyle}"
                       Text="Number of trackers: "
                       Margin="50,0,0,0"/>
                <TextBlock x:Name="TrackerCounter"
                       VerticalAlignment="Center"
                       Style="{ThemeResource BodyTextBlockStyle}"
                       Text="0"
                       Margin="10,0,0,0"/>
                <TextBlock x:Name="TrackerStateLabel"
                       VerticalAlignment="Center"
                       Style="{ThemeResource BodyTextBlockStyle}"
                       Text="State: "
                       Margin="50,0,0,0"/>
                <TextBlock x:Name="TrackerState"
                       VerticalAlignment="Center"
                       Style="{ThemeResource BodyTextBlockStyle}"
                       Text="n/a"
                       Margin="10,0,0,0"/>
            </StackPanel>
            <Canvas x:Name="gazePositionCanvas" Grid.Row="1">
                <controls:RadialProgressBar
                    x:Name="GazeRadialProgressBar"
                    Value="0"
                    Foreground="Blue"
                    Background="White"
                    Thickness="4"
                    Minimum="0"
                    Maximum="100"
                    Width="100"
                    Height="100"
                    Outline="Gray"
                    Visibility="Collapsed"/>
                <Ellipse
                    x:Name="eyeGazePositionEllipse"
```

```
                    Width="20" Height="20"
                    Fill="Blue"
                    Opacity="0.5"
                    Visibility="Collapsed">
                </Ellipse>
            </Canvas>
        </Grid>
    </Grid>
</Page>
```

2. Next, we initialize our app.

   In this snippet, we declare our global objects and override the OnNavigatedTo page event to start our gaze device watcher and the OnNavigatedFrom page event to stop our gaze device watcher.

C#

```
using System;
using Windows.Devices.Input.Preview;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml;
using Windows.Foundation;
using System.Collections.Generic;
using Windows.UI.Xaml.Media;
using Windows.UI.Xaml.Navigation;

namespace gazeinput
{
    public sealed partial class MainPage : Page
    {
        /// <summary>
        /// Reference to the user's eyes and head as detected
        /// by the eye-tracking device.
        /// </summary>
        private GazeInputSourcePreview gazeInputSource;

        /// <summary>
        /// Dynamic store of eye-tracking devices.
        /// </summary>
        /// <remarks>
        /// Receives event notifications when a device is added,
removed,
        /// or updated after the initial enumeration.
        /// </remarks>
        private GazeDeviceWatcherPreview gazeDeviceWatcher;

        /// <summary>
        /// Eye-tracking device counter.
        /// </summary>
        private int deviceCounter = 0;
```

```
        /// <summary>
        /// Timer for gaze focus on RadialProgressBar.
        /// </summary>
        DispatcherTimer timerGaze = new DispatcherTimer();

        /// <summary>
        /// Tracker used to prevent gaze timer restarts.
        /// </summary>
        bool timerStarted = false;

        /// <summary>
        /// Initialize the app.
        /// </summary>
        public MainPage()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Override of OnNavigatedTo page event starts
GazeDeviceWatcher.
        /// </summary>
        /// <param name="e">Event args for the NavigatedTo
event</param>
        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            // Start listening for device events on navigation to eye-
tracking page.
            StartGazeDeviceWatcher();
        }

        /// <summary>
        /// Override of OnNavigatedFrom page event stops
GazeDeviceWatcher.
        /// </summary>
        /// <param name="e">Event args for the NavigatedFrom
event</param>
        protected override void OnNavigatedFrom(NavigationEventArgs e)
        {
            // Stop listening for device events on navigation from eye-
tracking page.
            StopGazeDeviceWatcher();
        }
    }
}
```

3. Next, we add our gaze device watcher methods.

   In `StartGazeDeviceWatcher`, we call CreateWatcher and declare the eye-tracking device event listeners (DeviceAdded, DeviceUpdated, and DeviceRemoved).

In `DeviceAdded`, we check the state of the eye-tracking device. If a viable device, we increment our device count and enable gaze tracking. See next step for details.

In `DeviceUpdated`, we also enable gaze tracking as this event is triggered if a device is recalibrated.

In `DeviceRemoved`, we decrement our device counter and remove the device event handlers.

In `StopGazeDeviceWatcher`, we shut down the gaze device watcher.

C#

```
    /// <summary>
    /// Start gaze watcher and declare watcher event handlers.
    /// </summary>
    private void StartGazeDeviceWatcher()
    {
        if (gazeDeviceWatcher == null)
        {
            gazeDeviceWatcher = GazeInputSourcePreview.CreateWatcher();
            gazeDeviceWatcher.Added += this.DeviceAdded;
            gazeDeviceWatcher.Updated += this.DeviceUpdated;
            gazeDeviceWatcher.Removed += this.DeviceRemoved;
            gazeDeviceWatcher.Start();
        }
    }

    /// <summary>
    /// Shut down gaze watcher and stop listening for events.
    /// </summary>
    private void StopGazeDeviceWatcher()
    {
        if (gazeDeviceWatcher != null)
        {
            gazeDeviceWatcher.Stop();
            gazeDeviceWatcher.Added -= this.DeviceAdded;
            gazeDeviceWatcher.Updated -= this.DeviceUpdated;
            gazeDeviceWatcher.Removed -= this.DeviceRemoved;
            gazeDeviceWatcher = null;
        }
    }

    /// <summary>
    /// Eye-tracking device connected (added, or available when watcher is
initialized).
    /// </summary>
    /// <param name="sender">Source of the device added event</param>
    /// <param name="e">Event args for the device added event</param>
    private void DeviceAdded(GazeDeviceWatcherPreview source,
        GazeDeviceWatcherAddedPreviewEventArgs args)
    {
```

```
        if (IsSupportedDevice(args.Device))
        {
            deviceCounter++;
            TrackerCounter.Text = deviceCounter.ToString();
        }
        // Set up gaze tracking.
        TryEnableGazeTrackingAsync(args.Device);
    }

    /// <summary>
    /// Initial device state might be uncalibrated,
    /// but device was subsequently calibrated.
    /// </summary>
    /// <param name="sender">Source of the device updated event</param>
    /// <param name="e">Event args for the device updated event</param>
    private void DeviceUpdated(GazeDeviceWatcherPreview source,
        GazeDeviceWatcherUpdatedPreviewEventArgs args)
    {
        // Set up gaze tracking.
        TryEnableGazeTrackingAsync(args.Device);
    }

    /// <summary>
    /// Handles disconnection of eye-tracking devices.
    /// </summary>
    /// <param name="sender">Source of the device removed event</param>
    /// <param name="e">Event args for the device removed event</param>
    private void DeviceRemoved(GazeDeviceWatcherPreview source,
        GazeDeviceWatcherRemovedPreviewEventArgs args)
    {
        // Decrement gaze device counter and remove event handlers.
        if (IsSupportedDevice(args.Device))
        {
            deviceCounter--;
            TrackerCounter.Text = deviceCounter.ToString();

            if (deviceCounter == 0)
            {
                gazeInputSource.GazeEntered -= this.GazeEntered;
                gazeInputSource.GazeMoved -= this.GazeMoved;
                gazeInputSource.GazeExited -= this.GazeExited;
            }
        }
    }
```

4. Here, we check if the device is viable in `IsSupportedDevice` and, if so, attempt to enable gaze tracking in `TryEnableGazeTrackingAsync`.

   In `TryEnableGazeTrackingAsync`, we declare the gaze event handlers, and call GazeInputSourcePreview.GetForCurrentView() to get a reference to the input source (this must be called on the UI thread, see Keep the UI thread responsive).

> **Note**
>
> You should call GazeInputSourcePreview.GetForCurrentView() only when a compatible eye-tracking device is connected and required by your application. Otherwise, the consent dialog is unnecessary.

C#

```
    /// <summary>
    /// Initialize gaze tracking.
    /// </summary>
    /// <param name="gazeDevice"></param>
    private async void TryEnableGazeTrackingAsync(GazeDevicePreview
gazeDevice)
    {
        // If eye-tracking device is ready, declare event handlers and start
tracking.
        if (IsSupportedDevice(gazeDevice))
        {
            timerGaze.Interval = new TimeSpan(0, 0, 0, 0, 20);
            timerGaze.Tick += TimerGaze_Tick;

            SetGazeTargetLocation();

            // This must be called from the UI thread.
            gazeInputSource = GazeInputSourcePreview.GetForCurrentView();

            gazeInputSource.GazeEntered += GazeEntered;
            gazeInputSource.GazeMoved += GazeMoved;
            gazeInputSource.GazeExited += GazeExited;
        }
        // Notify if device calibration required.
        else if (gazeDevice.ConfigurationState ==

GazeDeviceConfigurationStatePreview.UserCalibrationNeeded ||
                    gazeDevice.ConfigurationState ==
                    GazeDeviceConfigurationStatePreview.ScreenSetupNeeded)
        {
            // Device isn't calibrated, so invoke the calibration handler.
            System.Diagnostics.Debug.WriteLine(
                "Your device needs to calibrate. Please wait for it to
finish.");
            await gazeDevice.RequestCalibrationAsync();
        }
        // Notify if device calibration underway.
        else if (gazeDevice.ConfigurationState ==
            GazeDeviceConfigurationStatePreview.Configuring)
        {
            // Device is currently undergoing calibration.
            // A device update is sent when calibration complete.
            System.Diagnostics.Debug.WriteLine(
```

```
                "Your device is being configured. Please wait for it to
finish");
        }
        // Device is not viable.
        else if (gazeDevice.ConfigurationState ==
GazeDeviceConfigurationStatePreview.Unknown)
        {
            // Notify if device is in unknown state.
            // Reconfigure/recalbirate the device.
            System.Diagnostics.Debug.WriteLine(
                "Your device is not ready. Please set up your device or
reconfigure it.");
        }
    }

    /// <summary>
    /// Check if eye-tracking device is viable.
    /// </summary>
    /// <param name="gazeDevice">Reference to eye-tracking device.</param>
    /// <returns>True, if device is viable; otherwise, false.</returns>
    private bool IsSupportedDevice(GazeDevicePreview gazeDevice)
    {
        TrackerState.Text = gazeDevice.ConfigurationState.ToString();
        return (gazeDevice.CanTrackEyes &&
                    gazeDevice.ConfigurationState ==
                    GazeDeviceConfigurationStatePreview.Ready);
    }
```

5. Next, we set up our gaze event handlers.

   We display and hide the gaze tracking ellipse in `GazeEntered` and `GazeExited`, respectively.

   In `GazeMoved`, we move our gaze tracking ellipse based on the EyeGazePosition provided by the CurrentPoint of the GazeEnteredPreviewEventArgs. We also manage the gaze focus timer on the RadialProgressBar, which triggers repositioning of the progress bar. See next step for details.

   C#

```
/// <summary>
/// GazeEntered handler.
/// </summary>
/// <param name="sender">Source of the gaze entered event</param>
/// <param name="e">Event args for the gaze entered event</param>
private void GazeEntered(
    GazeInputSourcePreview sender,
    GazeEnteredPreviewEventArgs args)
{
    // Show ellipse representing gaze point.
    eyeGazePositionEllipse.Visibility = Visibility.Visible;
```

```
    // Mark the event handled.
    args.Handled = true;
}

/// <summary>
/// GazeExited handler.
/// Call DisplayRequest.RequestRelease to conclude the
/// RequestActive called in GazeEntered.
/// </summary>
/// <param name="sender">Source of the gaze exited event</param>
/// <param name="e">Event args for the gaze exited event</param>
private void GazeExited(
    GazeInputSourcePreview sender,
    GazeExitedPreviewEventArgs args)
{
    // Hide gaze tracking ellipse.
    eyeGazePositionEllipse.Visibility = Visibility.Collapsed;

    // Mark the event handled.
    args.Handled = true;
}

/// <summary>
/// GazeMoved handler translates the ellipse on the canvas to reflect
gaze point.
/// </summary>
/// <param name="sender">Source of the gaze moved event</param>
/// <param name="e">Event args for the gaze moved event</param>
private void GazeMoved(GazeInputSourcePreview sender,
GazeMovedPreviewEventArgs args)
{
    // Update the position of the ellipse corresponding to gaze point.
    if (args.CurrentPoint.EyeGazePosition != null)
    {
        double gazePointX = args.CurrentPoint.EyeGazePosition.Value.X;
        double gazePointY = args.CurrentPoint.EyeGazePosition.Value.Y;

        double ellipseLeft =
            gazePointX -
            (eyeGazePositionEllipse.Width / 2.0f);
        double ellipseTop =
            gazePointY -
            (eyeGazePositionEllipse.Height / 2.0f) -
            (int)Header.ActualHeight;

        // Translate transform for moving gaze ellipse.
        TranslateTransform translateEllipse = new TranslateTransform
        {
            X = ellipseLeft,
            Y = ellipseTop
        };

        eyeGazePositionEllipse.RenderTransform = translateEllipse;
```

```
        // The gaze point screen location.
        Point gazePoint = new Point(gazePointX, gazePointY);

        // Basic hit test to determine if gaze point is on progress
bar.
        bool hitRadialProgressBar =
            DoesElementContainPoint(
                gazePoint,
                GazeRadialProgressBar.Name,
                GazeRadialProgressBar);

        // Use progress bar thickness for visual feedback.
        if (hitRadialProgressBar)
        {
            GazeRadialProgressBar.Thickness = 10;
        }
        else
        {
            GazeRadialProgressBar.Thickness = 4;
        }

        // Mark the event handled.
        args.Handled = true;
    }
}
```

6. Finally, here are the methods used to manage the gaze focus timer for this app.

`DoesElementContainPoint` checks if the gaze pointer is over the progress bar. If so, it starts the gaze timer and increments the progress bar on each gaze timer tick.

`SetGazeTargetLocation` sets the initial location of the progress bar and, if the progress bar completes (depending on the gaze focus timer), moves the progress bar to a random location.

C#

```
/// <summary>
/// Return whether the gaze point is over the progress bar.
/// </summary>
/// <param name="gazePoint">The gaze point screen location</param>
/// <param name="elementName">The progress bar name</param>
/// <param name="uiElement">The progress bar UI element</param>
/// <returns></returns>
private bool DoesElementContainPoint(
    Point gazePoint, string elementName, UIElement uiElement)
{
    // Use entire visual tree of progress bar.
    IEnumerable<UIElement> elementStack =
        VisualTreeHelper.FindElementsInHostCoordinates(gazePoint,
uiElement, true);
    foreach (UIElement item in elementStack)
```

```
    {
        //Cast to FrameworkElement and get element name.
        if (item is FrameworkElement feItem)
        {
            if (feItem.Name.Equals(elementName))
            {
                if (!timerStarted)
                {
                    // Start gaze timer if gaze over element.
                    timerGaze.Start();
                    timerStarted = true;
                }
                return true;
            }
        }
    }

    // Stop gaze timer and reset progress bar if gaze leaves element.
    timerGaze.Stop();
    GazeRadialProgressBar.Value = 0;
    timerStarted = false;
    return false;
}

/// <summary>
/// Tick handler for gaze focus timer.
/// </summary>
/// <param name="sender">Source of the gaze entered event</param>
/// <param name="e">Event args for the gaze entered event</param>
private void TimerGaze_Tick(object sender, object e)
{
    // Increment progress bar.
    GazeRadialProgressBar.Value += 1;

    // If progress bar reaches maximum value, reset and relocate.
    if (GazeRadialProgressBar.Value == 100)
    {
        SetGazeTargetLocation();
    }
}

/// <summary>
/// Set/reset the screen location of the progress bar.
/// </summary>
private void SetGazeTargetLocation()
{
    // Ensure the gaze timer restarts on new progress bar location.
    timerGaze.Stop();
    timerStarted = false;

    // Get the bounding rectangle of the app window.
    Rect appBounds =
Windows.UI.ViewManagement.ApplicationView.GetForCurrentView().VisibleBo
unds;
```

```
    // Translate transform for moving progress bar.
    TranslateTransform translateTarget = new TranslateTransform();

    // Calculate random location within gaze canvas.
        Random random = new Random();
        int randomX =
            random.Next(
                0,
                (int)appBounds.Width -
(int)GazeRadialProgressBar.Width);
        int randomY =
            random.Next(
                0,
                (int)appBounds.Height -
(int)GazeRadialProgressBar.Height - (int)Header.ActualHeight);

    translateTarget.X = randomX;
    translateTarget.Y = randomY;

    GazeRadialProgressBar.RenderTransform = translateTarget;

    // Show progress bar.
    GazeRadialProgressBar.Visibility = Visibility.Visible;
    GazeRadialProgressBar.Value = 0;
}
```

See also

Resources

- Windows Community Toolkit Gaze library

Topic samples

- Gaze sample (basic) (C#)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

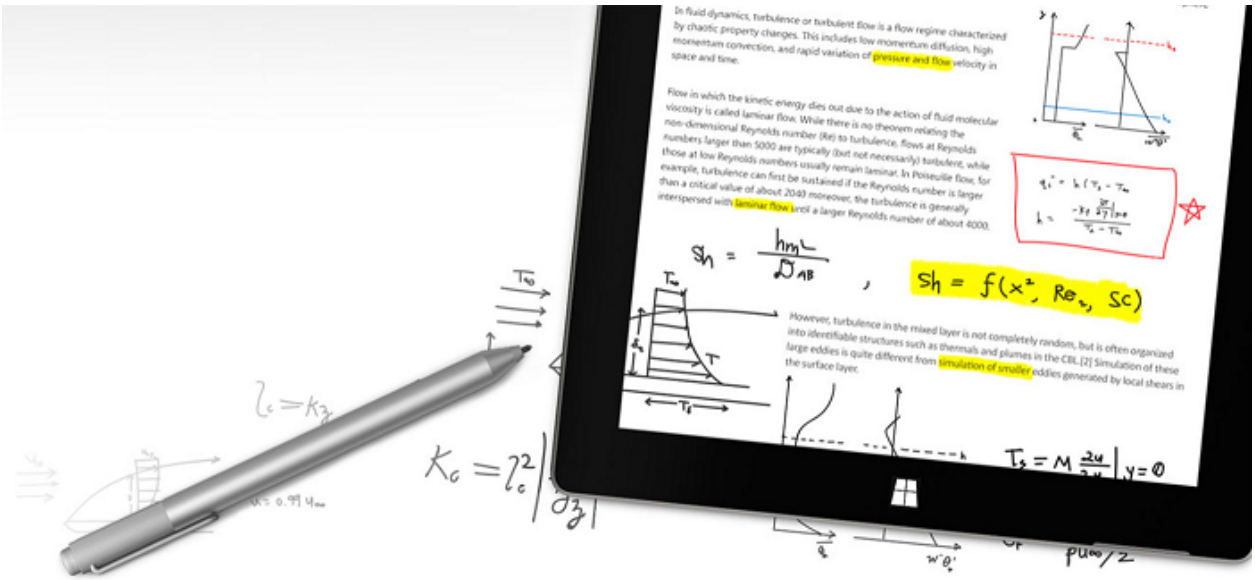
Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Pen interactions and Windows Ink in Windows apps

Article • 12/15/2021



Surface Pen (available for purchase at the Microsoft Store).

Overview

Optimize your Windows app for pen input to provide both standard **pointer device** functionality and the best Windows Ink experience for your users.

> **Note**
>
> This topic focuses on the Windows Ink platform. For general pointer input handling (similar to mouse, touch, and touchpad), see **Handle pointer input**.

Using ink in your Windows app

Use Windows Pen and Ink to build more engaging enterprise apps

The Windows Ink platform, together with a pen device, provides a natural way to create digital handwritten notes, drawings, and annotations. The platform supports capturing digitizer input as ink data, generating ink data, managing ink data, rendering ink data as ink strokes on the output device, and converting ink to text through handwriting recognition.

In addition to capturing the basic position and movement of the pen as the user writes or draws, your app can also track and collect the varying amounts of pressure used throughout a stroke. This information, along with settings for pen tip shape, size, and rotation, ink color, and purpose (plain ink, erasing, highlighting, and selecting), enables you to provide user experiences that closely resemble writing or drawing on paper with a pen, pencil, or brush.

> **Note**
>
> Your app can also support ink input from other pointer-based devices, including touch digitizers and mouse devices.

The ink platform is very flexible. It is designed to support various levels of functionality, depending on your requirements.

For Windows Ink UX guidelines, see Inking controls.

Components of the Windows Ink platform

Component	Description
InkCanvas	A XAML UI platform control that, by default, receives and displays all input from a pen as either an ink stroke or an erase stroke. For more information about how to use the InkCanvas, see Recognize Windows Ink strokes as text and Store and retrieve Windows Ink stroke data.
InkPresenter	A code-behind object, instantiated along with an **InkCanvas** control (exposed through the **InkCanvas.InkPresenter** property). This object provides all default inking functionality exposed by the **InkCanvas**, along with a comprehensive set of APIs for additional customization and personalization. For more information about how to use the InkPresenter, see Recognize Windows Ink strokes as text and Store and retrieve Windows Ink stroke data.
InkToolbar	A XAML UI platform control containing a customizable and extensible collection of buttons that activate ink-related features in an associated **InkCanvas**. For more information about how to use the InkToolbar, see Add an InkToolbar to a Windows app inking app.
IInkD2DRenderer	Enables the rendering of ink strokes onto the designated Direct2D device context of a Universal Windows app, instead of the default **InkCanvas** control. This enables full customization of the inking experience. For more information, see the Complex ink sample.

Basic inking with InkCanvas

To add basic inking functionality, just place an **InkCanvas** UWP platform control on the appropriate page in your app.

By default, the **InkCanvas** supports ink input only from a pen. The input is either rendered as an ink stroke using default settings for color and thickness (a black ballpoint pen with a thickness of 2 pixels), or treated as a stroke eraser (when input is from an eraser tip or the pen tip modified with an erase button).

> **Note**
>
> If an eraser tip or button is not present, the InkCanvas can be configured to process input from the pen tip as an erase stroke.

In this example, an **InkCanvas** overlays a background image.

> **Note**
>
> An InkCanvas has default **Height** and **Width** properties of zero, unless it is the child of an element that automatically sizes its child elements, such as **StackPanel** or **Grid** controls.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel x:Name="HeaderPanel" Orientation="Horizontal" Grid.Row="0">
        <TextBlock x:Name="Header"
                   Text="Basic ink sample"
                   Style="{ThemeResource HeaderTextBlockStyle}"
                   Margin="10,0,0,0" />
    </StackPanel>
    <Grid Grid.Row="1">
        <Image Source="Assets\StoreLogo.png" />
        <InkCanvas x:Name="inkCanvas" />
    </Grid>
</Grid>
```

This series of images shows how pen input is rendered by this **InkCanvas** control.

The blank **InkCanvas** with a background image.	The **InkCanvas** with ink strokes.	The **InkCanvas** with one stroke erased (note how erase operates on an entire stroke, not a portion).

The inking functionality supported by the **InkCanvas** control is provided by a code-behind object called the **InkPresenter**.

For basic inking, you don't have to concern yourself with the **InkPresenter**. However, to customize and configure inking behavior on the **InkCanvas**, you must access its corresponding **InkPresenter** object.

Basic customization with InkPresenter

An **InkPresenter** object is instantiated with each **InkCanvas** control.

> **Note**
>
> The **InkPresenter** cannot be instantiated directly. Instead, it is accessed through the **InkPresenter** property of the **InkCanvas**.

Along with providing all default inking behaviors of its corresponding **InkCanvas** control, the **InkPresenter** provides a comprehensive set of APIs for additional stroke customization and finer-grained management of the pen input (standard and modified). This includes stroke properties, supported input device types, and whether input is processed by the object or passed to the app for processing.

> **Note**
>
> Standard ink input (from either pen tip or eraser tip/button) is not modified with a secondary hardware affordance, such as a pen barrel button, right mouse button, or similar mechanism.

By default, ink is supported for pen input only. Here, we configure the **InkPresenter** to interpret input data from both pen and mouse as ink strokes. We also set some initial ink stroke attributes used for rendering strokes to the **InkCanvas**.

To enable mouse and touch inking, set the **InputDeviceTypes** property of the **InkPresenter** to the combination of **CoreInputDeviceTypes** values that you want.

C#

```
public MainPage()
{
    this.InitializeComponent();

    // Set supported inking device types.
    inkCanvas.InkPresenter.InputDeviceTypes =
        Windows.UI.Core.CoreInputDeviceTypes.Mouse |
        Windows.UI.Core.CoreInputDeviceTypes.Pen;

    // Set initial ink stroke attributes.
    InkDrawingAttributes drawingAttributes = new InkDrawingAttributes();
    drawingAttributes.Color = Windows.UI.Colors.Black;
    drawingAttributes.IgnorePressure = false;
    drawingAttributes.FitToCurve = true;

inkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(drawingAttributes);
}
```

Ink stroke attributes can be set dynamically to accommodate user preferences or app requirements.

Here, we let a user choose from a list of ink colors.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel x:Name="HeaderPanel" Orientation="Horizontal" Grid.Row="0">
        <TextBlock x:Name="Header"
                   Text="Basic ink customization sample"
                   VerticalAlignment="Center"
                   Style="{ThemeResource HeaderTextBlockStyle}"
                   Margin="10,0,0,0" />
        <TextBlock Text="Color:"
                   Style="{StaticResource SubheaderTextBlockStyle}"
                   VerticalAlignment="Center"
                   Margin="50,0,10,0"/>
        <ComboBox x:Name="PenColor"
                  VerticalAlignment="Center"
```

```
                    SelectedIndex="0"
                    SelectionChanged="OnPenColorChanged">
                <ComboBoxItem Content="Black"/>
                <ComboBoxItem Content="Red"/>
            </ComboBox>
        </StackPanel>
        <Grid Grid.Row="1">
            <Image Source="Assets\StoreLogo.png" />
            <InkCanvas x:Name="inkCanvas" />
        </Grid>
    </Grid>
```

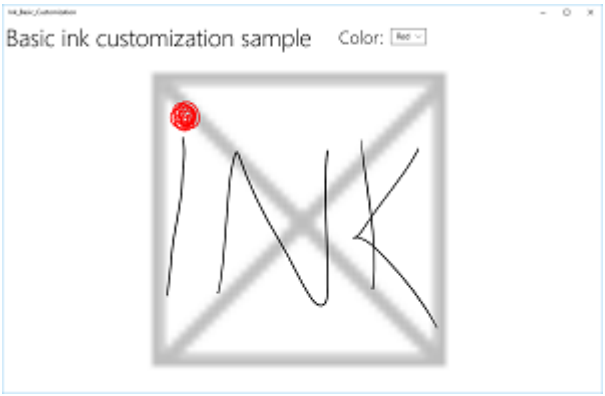
We then handle changes to the selected color and update the ink stroke attributes accordingly.

C#

```
// Update ink stroke color for new strokes.
private void OnPenColorChanged(object sender, SelectionChangedEventArgs e)
{
    if (inkCanvas != null)
    {
        InkDrawingAttributes drawingAttributes =
            inkCanvas.InkPresenter.CopyDefaultDrawingAttributes();

        string value =
((ComboBoxItem)PenColor.SelectedItem).Content.ToString();

        switch (value)
        {
            case "Black":
                drawingAttributes.Color = Windows.UI.Colors.Black;
                break;
            case "Red":
                drawingAttributes.Color = Windows.UI.Colors.Red;
                break;
            default:
                drawingAttributes.Color = Windows.UI.Colors.Black;
                break;
        };


inkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(drawingAttributes);
    }
}
```

These images shows how pen input is processed and customized by the **InkPresenter**.

The InkCanvas with default black ink strokes.



The InkCanvas with user selected red ink strokes.

To provide functionality beyond inking and erasing, such as stroke selection, your app must identify specific input for the InkPresenter to pass through unprocessed for handling by your app.

Pass-through input for advanced processing

By default, InkPresenter processes all input as either an ink stroke or an erase stroke, including input modified by a secondary hardware affordance such as a pen barrel button, a right mouse button, or similar. However, users typically expect some additional functionality or modified behavior with these secondary affordances.

In some cases, you might also need to expose additional functionality for pens without secondary affordances (functionality not usually associated with the pen tip), other input device types, or some type of modified behavior based on a user selection in your app's UI.

To support this, InkPresenter can be configured to leave specific input unprocessed. This unprocessed input is then passed through to your app for processing.

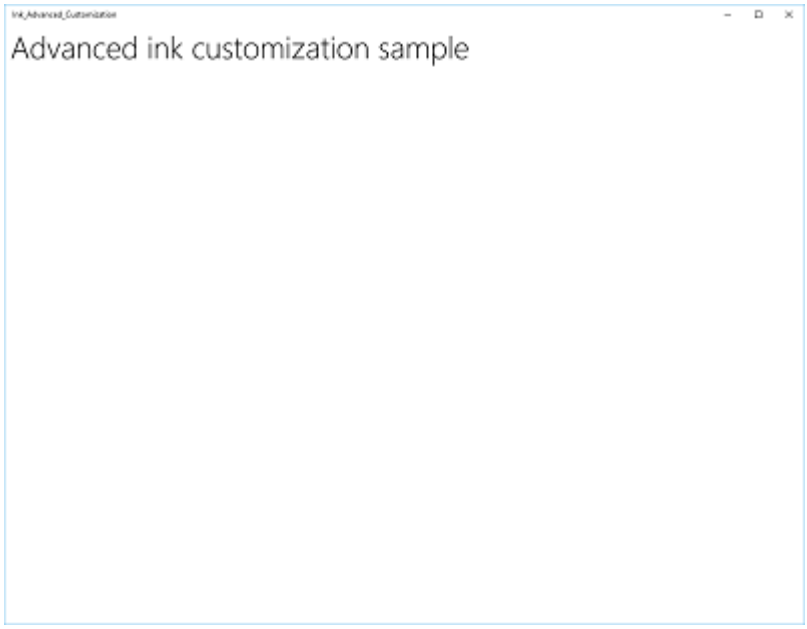
Example - Use unprocessed input to implement stroke selection

The Windows Ink platform does not provide built-in support for actions that require modified input, such as stroke selection. To support features like this, you must provide a custom solution in your apps.

The following code example (all code is in the MainPage.xaml and MainPage.xaml.cs files) steps through how to enable stroke selection when input is modified with a pen barrel button (or right mouse button).

1. First, we set up the UI in MainPage.xaml.

   Here, we add a canvas (below the **InkCanvas**) to draw the selection stroke. Using a separate layer to draw the selection stroke leaves the **InkCanvas** and its content untouched.



XAML

```
  <Grid Background="{ThemeResource
ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel x:Name="HeaderPanel" Orientation="Horizontal"
Grid.Row="0">
      <TextBlock x:Name="Header"
        Text="Advanced ink customization sample"
        VerticalAlignment="Center"
        Style="{ThemeResource HeaderTextBlockStyle}"
        Margin="10,0,0,0" />
    </StackPanel>
```

```
    <Grid Grid.Row="1">
      <!-- Canvas for displaying selection UI. -->
      <Canvas x:Name="selectionCanvas"/>
      <!-- Inking area -->
      <InkCanvas x:Name="inkCanvas"/>
    </Grid>
  </Grid>
```

2. In MainPage.xaml.cs, we declare a couple of global variables for keeping references to aspects of the selection UI. Specifically, the selection lasso stroke and the bounding rectangle that highlights the selected strokes.

```
C#
```

```
// Stroke selection tool.
private Polyline lasso;
// Stroke selection area.
private Rect boundingRect;
```

3. Next, we configure the **InkPresenter** to interpret input data from both pen and mouse as ink strokes, and set some initial ink stroke attributes used for rendering strokes to the **InkCanvas**.

   Most importantly, we use the **InputProcessingConfiguration** property of the InkPresenter to indicate that any modified input should be processed by the app. Modified input is specified by assigning **InputProcessingConfiguration.RightDragAction** a value of **InkInputRightDragAction.LeaveUnprocessed**. When this value is set, the InkPresenter passes through to the InkUnprocessedInput class, a set of pointer events for you to handle.

   We assign listeners for the unprocessed **PointerPressed**, **PointerMoved**, and **PointerReleased** events passed through by the **InkPresenter**. All selection functionality is implemented in the handlers for these events.

   Finally, we assign listeners for the **StrokeStarted** and **StrokesErased** events of the **InkPresenter**. We use the handlers for these events to clean up the selection UI if a new stroke is started or an existing stroke is erased.

C#

```
  public MainPage()
  {
    this.InitializeComponent();

    // Set supported inking device types.
    inkCanvas.InkPresenter.InputDeviceTypes =
      Windows.UI.Core.CoreInputDeviceTypes.Mouse |
      Windows.UI.Core.CoreInputDeviceTypes.Pen;

    // Set initial ink stroke attributes.
    InkDrawingAttributes drawingAttributes = new
InkDrawingAttributes();
    drawingAttributes.Color = Windows.UI.Colors.Black;
    drawingAttributes.IgnorePressure = false;
    drawingAttributes.FitToCurve = true;

inkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(drawingAttributes
);

    // By default, the InkPresenter processes input modified by
    // a secondary affordance (pen barrel button, right mouse
    // button, or similar) as ink.
    // To pass through modified input to the app for custom processing
    // on the app UI thread instead of the background ink thread, set
    // InputProcessingConfiguration.RightDragAction to
LeaveUnprocessed.
    inkCanvas.InkPresenter.InputProcessingConfiguration.RightDragAction
=
        InkInputRightDragAction.LeaveUnprocessed;

    // Listen for unprocessed pointer events from modified input.
    // The input is used to provide selection functionality.
    inkCanvas.InkPresenter.UnprocessedInput.PointerPressed +=
        UnprocessedInput_PointerPressed;
    inkCanvas.InkPresenter.UnprocessedInput.PointerMoved +=
```

```
        UnprocessedInput_PointerMoved;
    inkCanvas.InkPresenter.UnprocessedInput.PointerReleased +=
        UnprocessedInput_PointerReleased;

    // Listen for new ink or erase strokes to clean up selection UI.
    inkCanvas.InkPresenter.StrokeInput.StrokeStarted +=
        StrokeInput_StrokeStarted;
    inkCanvas.InkPresenter.StrokesErased +=
        InkPresenter_StrokesErased;
}
```

4. We then define handlers for the unprocessed **PointerPressed**, **PointerMoved**, and **PointerReleased** events passed through by the **InkPresenter**.

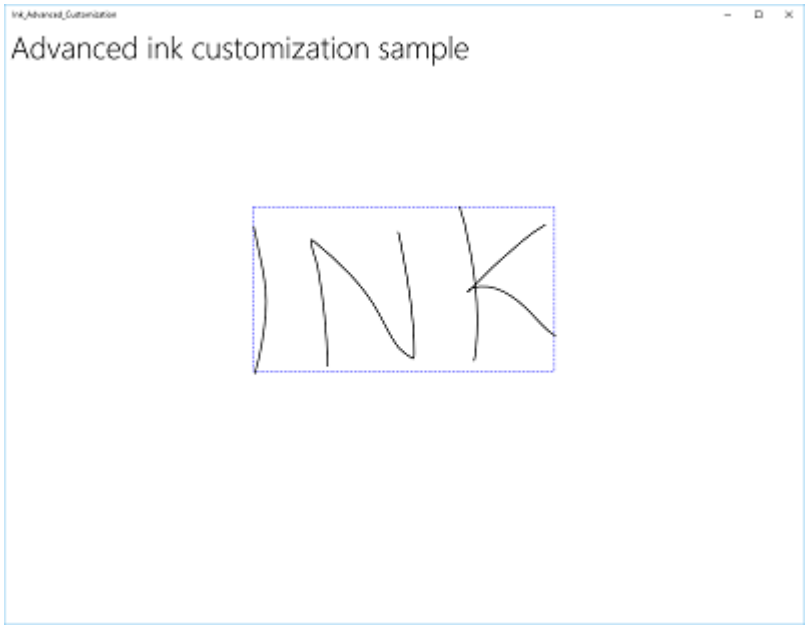
   All selection functionality is implemented in these handlers, including the lasso stroke and the bounding rectangle.

C#

```
// Handle unprocessed pointer events from modified input.
// The input is used to provide selection functionality.
// Selection UI is drawn on a canvas under the InkCanvas.
private void UnprocessedInput_PointerPressed(
    InkUnprocessedInput sender, PointerEventArgs args)
{
    // Initialize a selection lasso.
    lasso = new Polyline()
    {
        Stroke = new SolidColorBrush(Windows.UI.Colors.Blue),
        StrokeThickness = 1,
        StrokeDashArray = new DoubleCollection() { 5, 2 },
        };

        lasso.Points.Add(args.CurrentPoint.RawPosition);
```

```
            selectionCanvas.Children.Add(lasso);
        }

        private void UnprocessedInput_PointerMoved(
          InkUnprocessedInput sender, PointerEventArgs args)
        {
          // Add a point to the lasso Polyline object.
          lasso.Points.Add(args.CurrentPoint.RawPosition);
        }

        private void UnprocessedInput_PointerReleased(
          InkUnprocessedInput sender, PointerEventArgs args)
        {
          // Add the final point to the Polyline object and
          // select strokes within the lasso area.
          // Draw a bounding box on the selection canvas
          // around the selected ink strokes.
          lasso.Points.Add(args.CurrentPoint.RawPosition);

          boundingRect =
            inkCanvas.InkPresenter.StrokeContainer.SelectWithPolyLine(
              lasso.Points);

          DrawBoundingRect();
        }
```

5. To conclude the PointerReleased event handler, we clear the selection layer of all content (the lasso stroke) and then draw a single bounding rectangle around the ink strokes encompassed by the lasso area.



C#

```
// Draw a bounding rectangle, on the selection canvas, encompassing
// all ink strokes within the lasso area.
private void DrawBoundingRect()
```

```
{
  // Clear all existing content from the selection canvas.
  selectionCanvas.Children.Clear();

  // Draw a bounding rectangle only if there are ink strokes
  // within the lasso area.
  if (!((boundingRect.Width == 0) ||
    (boundingRect.Height == 0) ||
    boundingRect.IsEmpty))
    {
      var rectangle = new Rectangle()
      {
        Stroke = new SolidColorBrush(Windows.UI.Colors.Blue),
          StrokeThickness = 1,
          StrokeDashArray = new DoubleCollection() { 5, 2 },
          Width = boundingRect.Width,
          Height = boundingRect.Height
      };

      Canvas.SetLeft(rectangle, boundingRect.X);
      Canvas.SetTop(rectangle, boundingRect.Y);

      selectionCanvas.Children.Add(rectangle);
    }
  }
```

6. Finally, we define handlers for the StrokeStarted and StrokesErased InkPresenter events.

   These both just call the same cleanup function to clear the current selection whenever a new stroke is detected.

```
C#

  // Handle new ink or erase strokes to clean up selection UI.
  private void StrokeInput_StrokeStarted(
    InkStrokeInput sender, Windows.UI.Core.PointerEventArgs args)
  {
    ClearSelection();
  }

  private void InkPresenter_StrokesErased(
    InkPresenter sender, InkStrokesErasedEventArgs args)
  {
    ClearSelection();
  }
```

7. Here's the function to remove all selection UI from the selection canvas when a new stroke is started or an existing stroke is erased.

C#

```
// Clean up selection UI.
private void ClearSelection()
{
  var strokes = inkCanvas.InkPresenter.StrokeContainer.GetStrokes();
  foreach (var stroke in strokes)
  {
    stroke.Selected = false;
  }
  ClearDrawnBoundingRect();
 }

private void ClearDrawnBoundingRect()
{
  if (selectionCanvas.Children.Any())
  {
    selectionCanvas.Children.Clear();
    boundingRect = Rect.Empty;
  }
}
```

Custom ink rendering

By default, ink input is processed on a low-latency background thread and rendered in-progress, or "wet", as it is drawn. When the stroke is completed (pen or finger lifted, or mouse button released), the stroke is processed on the UI thread and rendered "dry" to the **InkCanvas** layer (above the application content and replacing the wet ink).

You can override this default behavior and completely control the inking experience by "custom drying" the wet ink strokes. While the default behavior is typically sufficient for most applications, there are a few cases where custom drying might be required, these include:

- More efficient management of large, or complex, collections of ink strokes
- More efficient panning and zooming support on large ink canvases
- Interleaving ink and other objects, such as shapes or text, while maintaining z-order
- Drying and converting ink synchronously into a DirectX shape (for example, a straight line or shape rasterized and integrated into application content instead of as a separate **InkCanvas** layer).

Custom drying requires an **IInkD2DRenderer** object to manage the ink input and render it to the Direct2D device context of your Universal Windows app, instead of the default **InkCanvas** control.

By calling **ActivateCustomDrying** (before the **InkCanvas** is loaded), an app creates an **InkSynchronizer** object to customize how an ink stroke is rendered dry to a **SurfaceImageSource** or **VirtualSurfaceImageSource**.

Both **SurfaceImageSource** and **VirtualSurfaceImageSource** provide a DirectX shared surface for your app to draw into and compose into your application's content, although VSIS provides a virtual surface that's larger than the screen for performant panning and zooming. Because visual updates to these surfaces are synchronized with the XAML UI thread, when ink is rendered to either, the wet ink can be removed from the InkCanvas simultaneously.

You can also custom dry ink to a SwapChainPanel, but synchronization with the UI thread is not guaranteed and there might be a delay between when the ink is rendered to your SwapChainPanel and when ink is removed from the InkCanvas.

For a full example of this functionality, see the Complex ink sample.

> **Note**
>
> Custom drying and the **InkToolbar**
> If your app overrides the default ink rendering behavior of the **InkPresenter** with a custom drying implementation, the rendered ink strokes are no longer available to the InkToolbar and the built-in erase commands of the InkToolbar do not work as expected. To provide erase functionality, you must handle all pointer events, perform hit-testing on each stroke, and override the built-in "Erase all ink" command.

Other articles in this section

Topic	Description
Recognize ink strokes	Convert ink strokes to text using handwriting recognition, or to shapes using custom recognition.
Store and retrieve ink strokes	Store ink stroke data in a Graphics Interchange Format (GIF) file using embedded Ink Serialized Format (ISF) metadata.
Add an InkToolbar to a Windows inking app	Add a default InkToolbar to a Windows app inking app, add a custom pen button to the InkToolbar, and bind the custom pen button to a custom pen definition.

Related articles

- Get started: Support ink in your Windows app
- Handle pointer input
- Identify input devices
- Ink Serialized Format (ISF) Specification .

APIs

- **Windows.Devices.Input**
- **Windows.UI.Input.Inking**
- **Windows.UI.Input.Inking.Core**

Samples

- Get Started Tutorial: Support ink in your Windows app
- Simple ink sample (C#/C++)
- Complex ink sample (C++)
- Ink sample (JavaScript)
- Coloring book sample
- Family notes sample
- Basic input sample
- Low latency input sample
- User interaction mode sample
- Focus visuals sample

Archive Samples

- Input: Device capabilities sample
- Input: XAML user input events sample
- XAML scrolling, panning, and zooming sample
- Input: Gestures and manipulations with GestureRecognizer

Pen interactions and haptic (tactile) feedback

Article • 06/16/2022

Windows has long supported digital pens that let users interact with their devices in a natural, direct fashion and to express their creativity through rich writing and drawing experiences using digital ink.

With Windows 11, a new capability is being introduced that makes the digital pen experience even more natural and compelling: When using a pen that supports "haptic feedback", users can actually feel their pen interacting in a tactile manner with the user interface (UI) of an app.

> **Note**
>
> When referring to this new feature, "haptic" is used throughout the developer APIs and related documentation, while "tactile" is the friendly name presented to users for setting feedback preferences in Windows Settings.

Haptic feedback experiences supported in Windows 11 include *inking feedback* and *interaction feedback*:

- Inking feedback simulates the feel of various types of writing or drawing tools (such as pen, marker, pencil, highlighter, and so on) through continuous vibrations while the pen is in contact with the screen. By default, the Windows Ink Platform supports haptic feedback for all drawing tools (this topic covers how to provide a custom inking solution beyond the one supported by Windows Ink).
- Interaction feedback, on the other hand, is direct feedback based on key user actions such as hovering over or clicking a button, responding to the completion of an action, or to draw the user's attention.

Typically, five steps are required to fully support haptic feedback:

- Detect pen input.
- Determine if the current pen and device support haptic feedback and, if so, what haptic feedback features it supports.
- Decide on the haptic feedback signal to send.
- Send the haptic feedback.
- Stop the haptic feedback

Detect pen input

To detect and isolate pen input, you must first register for the PointerEntered event and then check whether the PointerDeviceType is a pen.

The following code shows how to check the pointer device type within a PointerEntered event. For this example, if the input is not from a pen we simply return from the event handler. Otherwise, we check the pen capabilities and configure the haptic feedback.

C#

```
private void InputObserver_PointerEntered(object sender,
PointerRoutedEventArgs e)
{
    ...

    // If the current Pointer device is not a pen, exit.
    if (e.Pointer.PointerDeviceType != PointerDeviceType.Pen)
    {
       return;
    }

    ...
}
```

Determine support for haptic feedback

Not all pens and digitizers support haptic feedback, and the pens that do won't necessarily support all haptic feedback features described in this topic. As such, it is important to programmatically confirm which features are supported by the active pen.

In a continuation of the preceding example, we show how to check whether the active pen supports haptic feedback.

We first attempt to retrieve a PenDevice object from the current PointerId. If a PenDevice cannot be obtained, we simply return from the event handler.

If a PenDevice was obtained, we test if it supports a SimpleHapticsController property. If not, we again simply return from the event handler.

C#

```
// Attempt to retrieve the PenDevice from the current PointerId.
penDevice = PenDevice.GetFromPointerId(e.Pointer.PointerId);
```

```
// If a PenDevice cannot be retrieved based on the PointerId, it does not su
pport
// advanced pen features, such as haptic feedback.
if (penDevice == null)
{
    return;
}

// Check to see if the current PenDevice supports haptic
feedback by seeing if it
// has a SimpleHapticsController.
hapticsController = penDevice.SimpleHapticsController;
if (hapticsController == null)
{
    return;
}
```

The SimpleHapticsController retrieved in the preceding example is used in subsequent examples to query haptic capabilities and to send/stop haptic feedback.

> **Note**
>
> If you are building apps with the **Windows App SDK Preview 1.0**, you can use **PenDevice interop** (**PenDeviceInterop.FromPointerPoint(PointerPoint)**) to access the system **PenDevice**.

C#

```
private void InputObserver_PointerEntered(PointerInputObserver sender,
PointerEventArgs args)
{
    var penDevice =
PenDeviceInterop.PenDeviceFromPointerPoint(args.CurrentPoint);
}
```

The following sections describe the feedback features that haptic pens must support, as well as those that are optional. A required haptic feedback type can typically be used as a fallback instead of an optional feature.

Inking waveforms

Inking waveforms play continuously while the pen is in contact with the screen, and attempt to simulate the feel of various writing or drawing tools.

Feature	Description	Required / Optional
InkContinous waveform	Simulates the feel of inking with a physical ball point pen. This is the default fallback when an inking waveform is not supported by a haptic pen.	Required
BrushContinuous waveform	Continuous haptic signal when user selects brush as inking tool.	Optional
ChiselMarkerContinuous waveform	Continuous haptic signal when user selects chisel marker/highlighter as inking tool.	Optional
EraserContinuous waveform	Continuous haptic signal when user selects eraser as inking tool.	Optional
GalaxyContinuous waveform (the HID documentation and implementation guide refers to this waveform as *SparkleContinuous*)	Continuous haptic signal for special ink tools, such as a multi-colored brush.	Optional
MarkerContinuous waveform	Continuous haptic signal when user selects marker as inking tool.	Optional
PencilContinuous waveform	Continuous haptic signal when user selects pencil as inking tool.	Optional

Interaction waveforms

Interaction waveforms are typically short (exceptions noted in the following table), direct feedback waveforms generated on demand to confirm key actions such as hovering over or clicking a button, responding to the completion of an action, or to draw the user's attention.

Feature	Description	Required / Optional
Click waveform	A short "click" feedback. This is the default fallback when an interaction waveform selected by the app is not supported by a haptic pen.	Required
Error waveform	A strong signal to alert the user that an action failed, or an error has occurred.	Optional

Feature	Description	Required / Optional
Hover waveform	Indicates that the user has started hovering over an interactive UI element.	Optional
Press waveform	Indicates when a user presses an interactive UI element in an incremental action (see Release).	Optional
Release waveform	Indicates when a user releases an interactive UI element in an incremental action (see Press).	Optional
Success waveform	Strong signal to alert the user that an action succeeded.	Optional
BuzzContinuous waveform	Continuous buzzing sensation.	Optional
RumbleContinuous waveform	Continuous rumbling sensation.	Optional

Haptic feedback customizations

Some haptic pens can support the following customizations.

Feature	Description	Required / Optional
Intensity	Sets the intensity of the haptic signal.	Optional
Play Count	Repeats a haptic signal a specified number of times.	Optional
Replay Pause Interval	Sets the time between each repeated playing of the haptic signal.	Optional
Play Duration	Sets the interval of time that a haptic signal is played.	Optional

Check for custom settings support

To check for Intensity, Play Count, Replay Pause Interval, and Play Duration support, use the following properties of the SimpleHapticsController:

- IsIntensitySupported
- IsPlayCountSupported
- IsPlayDurationSupported
- IsReplayPauseIntervalSupported

Send and stop inking haptic feedback

Use the SendHapticFeedback method of the SimpleHapticsController object to pass inking waveforms to the user's pen. This method supports passing in either a waveform or both a waveform with a customized intensity value (see Customize haptic feedback).

Call SendHapticFeedback and pass in an inking waveform to configure the pen to start playing that waveform as soon as the tip of the pen touches anywhere on the screen. The waveform will continue playing until the pen is lifted or StopFeedback is called, whichever happens first. We recommend doing this in the PointerEntered event handler for the element in which you want haptics to be played. For example, an app with a custom inking implementation would do this in the PointerEntered method of its inking canvas.

To retrieve the desired inking waveform, you must iterate through the SupportedFeedback collection of the SimpleHapticsController, ensuring it is supported by the active pen.

If it is not supported, you can either choose to not play anything at all or fall back to the InkContinuous waveform, as that is guaranteed to be supported.

In the following example, we attempt to send the BrushContinuous waveform (but fall back to InkContinuous if BrushContinuous is not supported).

C#

```
SimpleHapticsControllerFeedback currentWaveform;

// Attempt to set the currentWaveform to BrushContinuous.
foreach (var waveform in hapticsController.SupportedFeedback)
{
    if (waveform.Waveform == KnownSimpleHapticsControllerWaveforms.BrushCont
inuous)
    {
        currentWaveform = waveform;
    }
}

// If currentWaveform is null, it was not in the SupportedFeedback collectio
n, so instead set
// the waveform to InkContinuous.
if (currentWaveform == null)
{
    foreach (var waveform in hapticsController.SupportedFeedback)
    {
        if (waveform.Waveform == KnownSimpleHapticsControllerWaveforms.InkCo
ntinuous)
        {
```

```
            currentWaveform = waveform;
        }
    }
}

// Send the currentWaveform
hapticsController.SendHapticFeedback(currentWaveform);
```

It is important that you also stop haptic feedback when the associated pointer exits the element you registered for haptic feedback. Otherwise, the waveform will continue attempting to play on the active pen.

> **Note**
>
> Some pens may optionally stop haptics on their own when the pen leaves the range of the screen. However, it is not required for all pens to do this, so applications should always explicitly stop haptic feedback as described here.

To stop haptic feedback on an element, register for the PointerExited event on same element as you registered the PointerEntered handler that sent the haptic signal. In that exited event handler, call StopFeedback as shown here.

C#

```
hapticsController.StopFeedback();
```

Send and stop interaction feedback

Sending Interaction feedback is quite similar to sending inking feedback.

Use the SendHapticFeedback method of the SimpleHapticsController object to pass interaction waveforms to the user's pen. This method supports passing in either a waveform or both a waveform with a customized intensity value (see Customize haptic feedback).

Call SendHapticFeedback and pass in an inking waveform to configure the pen to start playing that waveform immediately based on some interaction within your app (instead of when the tip of the pen touches the screen for inking feedback).

When using any of the non-continuous Interaction waveforms, it is not necessary to make a corresponding StopFeedback call. You do still need to call StopFeedback for the continuous Interaction waveforms.

> **Note**
>
> Sending an interaction waveform when an inking waveform is being played will temporarily interrupt the inking waveform. The inking waveform will resume when the interaction waveform stops.

To retrieve the desired interaction waveform, you must iterate through the SupportedFeedback collection of the SimpleHapticsController, ensuring it is supported by the active pen.

If it is not supported, you can either choose to not play anything at all or fall back to the Click waveform, as that is guaranteed to be supported.

In the following example, we attempt to send the Error waveform (but fall back to Click if Error is not supported).

C#

```
SimpleHapticsControllerFeedback currentWaveform;

// Attempt to set the currentWaveform to BrushContinuous.
foreach (var waveform in hapticsController.SupportedFeedback)
{
    if (waveform.Waveform == KnownSimpleHapticsControllerWaveforms.Error)
    {
        currentWaveform = waveform;
    }
}

// If currentWaveform is null, it was not in the SupportedFeedback collectio
n, so instead set
// the waveform to Click.
if (currentWaveform == null)
{
    foreach (var waveform in hapticsController.SupportedFeedback)
    {
        if (waveform.Waveform == KnownSimpleHapticsControllerWaveforms.Click
)
        {
            currentWaveform = waveform;
        }
    }
}

// Send the currentWaveform.
hapticsController.SendHapticFeedback(currentWaveform);
```

Customize haptic feedback

There are three ways to customize haptic feedback. The first is supported by both Inking and Interaction feedback, while the second and third are only supported by Interaction feedback.

1. Adjust the intensity of the feedback relative to the maximum system intensity setting. To do this, you must first check to ensure that the SimpleHapticsController supports setting the intensity and then call SendHapticFeedback with the desired `Intensity` value.

```
C#
```

```
if (hapticsController.IsIntensitySupported)
{
    foreach (var waveform in hapticsController.SupportedFeedback)
    {
        if (waveform.Waveform == KnownSimpleHapticsControllerWaveforms.
Click)
        {
            double intensity = 0.75;
            hapticsController.SendHapticFeedback(waveform, intensity);
        }
    }
}
```

2. Repeat the haptic signal a specified number of times. To do this, you must first check to ensure that the SimpleHapticsController supports setting the intensity and then call SendHapticFeedbackForPlayCount with the desired count value. You can also set both the intensity and the replay pause interval.

> **Note**
>
> If the SimpleHapticsController does not support setting the intensity or the replay pause interval, the supplied values will be ignored.

```
C#
```

```
if (hapticsController.IsPlayCountSupported && hapticsController.IsInten
sitySupported && hapticsController.IsReplayPauseIntervalSupported)
{
    foreach (var waveform in hapticsController.SupportedFeedback)
    {
        if (waveform.Waveform == KnownSimpleHapticsControllerWaveforms.
Click)
        {
```

```
            double intensity = 0.75;
            int playCount = 3;
            System.TimeSpan pauseDuration = new System.TimeSpan(1000000
);
            hapticsController.SendHapticFeedbackForPlayCount(currentWav
eform, intensity, playCount, pauseDuration);
        }
    }
}
```

3. Set the duration of the haptic signal. To do this, you must first check to ensure that the SimpleHapticsController supports setting the play duration and then call SendHapticFeedbackForDuration with the desired time interval value. You can also set the intensity.

> **Note**
>
> If the SimpleHapticsController does not support setting the intensity, the supplied value will be ignored.

C#

```
if (hapticsController.IsPlayDurationSupported && hapticsController.IsIn
tensitySupported)
{
    foreach (var waveform in hapticsController.SupportedFeedback)
    {
        if (waveform.Waveform == KnownSimpleHapticsControllerWaveforms.
RumbleContinuous)
        {
            double intensity = 0.75;
            System.TimeSpan playDuration = new System.TimeSpan(5000000)
;
            hapticsController.SendHapticFeedbackForDuration(currentWave
form, intensity, playDuration);
        }
    }
}
```

Examples

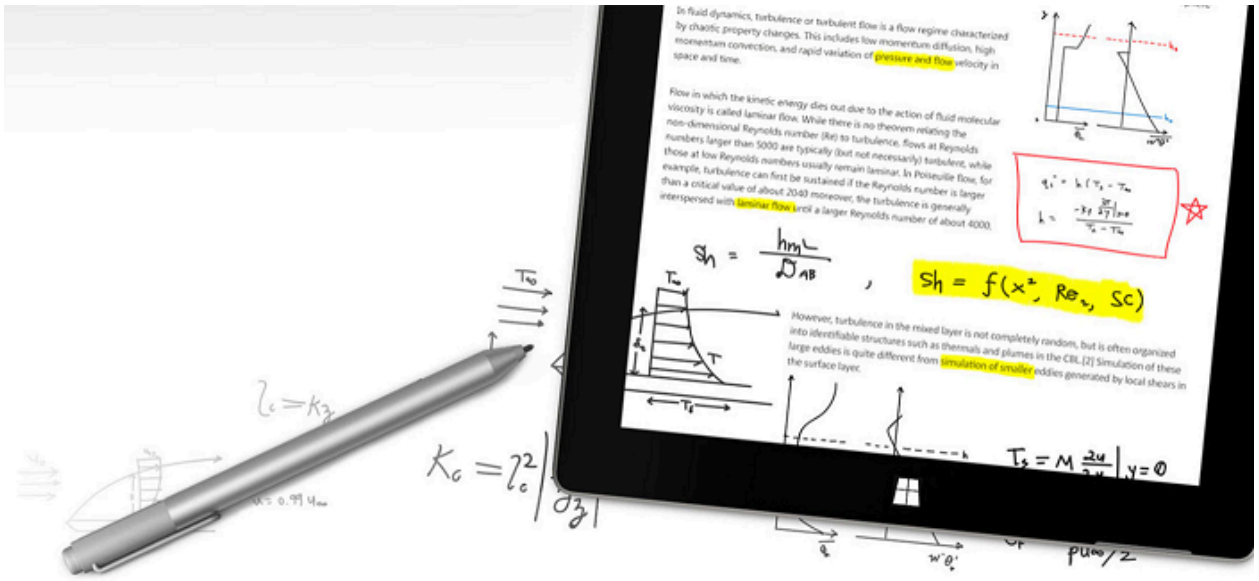
See the Pen haptics sample for working examples of the following functionality:

- Get a SimpleHapticsController from pen input: Go from PointerId to PenDevice to SimpleHapticsController (requires both a haptic-capable pen and a device that supports the pen).

- Check pen haptics capabilities: A SimpleHapticsController exposes properties for pen hardware capabilities, including IsIntensitySupported, IsPlayCountSupported, SupportedFeedback, and so on.
- Start and stop haptic feedback: Use the SendHapticFeedback and StopFeedback methods appropriately.
- Trigger haptic feedback: Feedback for both *inking feedback* and *interaction feedback*.

Tutorial: Support ink in your Windows app

Article • 02/27/2025



Surface Pen (available for purchase at the Microsoft Store).

This tutorial steps through how to create a basic Windows app that supports writing and drawing with Windows Ink. We use snippets from a sample app, which you can download from GitHub (see Sample code), to demonstrate the various features and associated Windows Ink APIs (see Components of the Windows Ink platform) discussed in each step.

We focus on the following:

- Adding basic ink support
- Adding an ink toolbar
- Supporting handwriting recognition
- Supporting basic shape recognition
- Saving and loading ink

For more detail about implementing these features, see Pen interactions and Windows Ink in Windows apps.

Introduction

With Windows Ink, you can provide your customers with the digital equivalent of almost any pen-and-paper experience imaginable, from quick, handwritten notes and

annotations to whiteboard demos, and from architectural and engineering drawings to personal masterpieces.

Prerequisites

- A computer (or a virtual machine) running the current version of Windows 10 or Windows 11
- Visual Studio 2019 and the RS2 SDK or later
- Windows 10 SDK (10.0.15063.0)
- Depending on your configuration, you might have to install the Microsoft.NETCore.UniversalWindowsPlatform NuGet package and enable **Developer mode** in your system settings (Settings -> Update & Security -> For developers -> Use developer features).
- If you're new to Windows app development with Visual Studio, have a look through these topics before you start this tutorial:
  - Install tools for the Windows App SDK
  - Create a "Hello, world" app (XAML)
- **[OPTIONAL]** A digital pen and a computer with a display that supports input from that digital pen.

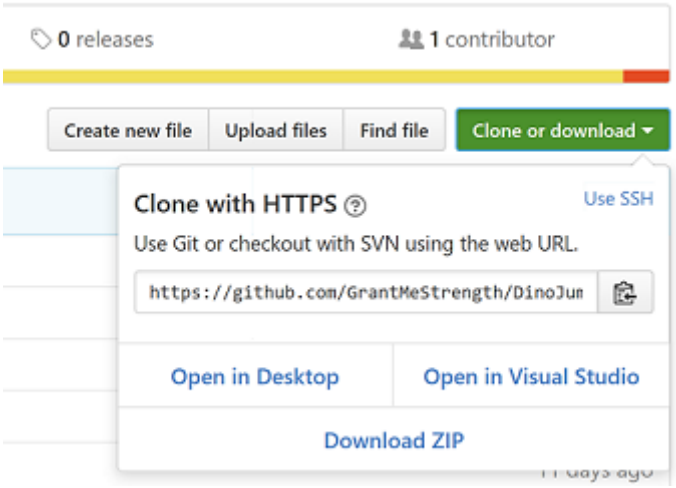
> **Note**
>
> While Windows Ink can support drawing with a mouse and touch (we show how to do this in Step 3 of this tutorial) for an optimal Windows Ink experience, we recommend that you have a digital pen and a computer with a display that supports input from that digital pen.

Sample code

Throughout this tutorial, we use a sample ink app to demonstrate the concepts and functionality discussed.

Download this Visual Studio sample and source code from GitHub at windows-appsample-get-started-ink sample:

1. Select the green **Clone or download** button



2. If you have a GitHub account, you can clone the repo to your local machine by choosing **Open in Visual Studio**
3. If you don't have a GitHub account, or you just want a local copy of the project, choose **Download ZIP** (you'll have to check back regularly to download the latest updates)

> **ⓘ Important**
>
> Most of the code in the sample is commented out. As we go through each step, you'll be asked to uncomment various sections of the code. In Visual Studio, just highlight the lines of code, and press CTRL-K and then CTRL-U.

Components of the Windows Ink platform

These objects provide the bulk of the inking experience for Windows apps.

Expand table

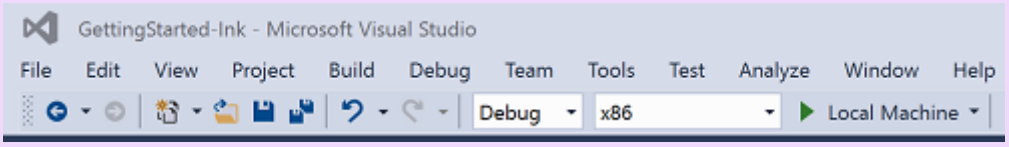
Component	Description
InkCanvas	A XAML UI platform control that, by default, receives and displays all input from a pen as either an ink stroke or an erase stroke.
InkPresenter	A code-behind object, instantiated along with an InkCanvas control (exposed through the InkCanvas.InkPresenter property). This object provides all default inking functionality exposed by the InkCanvas, along with a comprehensive set of APIs for additional customization and personalization.
InkToolbar	A XAML UI platform control containing a customizable and extensible collection of buttons that activate ink-related features

Component	Description
	in an associated **InkCanvas**.
IInkD2DRenderer We do not cover this functionality here, for more information, see the Complex ink sample.	Enables the rendering of ink strokes onto the designated Direct2D device context of a Universal Windows app, instead of the default **InkCanvas** control.

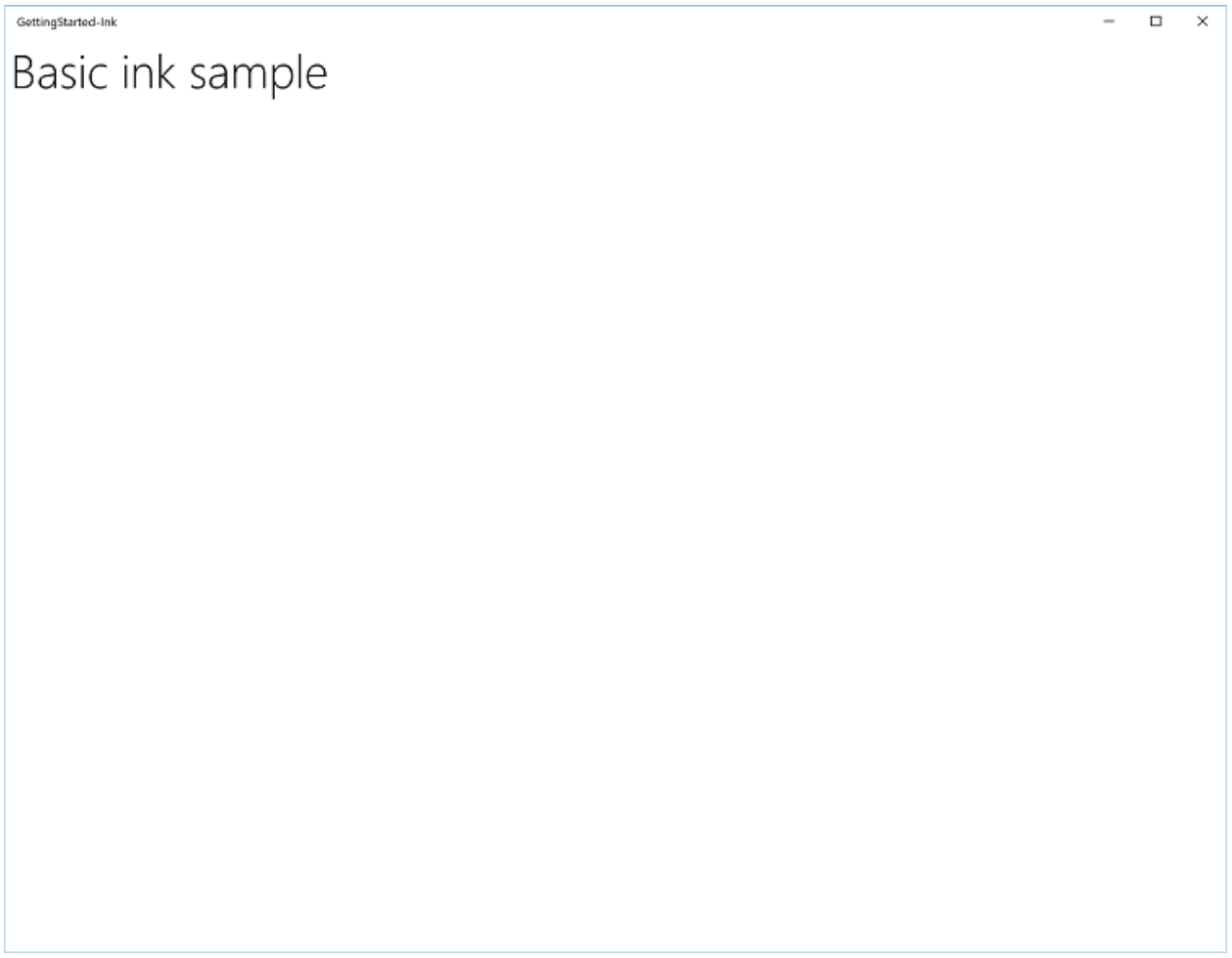
Step 1: Run the sample

After you've downloaded the RadialController sample app, verify that it runs:

1. Open the sample project in Visual Studio.
2. Set the **Solution Platforms** dropdown to a non-Arm selection.
3. Press F5 to compile, deploy, and run.

> **Note**
>
> Alternatively, you can select **Debug** > **Start debugging** menu item, or select the **Local Machine** Run button shown here.
>
> 
>

The app window opens, and after a splash screen appears for a few seconds, you'll see this initial screen.



Okay, we now have the basic Windows app that we'll use throughout the rest of this tutorial. In the following steps, we add our ink functionality.

Step 2: Use InkCanvas to support basic inking

Perhaps you've probably already noticed that the app, in it's initial form, doesn't let you draw anything with the pen (although you can use the pen as a standard pointer device to interact with the app).

Let's fix that little shortcoming in this step.

To add basic inking functionality, just place an **InkCanvas** control on the appropriate page in your app.

> ⓘ **Note**
>
> An InkCanvas has default **Height** and **Width** properties of zero, unless it is the child of an element that automatically sizes its child elements.

In the sample:

1. Open the MainPage.xaml.cs file.
2. Find the code marked with the title of this step ("// Step 2: Use InkCanvas to support basic inking").
3. Uncomment the following lines. (These references are required for the functionality used in the subsequent steps).

C#

```
using Windows.UI.Input.Inking;
using Windows.UI.Input.Inking.Analysis;
using Windows.UI.Xaml.Shapes;
using Windows.Storage.Streams;
```

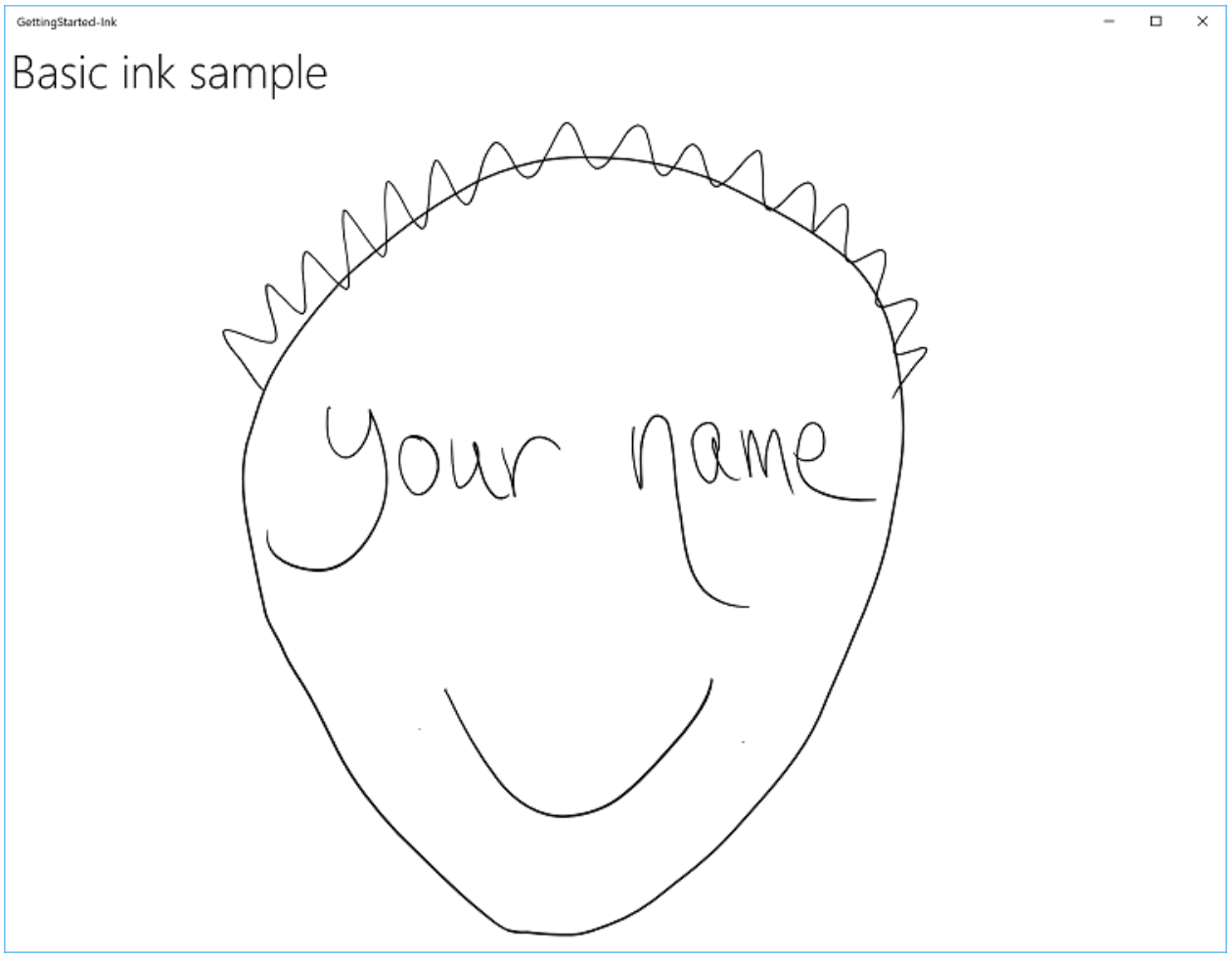
4. Open the MainPage.xaml file.
5. Find the code marked with the title of this step ("<!-- Step 2: Basic inking with InkCanvas -->").
6. Uncomment the following line.

XAML

```
<InkCanvas x:Name="inkCanvas" />
```

That's it!

Now run the app again. Go ahead and scribble, write your name, or (if you're holding a mirror or have a very good memory) draw your self-portrait.



Step 3: Support inking with touch and mouse

You'll notice that, by default, ink is supported for pen input only. If you try to write or draw with your finger, your mouse, or your touchpad, you'll be disappointed.

To turn that frown upside down , you need to add a second line of code. This time it's in the code-behind for the XAML file in which you declared your **InkCanvas**.

In this step, we introduce the **InkPresenter** object, which provides finer-grained management of the input, processing, and rendering of ink input (standard and modified) on your **InkCanvas**.

> ⓘ **Note**
>
> Standard ink input (pen tip or eraser tip/button) is not modified with a secondary hardware affordance, such as a pen barrel button, right mouse button, or similar mechanism.

To enable mouse and touch inking, set the **InputDeviceTypes** property of the **InkPresenter** to the combination of **CoreInputDeviceTypes** values that you want.

In the sample:

1. Open the MainPage.xaml.cs file.
2. Find the code marked with the title of this step ("// Step 3: Support inking with touch and mouse").
3. Uncomment the following lines.

```
C#
```

```
    inkCanvas.InkPresenter.InputDeviceTypes =
        Windows.UI.Core.CoreInputDeviceTypes.Mouse |
        Windows.UI.Core.CoreInputDeviceTypes.Touch |
        Windows.UI.Core.CoreInputDeviceTypes.Pen;
```

Run the app again and you'll find that all your finger-painting-on-a-computer-screen dreams have come true!

> **Note**
>
> When specifying input device types, you must indicate support for each specific input type (including pen), because setting this property overrides the default InkCanvas setting.

Step 4: Add an ink toolbar

The **InkToolbar** is a UWP platform control that provides a customizable and extensible collection of buttons for activating ink-related features.

By default, the **InkToolbar** includes a basic set of buttons that let users quickly select between a pen, a pencil, a highlighter, or an eraser, any of which can be used together with a stencil (ruler or protractor). The pen, pencil, and highlighter buttons each also provide a flyout for selecting ink color and stroke size.

To add a default **InkToolbar** to an inking app, just place it on the same page as your **InkCanvas** and associate the two controls.

In the sample

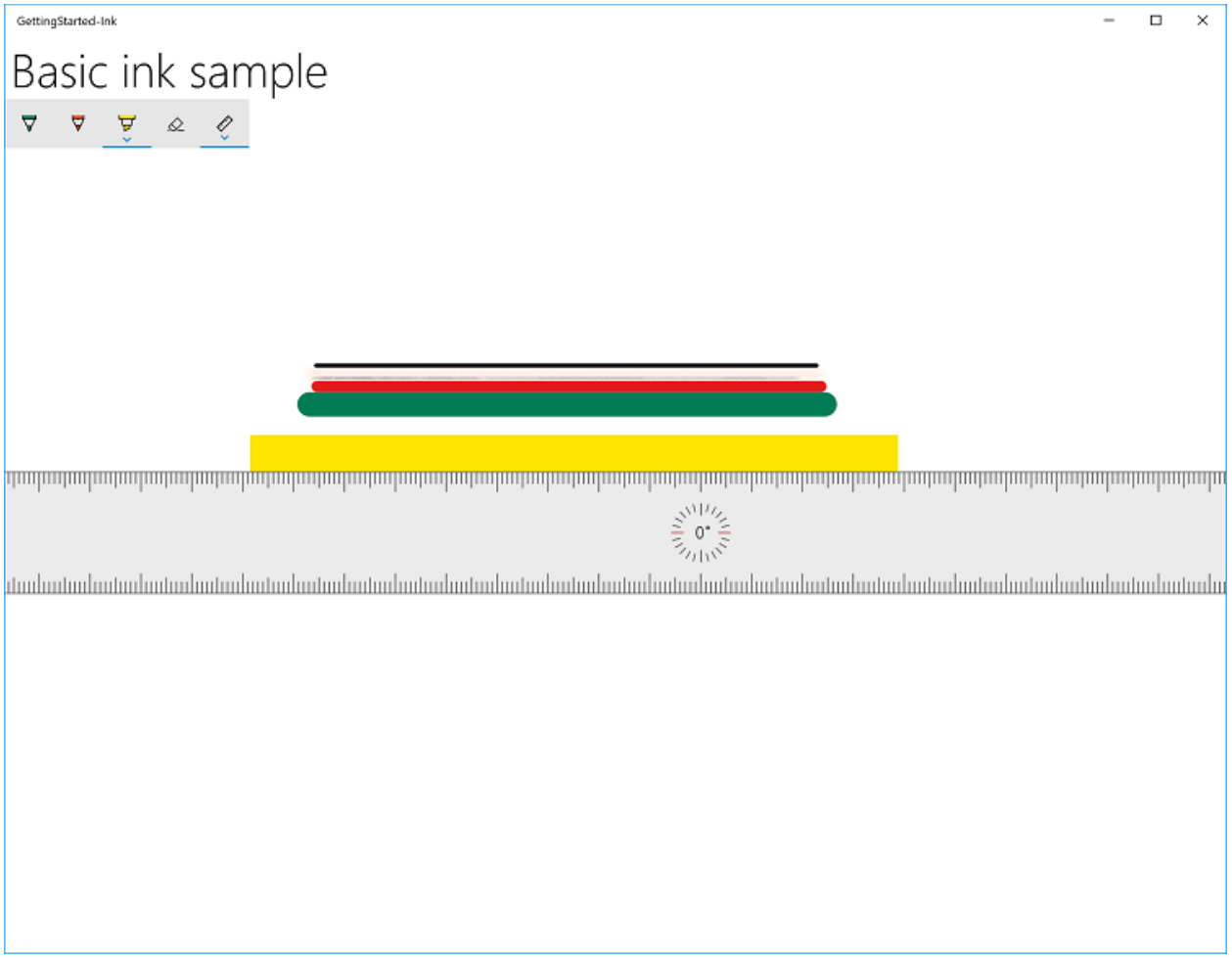
1. Open the MainPage.xaml file.
2. Find the code marked with the title of this step ("<!-- Step 4: Add an ink toolbar -->").
3. Uncomment the following lines.

XAML

```
    <InkToolbar x:Name="inkToolbar"
                        VerticalAlignment="Top"
                        Margin="10,0,10,0"
                        TargetInkCanvas="{x:Bind inkCanvas}">
    </InkToolbar>
```

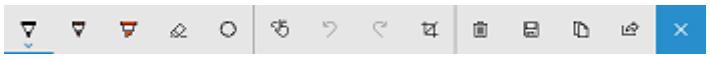
> **Note**
>
> To keep the UI and code as uncluttered and simple as possible, we use a basic Grid layout and declare the InkToolbar after the InkCanvas in a grid row. If you declare it before the InkCanvas, the InkToolbar is rendered first, below the canvas and inaccessible to the user.

Now run the app again to see the **InkToolbar** and try out some of the tools.



Challenge: Add a custom button

Here's an example of a custom **InkToolbar** (from Sketchpad in the Windows Ink Workspace).

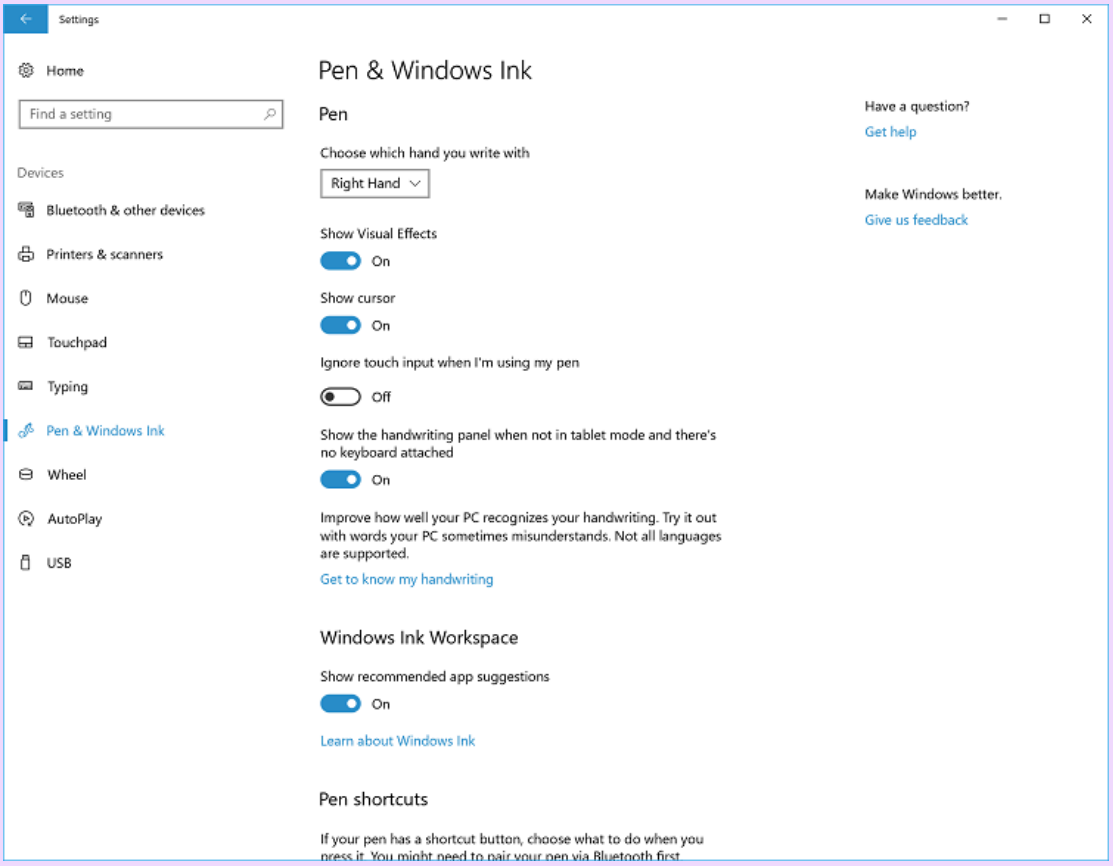


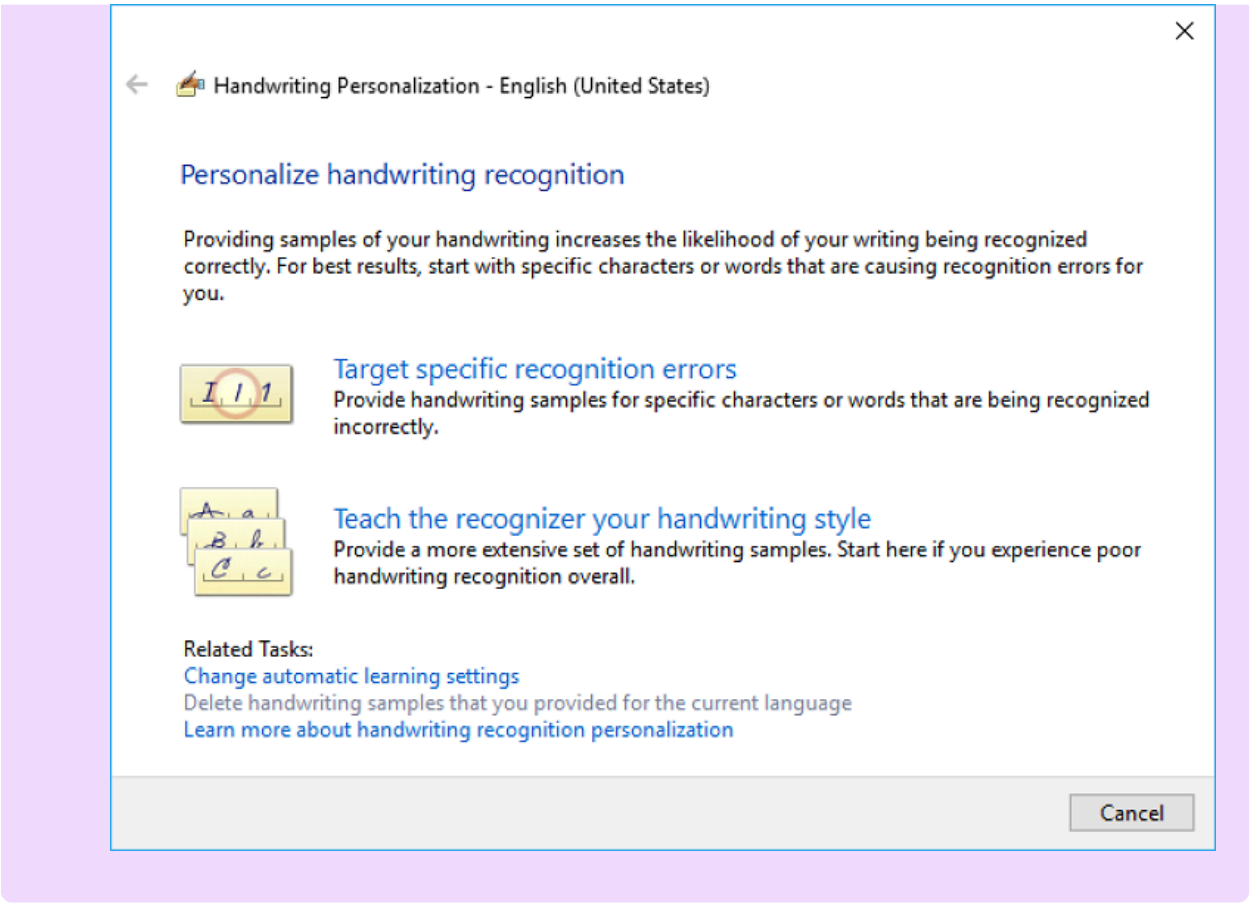
For more details about customizing an InkToolbar, see Add an InkToolbar to a Windows app inking app.

Step 5: Support handwriting recognition

Now that you can write and draw in your app, let's try to do something useful with those scribbles.

In this step, we use the handwriting recognition features of Windows Ink to try to decipher what you've written.

> **Note**
>
> Handwriting recognition can be improved through the **Pen & Windows Ink** settings:
>
> 1. Open the Start menu and select **Settings**.
> 2. From the Settings screen select **Devices** > **Pen & Windows Ink**.
>
> 
>
>
> 3. Select **Get to know my handwriting** to open the **Handwriting Personalization** dialog.



In the sample:

1. Open the MainPage.xaml file.
2. Find the code marked with the title of this step ("<!-- Step 5: Support handwriting recognition -->").
3. Uncomment the following lines.

XAML

```
<Button x:Name="recognizeText"
        Content="Recognize text"
        Grid.Row="0" Grid.Column="0"
        Margin="10,10,10,10"
        Click="recognizeText_ClickAsync"/>
<TextBlock x:Name="recognitionResult"
           Text="Recognition results: "
           VerticalAlignment="Center"
           Grid.Row="0" Grid.Column="1"
           Margin="50,0,0,0" />
```

4. Open the MainPage.xaml.cs file.
5. Find the code marked with the title of this step (" Step 5: Support handwriting recognition").
6. Uncomment the following lines.

- These are the global variables required for this step.

C#

```
    InkAnalyzer analyzerText = new InkAnalyzer();
    IReadOnlyList<InkStroke> strokesText = null;
    InkAnalysisResult resultText = null;
    IReadOnlyList<IInkAnalysisNode> words = null;
```

- This is the handler for the **Recognize text** button, where we do the recognition processing.

C#

```
    private async void recognizeText_ClickAsync(object sender,
RoutedEventArgs e)
    {
        strokesText = inkCanvas.InkPresenter.StrokeContainer.GetStrokes();
        // Ensure an ink stroke is present.
        if (strokesText.Count > 0)
        {
            analyzerText.AddDataForStrokes(strokesText);

            resultText = await analyzerText.AnalyzeAsync();

            if (resultText.Status == InkAnalysisStatus.Updated)
            {
                words =
analyzerText.AnalysisRoot.FindNodes(InkAnalysisNodeKind.InkWord);
                foreach (var word in words)
                {
                    InkAnalysisInkWord concreteWord =
(InkAnalysisInkWord)word;
                    foreach (string s in concreteWord.TextAlternates)
                    {
                        recognitionResult.Text += s;
                    }
                }
            }
            analyzerText.ClearDataForAllStrokes();
        }
    }
```

7. Run the app again, write something, and then click the **Recognize text** button
8. The results of the recognition are displayed beside the button

Challenge 1: International recognition

Windows Ink supports text recognition for many of the of the languages supported by Windows. Each language pack includes a handwriting recognition engine that can be installed with the language pack.

Target a specific language by querying the installed handwriting recognition engines.

For more details about international handwriting recognition, see Recognize Windows Ink strokes as text.

Challenge 2: Dynamic recognition

For this tutorial, we require that a button be pressed to initiate recognition. You can also perform dynamic recognition by using a basic timing function.

For more details about dynamic recognition, see Recognize Windows Ink strokes as text.

Step 6: Recognize shapes

Ok, so now you can convert your handwritten notes into something a little more legible. But what about those shaky, caffeinated doodles from your morning Flowcharters Anonymous meeting?

Using ink analysis, your app can also recognize a set of core shapes, including:

- Circle
- Diamond
- Drawing
- Ellipse
- EquilateralTriangle
- Hexagon
- IsoscelesTriangle
- Parallelogram
- Pentagon
- Quadrilateral
- Rectangle
- RightTriangle
- Square
- Trapezoid
- Triangle

In this step, we use the shape-recognition features of Windows Ink to try to clean up your doodles.

For this example, we don't attempt to redraw ink strokes (although that's possible). Instead, we add a standard canvas under the InkCanvas where we draw equivalent Ellipse or Polygon objects derived from the original ink. We then delete the corresponding ink strokes.

In the sample:

1. Open the MainPage.xaml file
2. Find the code marked with the title of this step ("<!-- Step 6: Recognize shapes -->")
3. Uncomment this line.

XAML

```
   <Canvas x:Name="canvas" />

   And these lines.

    <Button Grid.Row="1" x:Name="recognizeShape"
Click="recognizeShape_ClickAsync"
        Content="Recognize shape"
        Margin="10,10,10,10" />
```

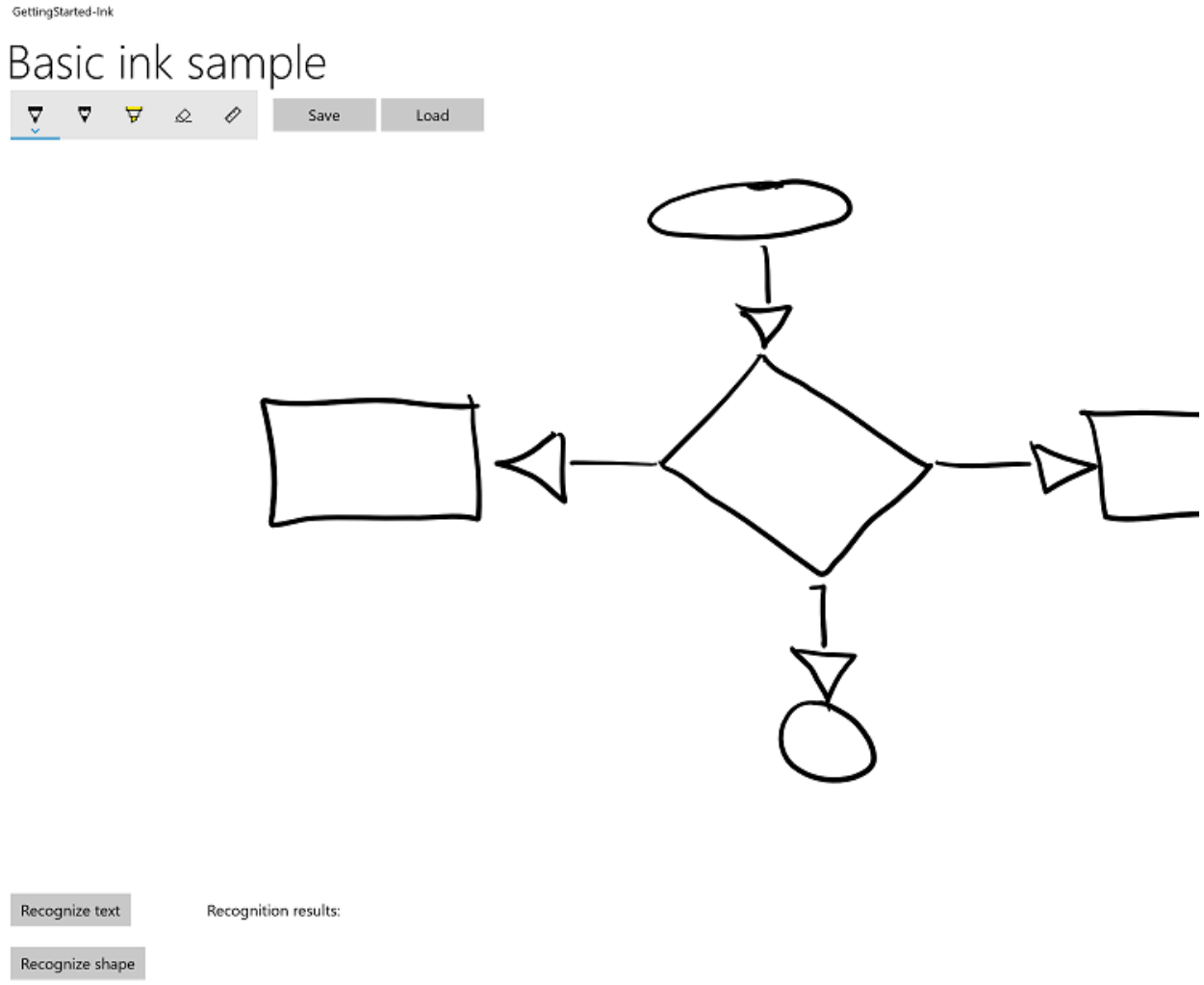
4. Open the MainPage.xaml.cs file
5. Find the code marked with the title of this step ("// Step 6: Recognize shapes")
6. Uncomment these lines:

C#

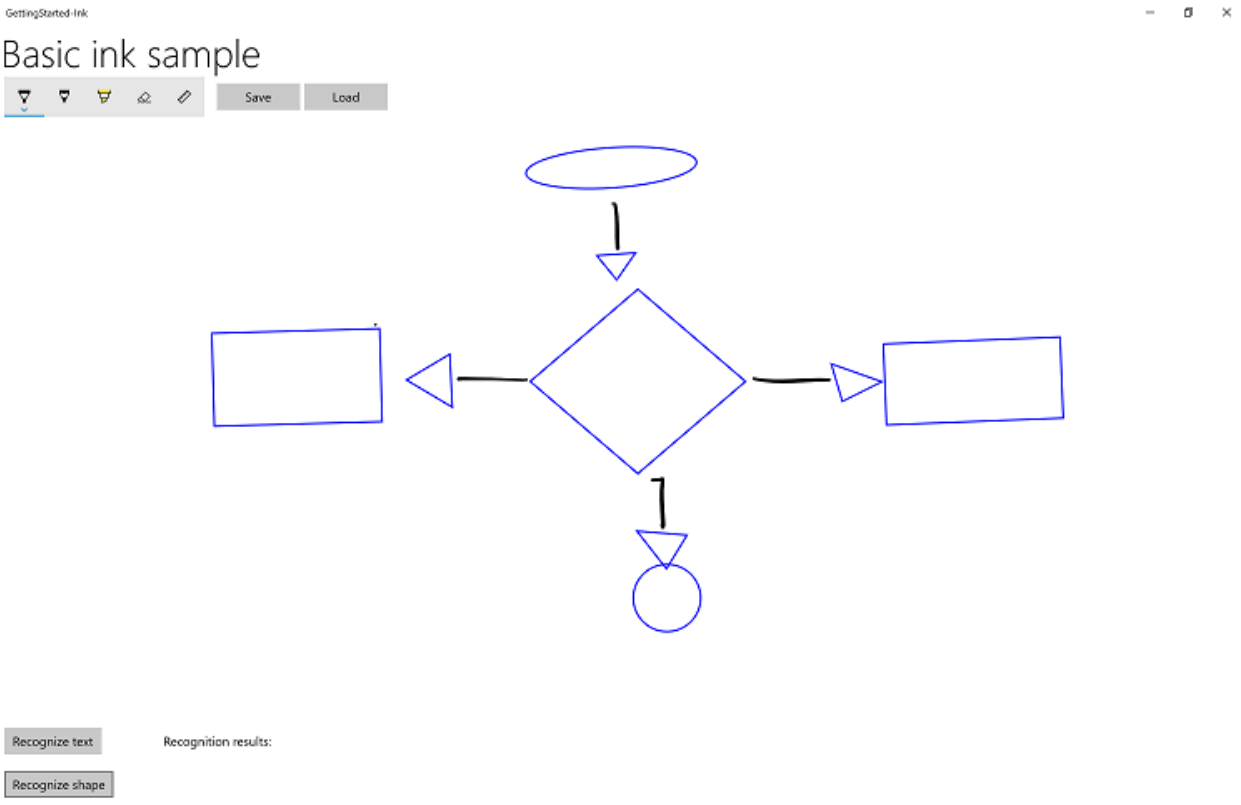
```
    private async void recognizeShape_ClickAsync(object sender,
RoutedEventArgs e)
    {
        ...
    }

    private void DrawEllipse(InkAnalysisInkDrawing shape)
    {
        ...
    }

    private void DrawPolygon(InkAnalysisInkDrawing shape)
    {
        ...
    }
```

7. Run the app, draw some shapes, and click the **Recognize shape** button

Here's an example of a rudimentary flowchart from a digital napkin.



Here's the same flowchart after shape recognition.



Step 7: Save and load ink

So, you're done doodling and you like what you see, but think you might like to tweak a couple of things later? You can save your ink strokes to an Ink Serialized Format (ISF) file and load them for editing whenever the inspiration strikes.

The ISF file is a basic GIF image that includes additional metadata describing ink-stroke properties and behaviors. Apps that are not ink enabled can ignore the extra metadata and still load the basic GIF image (including alpha-channel background transparency).

> **Note**
>
> The Ink Serialized Format (ISF) Specification can be downloaded from the Microsoft Download Center.

In this step, we hook up the **Save** and **Load** buttons located beside the ink toolbar.

In the sample:

1. Open the MainPage.xaml file.
2. Find the code marked with the title of this step ("<!-- Step 7: Saving and loading ink -->").
3. Uncomment the following lines.

XAML

```
    <Button x:Name="buttonSave"
            Content="Save"
            Click="buttonSave_ClickAsync"
            Width="100"
            Margin="5,0,0,0"/>
    <Button x:Name="buttonLoad"
            Content="Load"
            Click="buttonLoad_ClickAsync"
            Width="100"
            Margin="5,0,0,0"/>
```

4. Open the MainPage.xaml.cs file.
5. Find the code marked with the title of this step ("// Step 7: Save and load ink").
6. Uncomment the following lines.

C#

```
    private async void buttonSave_ClickAsync(object sender, RoutedEventArgs
e)
    {
        ...
    }

    private async void buttonLoad_ClickAsync(object sender, RoutedEventArgs
e)
    {
        ...
    }
```

7. Run the app and draw something.
8. Select the **Save** button and save your drawing.
9. Erase the ink or restart the app.
10. Select the **Load** button and open the ink file you just saved.

Challenge: Use the clipboard to copy and paste ink strokes

Windows ink also supports copying and pasting ink strokes to and from the clipboard.

For more details about using the clipboard with ink, see Store and retrieve Windows Ink stroke data.

Summary

Congratulations, you've completed the **Input: Support ink in your Windows app** tutorial ! We showed you the basic code required for supporting ink in your Windows apps, and how to provide some of the richer user experiences supported by the Windows Ink platform.

Related articles

- Pen interactions and Windows Ink in Windows apps

Samples

- Ink analysis sample (basic) (C#)
- Ink handwriting recognition sample (C#)
- Save and load ink strokes from an Ink Serialized Format (ISF) file
- Save and load ink strokes from the clipboard

- Ink toolbar location and orientation sample (basic)
- Ink toolbar location and orientation sample (dynamic)
- Simple ink sample (C#/C++)
- Complex ink sample (C++)
- Ink sample (JavaScript)
- Get Started Tutorial: Support ink in your Windows app
- Coloring book sample
- Family notes sample

Recognize Windows Ink strokes as text and shapes

Article • 06/24/2021

Convert ink strokes to text and shapes using the recognition capabilities built into Windows Ink.

> **Important APIs**: InkCanvas, Windows.UI.Input.Inking

Free-form recognition with ink analysis

Here, we demonstrate how to use the Windows Ink analysis engine (Windows.UI.Input.Inking.Analysis) to classify, analyze, and recognize a set of free-form strokes on an **InkCanvas** as either text or shapes. (In addition to text and shape recognition, ink analysis can also be used to recognize document structure, bullet lists, and generic drawings.)

> ⓘ **Note**
>
> For basic, single-line, plain text scenarios such as form input, see **Constrained handwriting recognition** later in this topic.

In this example, recognition is initiated when the user clicks a button to indicate they are finished drawing.

Download this sample from Ink analysis sample (basic)

1. First, we set up the UI (MainPage.xaml).

   The UI includes a "Recognize" button, an **InkCanvas**, and a standard **Canvas**. When the "Recognize" button is pressed, all ink strokes on the ink canvas are analyzed and (if recognized) corresponding shapes and text are drawn on the standard canvas. The original ink strokes are then deleted from the ink canvas.

   XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
     <Grid.RowDefinitions>
         <RowDefinition Height="Auto"/>
         <RowDefinition Height="*"/>
     </Grid.RowDefinitions>
     <StackPanel x:Name="HeaderPanel"
```

```
                Orientation="Horizontal"
                Grid.Row="0">
        <TextBlock x:Name="Header"
                   Text="Basic ink analysis sample"
                   Style="{ThemeResource HeaderTextBlockStyle}"
                   Margin="10,0,0,0" />
        <Button x:Name="recognize"
                Content="Recognize"
                Margin="50,0,10,0"/>
    </StackPanel>
    <Grid x:Name="drawingCanvas" Grid.Row="1">

        <!-- The canvas where we render the replacement text and
shapes. -->
        <Canvas x:Name="recognitionCanvas" />
        <!-- The canvas for ink input. -->
        <InkCanvas x:Name="inkCanvas" />

    </Grid>
</Grid>
```

2. In the UI code-behind file (MainPage.xaml.cs), add the namespace type references required for our ink and ink analysis functionality:

   - Windows.UI.Input.Inking
   - Windows.UI.Input.Inking.Analysis
   - Windows.UI.Xaml.Shapes

3. We then specify our global variables:

C#

```
InkAnalyzer inkAnalyzer = new InkAnalyzer();
IReadOnlyList<InkStroke> inkStrokes = null;
InkAnalysisResult inkAnalysisResults = null;
```

4. Next, we set some basic ink input behaviors:

   - The **InkPresenter** is configured to interpret input data from pen, mouse, and touch as ink strokes (**InputDeviceTypes**).
   - Ink strokes are rendered on the **InkCanvas** using the specified **InkDrawingAttributes**.
   - A listener for the click event on the "Recognize" button is also declared.

C#

```
/// <summary>
/// Initialize the UI page.
/// </summary>
```

```
public MainPage()
{
    this.InitializeComponent();

    // Set supported inking device types.
    inkCanvas.InkPresenter.InputDeviceTypes =
        Windows.UI.Core.CoreInputDeviceTypes.Mouse |
        Windows.UI.Core.CoreInputDeviceTypes.Pen |
        Windows.UI.Core.CoreInputDeviceTypes.Touch;

    // Set initial ink stroke attributes.
    InkDrawingAttributes drawingAttributes = new
InkDrawingAttributes();
    drawingAttributes.Color = Windows.UI.Colors.Black;
    drawingAttributes.IgnorePressure = false;
    drawingAttributes.FitToCurve = true;

inkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(drawingAttributes
);

    // Listen for button click to initiate recognition.
    recognize.Click += RecognizeStrokes_Click;
}
```

5. For this example, we perform the ink analysis in the click event handler of the "Recognize" button.

    - First, call **GetStrokes** on the **StrokeContainer** of the **InkCanvas.InkPresenter** to get the collection of all current ink strokes.
    - If ink strokes are present, pass them in a call to **AddDataForStrokes** of the InkAnalyzer.
    - We're trying to recognize both drawings and text, but you can use the **SetStrokeDataKind** method to specify whether you're interested only in text (including document structure and bullet lists) or only in drawings (including shape recognition).
    - Call **AnalyzeAsync** to initiate ink analysis and get the **InkAnalysisResult**.
    - If **Status** returns a state of **Updated**, call **FindNodes** for both **InkAnalysisNodeKind.InkWord** and **InkAnalysisNodeKind.InkDrawing**.
    - Iterate through both sets of node types and draw the respective text or shape on the recognition canvas (below the ink canvas).
    - Finally, delete the recognized nodes from the InkAnalyzer and the corresponding ink strokes from the ink canvas.

C#

```
/// <summary>
/// The "Analyze" button click handler.
/// Ink recognition is performed here.
```

```
/// </summary>
/// <param name="sender">Source of the click event</param>
/// <param name="e">Event args for the button click routed
event</param>
private async void RecognizeStrokes_Click(object sender,
RoutedEventArgs e)
{
    inkStrokes = inkCanvas.InkPresenter.StrokeContainer.GetStrokes();
    // Ensure an ink stroke is present.
    if (inkStrokes.Count > 0)
    {
        inkAnalyzer.AddDataForStrokes(inkStrokes);

        // In this example, we try to recognizing both
        // writing and drawing, so the platform default
        // of "InkAnalysisStrokeKind.Auto" is used.
        // If you're only interested in a specific type of recognition,
        // such as writing or drawing, you can constrain recognition
        // using the SetStrokDataKind method as follows:
        // foreach (var stroke in strokesText)
        // {
        //     analyzerText.SetStrokeDataKind(
        //      stroke.Id, InkAnalysisStrokeKind.Writing);
        // }
        // This can improve both efficiency and recognition results.
        inkAnalysisResults = await inkAnalyzer.AnalyzeAsync();

        // Have ink strokes on the canvas changed?
        if (inkAnalysisResults.Status == InkAnalysisStatus.Updated)
        {
            // Find all strokes that are recognized as handwriting and
            // create a corresponding ink analysis InkWord node.
            var inkwordNodes =
                inkAnalyzer.AnalysisRoot.FindNodes(
                    InkAnalysisNodeKind.InkWord);

            // Iterate through each InkWord node.
            // Draw primary recognized text on recognitionCanvas
            // (for this example, we ignore alternatives), and delete
            // ink analysis data and recognized strokes.
            foreach (InkAnalysisInkWord node in inkwordNodes)
            {
                // Draw a TextBlock object on the recognitionCanvas.
                DrawText(node.RecognizedText, node.BoundingRect);

                foreach (var strokeId in node.GetStrokeIds())
                {
                    var stroke =

inkCanvas.InkPresenter.StrokeContainer.GetStrokeById(strokeId);
                    stroke.Selected = true;
                }
                inkAnalyzer.RemoveDataForStrokes(node.GetStrokeIds());
            }
            inkCanvas.InkPresenter.StrokeContainer.DeleteSelected();
```

```
            // Find all strokes that are recognized as a drawing and
            // create a corresponding ink analysis InkDrawing node.
            var inkdrawingNodes =
                inkAnalyzer.AnalysisRoot.FindNodes(
                    InkAnalysisNodeKind.InkDrawing);
            // Iterate through each InkDrawing node.
            // Draw recognized shapes on recognitionCanvas and
            // delete ink analysis data and recognized strokes.
            foreach (InkAnalysisInkDrawing node in inkdrawingNodes)
            {
                if (node.DrawingKind == InkAnalysisDrawingKind.Drawing)
                {
                    // Catch and process unsupported shapes (lines and
so on) here.
                }
                // Process generalized shapes here (ellipses and
polygons).
                else
                {
                    // Draw an Ellipse object on the recognitionCanvas
(circle is a specialized ellipse).
                    if (node.DrawingKind ==
InkAnalysisDrawingKind.Circle || node.DrawingKind ==
InkAnalysisDrawingKind.Ellipse)
                    {
                        DrawEllipse(node);
                    }
                    // Draw a Polygon object on the recognitionCanvas.
                    else
                    {
                        DrawPolygon(node);
                    }
                    foreach (var strokeId in node.GetStrokeIds())
                    {
                        var stroke =
inkCanvas.InkPresenter.StrokeContainer.GetStrokeById(strokeId);
                        stroke.Selected = true;
                    }
                }
                inkAnalyzer.RemoveDataForStrokes(node.GetStrokeIds());
            }
            inkCanvas.InkPresenter.StrokeContainer.DeleteSelected();
        }
    }
}
```

6. Here's the function for drawing a TextBlock on our recognition canvas. We use the bounding rectangle of the associated ink stroke on the ink canvas to set the position and font size of the TextBlock.

```
C#
```

```
 /// <summary>
 /// Draw ink recognition text string on the recognitionCanvas.
 /// </summary>
 /// <param name="recognizedText">The string returned by text
recognition.</param>
 /// <param name="boundingRect">The bounding rect of the original ink
writing.</param>
 private void DrawText(string recognizedText, Rect boundingRect)
 {
     TextBlock text = new TextBlock();
     Canvas.SetTop(text, boundingRect.Top);
     Canvas.SetLeft(text, boundingRect.Left);

     text.Text = recognizedText;
     text.FontSize = boundingRect.Height;

     recognitionCanvas.Children.Add(text);
 }
```

7. Here are the functions for drawing ellipses and polygons on our recognition canvas. We use the bounding rectangle of the associated ink stroke on the ink canvas to set the position and font size of the shapes.

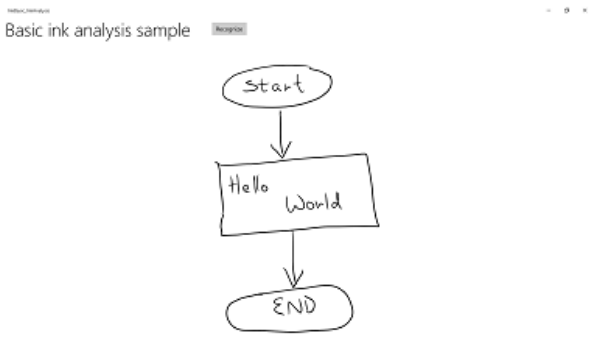
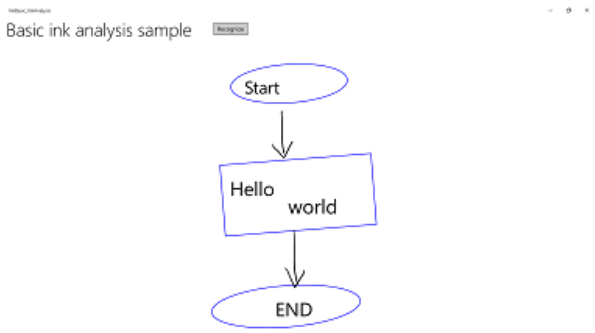
C#

```
 // Draw an ellipse on the recognitionCanvas.
 private void DrawEllipse(InkAnalysisInkDrawing shape)
 {
     var points = shape.Points;
     Ellipse ellipse = new Ellipse();

     ellipse.Width = shape.BoundingRect.Width;
     ellipse.Height = shape.BoundingRect.Height;

     Canvas.SetTop(ellipse, shape.BoundingRect.Top);
     Canvas.SetLeft(ellipse, shape.BoundingRect.Left);

     var brush = new
SolidColorBrush(Windows.UI.ColorHelper.FromArgb(255, 0, 0, 255));
     ellipse.Stroke = brush;
     ellipse.StrokeThickness = 2;
     recognitionCanvas.Children.Add(ellipse);
 }

 // Draw a polygon on the recognitionCanvas.
 private void DrawPolygon(InkAnalysisInkDrawing shape)
 {
     List<Point> points = new List<Point>(shape.Points);
     Polygon polygon = new Polygon();

     foreach (Point point in points)
     {
```

```
            polygon.Points.Add(point);
        }

        var brush = new
SolidColorBrush(Windows.UI.ColorHelper.FromArgb(255, 0, 0, 255));
        polygon.Stroke = brush;
        polygon.StrokeThickness = 2;
        recognitionCanvas.Children.Add(polygon);
    }
```

Here's this sample in action:

Expand table

Before analysis	After analysis
	

Constrained handwriting recognition

In the preceding section (Free-form recognition with ink analysis), we demonstrated how to use the ink analysis APIs to analyze and recognize arbitrary ink strokes within an InkCanvas area.

In this section, we demonstrate how to use the Windows Ink handwriting recognition engine (not ink analysis) to convert a set of strokes on an **InkCanvas** to text (based on the installed default language pack).

> **Note**
>
> The basic handwriting recognition shown in this section is best suited for single-line, text input scenarios such as form input. For richer recognition scenarios that include analysis and interpretation of document structure, list items, shapes, and drawings (in addition to text recognition), see the previous section: **Free-form recognition with ink analysis**.

In this example, recognition is initiated when the user clicks a button to indicate they are finished writing.

Download this sample from Ink handwriting recognition sample

1. First, we set up the UI.

   The UI includes a "Recognize" button, the **InkCanvas**, and an area to display recognition results.

   XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
     <Grid.RowDefinitions>
         <RowDefinition Height="Auto"/>
         <RowDefinition Height="*"/>
     </Grid.RowDefinitions>
     <StackPanel x:Name="HeaderPanel"
                 Orientation="Horizontal"
                 Grid.Row="0">
             <TextBlock x:Name="Header"
                 Text="Basic ink recognition sample"
                 Style="{ThemeResource HeaderTextBlockStyle}"
                 Margin="10,0,0,0" />
             <Button x:Name="recognize"
                 Content="Recognize"
                 Margin="50,0,10,0"/>
     </StackPanel>
     <Grid Grid.Row="1">
         <Grid.RowDefinitions>
             <RowDefinition Height="*"/>
             <RowDefinition Height="Auto"/>
         </Grid.RowDefinitions>
         <InkCanvas x:Name="inkCanvas"
             Grid.Row="0"/>
         <TextBlock x:Name="recognitionResult"
             Grid.Row="1"
             Margin="50,0,10,0"/>
     </Grid>
 </Grid>
```

2. For this example, you need to first add the namespace type references required for our ink functionality:

   - Windows.UI.Input
   - Windows.UI.Input.Inking

3. We then set some basic ink input behaviors.

The InkPresenter is configured to interpret input data from both pen and mouse as ink strokes (InputDeviceTypes). Ink strokes are rendered on the InkCanvas using the specified InkDrawingAttributes. A listener for the click event on the "Recognize" button is also declared.

C#

```
public MainPage()
    {
        this.InitializeComponent();

        // Set supported inking device types.
        inkCanvas.InkPresenter.InputDeviceTypes =
            Windows.UI.Core.CoreInputDeviceTypes.Mouse |
            Windows.UI.Core.CoreInputDeviceTypes.Pen;

        // Set initial ink stroke attributes.
        InkDrawingAttributes drawingAttributes = new
InkDrawingAttributes();
        drawingAttributes.Color = Windows.UI.Colors.Black;
        drawingAttributes.IgnorePressure = false;
        drawingAttributes.FitToCurve = true;

inkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(drawingAttributes
);

        // Listen for button click to initiate recognition.
        recognize.Click += Recognize_Click;
    }
```

4. Finally, we perform the basic handwriting recognition. For this example, we use the click event handler of the "Recognize" button to perform the handwriting recognition.

    - An InkPresenter stores all ink strokes in an InkStrokeContainer object. The strokes are exposed through the StrokeContainer property of the **InkPresenter** and retrieved using the GetStrokes method.

    C#

    ```
    // Get all strokes on the InkCanvas.
        IReadOnlyList<InkStroke> currentStrokes =
    inkCanvas.InkPresenter.StrokeContainer.GetStrokes();
    ```

    - An InkRecognizerContainer is created to manage the handwriting recognition process.

    C#

```
// Create a manager for the InkRecognizer object
    // used in handwriting recognition.
    InkRecognizerContainer inkRecognizerContainer =
        new InkRecognizerContainer();
```

- RecognizeAsync is called to retrieve a set of InkRecognitionResult objects. Recognition results are produced for each word that is detected by an InkRecognizer.

C#

```
// Recognize all ink strokes on the ink canvas.
    IReadOnlyList<InkRecognitionResult> recognitionResults =
        await inkRecognizerContainer.RecognizeAsync(
            inkCanvas.InkPresenter.StrokeContainer,
            InkRecognitionTarget.All);
```

- Each InkRecognitionResult object contains a set of text candidates. The topmost item in this list is considered by the recognition engine to be the best match, followed by the remaining candidates in order of decreasing confidence.

  We iterate through each InkRecognitionResult and compile the list of candidates. The candidates are then displayed and the InkStrokeContainer is cleared (which also clears the InkCanvas).

C#

```
string str = "Recognition result\n";
    // Iterate through the recognition results.
    foreach (var result in recognitionResults)
    {
        // Get all recognition candidates from each recognition result.
        IReadOnlyList<string> candidates = result.GetTextCandidates();
        str += "Candidates: " + candidates.Count.ToString() + "\n";
        foreach (string candidate in candidates)
        {
            str += candidate + " ";
        }
    }
    // Display the recognition candidates.
    recognitionResult.Text = str;
    // Clear the ink canvas once recognition is complete.
    inkCanvas.InkPresenter.StrokeContainer.Clear();
```

- Here's the click handler example, in full.

C#

```
// Handle button click to initiate recognition.
    private async void Recognize_Click(object sender, RoutedEventArgs
e)
    {
        // Get all strokes on the InkCanvas.
        IReadOnlyList<InkStroke> currentStrokes =
inkCanvas.InkPresenter.StrokeContainer.GetStrokes();

        // Ensure an ink stroke is present.
        if (currentStrokes.Count > 0)
        {
            // Create a manager for the InkRecognizer object
            // used in handwriting recognition.
            InkRecognizerContainer inkRecognizerContainer =
                new InkRecognizerContainer();

            // inkRecognizerContainer is null if a recognition engine
is not available.
            if (!(inkRecognizerContainer == null))
            {
                // Recognize all ink strokes on the ink canvas.
                IReadOnlyList<InkRecognitionResult> recognitionResults
=
                    await inkRecognizerContainer.RecognizeAsync(
                        inkCanvas.InkPresenter.StrokeContainer,
                        InkRecognitionTarget.All);
                // Process and display the recognition results.
                if (recognitionResults.Count > 0)
                {
                    string str = "Recognition result\n";
                    // Iterate through the recognition results.
                    foreach (var result in recognitionResults)
                    {
                        // Get all recognition candidates from each
recognition result.
                        IReadOnlyList<string> candidates =
result.GetTextCandidates();
                        str += "Candidates: " +
candidates.Count.ToString() + "\n";
                        foreach (string candidate in candidates)
                        {
                            str += candidate + " ";
                        }
                    }
                    // Display the recognition candidates.
                    recognitionResult.Text = str;
                    // Clear the ink canvas once recognition is
complete.
                    inkCanvas.InkPresenter.StrokeContainer.Clear();
                }
                else
                {
```

```
                    recognitionResult.Text = "No recognition results.";
                }
            }
            else
            {
                Windows.UI.Popups.MessageDialog messageDialog = new
Windows.UI.Popups.MessageDialog("You must install handwriting
recognition engine.");
                await messageDialog.ShowAsync();
            }
        }
        else
        {
            recognitionResult.Text = "No ink strokes to recognize.";
        }
    }
```

International recognition

The handwriting recognition built into the Windows ink platform includes an extensive subset of locales and languages supported by Windows.

See the InkRecognizer.Name property topic for a list of languages supported by the InkRecognizer .

Your app can query the set of installed handwriting recognition engines and use one of those, or let a user select their preferred language.

Note Users can see a list of installed languages by going to **Settings -> Time & Language**. Installed languages are listed under **Languages**.

To install new language packs and enable handwriting recognition for that language:

1. Go to **Settings** > **Time & language** > **Region & language**.
2. Select **Add a language**.
3. Select a language from the list, then choose the region version. The language is now listed on the **Region & language** page.
4. Click the language and select **Options**.
5. On the **Language options** page, download the **Handwriting recognition engine** (they can also download the full language pack, speech recognition engine, and keyboard layout here).

Here, we demonstrate how to use the handwriting recognition engine to interpret a set of strokes on an InkCanvas based on the selected recognizer.

The recognition is initiated by the user clicking a button when they are finished writing.

1. First, we set up the UI.

   The UI includes a "Recognize" button, a combo box that lists all installed handwriting recognizers, the **InkCanvas**, and an area to display recognition results.

   XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <StackPanel x:Name="HeaderPanel"
                    Orientation="Horizontal"
                    Grid.Row="0">
            <TextBlock x:Name="Header"
                       Text="Advanced international ink recognition
sample"
                       Style="{ThemeResource HeaderTextBlockStyle}"
                       Margin="10,0,0,0" />
            <ComboBox x:Name="comboInstalledRecognizers"
                      Margin="50,0,10,0">
                <ComboBox.ItemTemplate>
                    <DataTemplate>
                        <StackPanel Orientation="Horizontal">
                            <TextBlock Text="{Binding Name}" />
                        </StackPanel>
                    </DataTemplate>
                </ComboBox.ItemTemplate>
            </ComboBox>
            <Button x:Name="buttonRecognize"
                    Content="Recognize"
                    IsEnabled="False"
                    Margin="50,0,10,0"/>
        </StackPanel>
        <Grid Grid.Row="1">
            <Grid.RowDefinitions>
                <RowDefinition Height="*"/>
                <RowDefinition Height="Auto"/>
            </Grid.RowDefinitions>
            <InkCanvas x:Name="inkCanvas"
                       Grid.Row="0"/>
            <TextBlock x:Name="recognitionResult"
                       Grid.Row="1"
                       Margin="50,0,10,0"/>
        </Grid>
    </Grid>
```

2. We then set some basic ink input behaviors.

   The **InkPresenter** is configured to interpret input data from both pen and mouse as ink strokes (**InputDeviceTypes**). Ink strokes are rendered on the **InkCanvas** using

the specified InkDrawingAttributes.

We call an `InitializeRecognizerList` function to populate the recognizer combo box with a list of installed handwriting recognizers.

We also declare listeners for the click event on the "Recognize" button and the selection changed event on the recognizer combo box.

C#

```
 public MainPage()
 {
    this.InitializeComponent();

    // Set supported inking device types.
    inkCanvas.InkPresenter.InputDeviceTypes =
        Windows.UI.Core.CoreInputDeviceTypes.Mouse |
        Windows.UI.Core.CoreInputDeviceTypes.Pen;

    // Set initial ink stroke attributes.
    InkDrawingAttributes drawingAttributes = new
InkDrawingAttributes();
    drawingAttributes.Color = Windows.UI.Colors.Black;
    drawingAttributes.IgnorePressure = false;
    drawingAttributes.FitToCurve = true;

inkCanvas.InkPresenter.UpdateDefaultDrawingAttributes(drawingAttributes
);

    // Populate the recognizer combo box with installed recognizers.
    InitializeRecognizerList();

    // Listen for combo box selection.
    comboInstalledRecognizers.SelectionChanged +=
        comboInstalledRecognizers_SelectionChanged;

    // Listen for button click to initiate recognition.
    buttonRecognize.Click += Recognize_Click;
}
```

3. We populate the recognizer combo box with a list of installed handwriting recognizers.

   An InkRecognizerContainer is created to manage the handwriting recognition process. Use this object to call GetRecognizers and retrieve the list of installed recognizers to populate the recognizer combo box.

   C#

```
// Populate the recognizer combo box with installed recognizers.
private void InitializeRecognizerList()
{
    // Create a manager for the handwriting recognition process.
    inkRecognizerContainer = new InkRecognizerContainer();
    // Retrieve the collection of installed handwriting recognizers.
    IReadOnlyList<InkRecognizer> installedRecognizers =
        inkRecognizerContainer.GetRecognizers();
    // inkRecognizerContainer is null if a recognition engine is not
available.
    if (!(inkRecognizerContainer == null))
    {
        comboInstalledRecognizers.ItemsSource = installedRecognizers;
        buttonRecognize.IsEnabled = true;
    }
}
```

4. Update the handwriting recognizer if the recognizer combo box selection changes.

   Use the InkRecognizerContainer to call SetDefaultRecognizer based on the selected recognizer from the recognizer combo box.

   C#

   ```
   // Handle recognizer change.
       private void comboInstalledRecognizers_SelectionChanged(
           object sender, SelectionChangedEventArgs e)
       {
           inkRecognizerContainer.SetDefaultRecognizer(
               (InkRecognizer)comboInstalledRecognizers.SelectedItem);
       }
   ```

5. Finally, we perform the handwriting recognition based on the selected handwriting recognizer. For this example, we use the click event handler of the "Recognize" button to perform the handwriting recognition.

   - An InkPresenter stores all ink strokes in an InkStrokeContainer object. The strokes are exposed through the StrokeContainer property of the **InkPresenter** and retrieved using the GetStrokes method.

   C#

   ```
   // Get all strokes on the InkCanvas.
       IReadOnlyList<InkStroke> currentStrokes =
           inkCanvas.InkPresenter.StrokeContainer.GetStrokes();
   ```

   - RecognizeAsync is called to retrieve a set of InkRecognitionResult objects.

Recognition results are produced for each word that is detected by an InkRecognizer.

```
C#

// Recognize all ink strokes on the ink canvas.
IReadOnlyList<InkRecognitionResult> recognitionResults =
    await inkRecognizerContainer.RecognizeAsync(
        inkCanvas.InkPresenter.StrokeContainer,
        InkRecognitionTarget.All);
```

- Each InkRecognitionResult object contains a set of text candidates. The topmost item in this list is considered by the recognition engine to be the best match, followed by the remaining candidates in order of decreasing confidence.

  We iterate through each InkRecognitionResult and compile the list of candidates. The candidates are then displayed and the InkStrokeContainer is cleared (which also clears the InkCanvas).

```
C#

string str = "Recognition result\n";
// Iterate through the recognition results.
foreach (InkRecognitionResult result in recognitionResults)
{
        // Get all recognition candidates from each recognition result.
        IReadOnlyList<string> candidates =
            result.GetTextCandidates();
        str += "Candidates: " + candidates.Count.ToString() + "\n";
        foreach (string candidate in candidates)
        {
            str += candidate + " ";
        }
}
// Display the recognition candidates.
recognitionResult.Text = str;
// Clear the ink canvas once recognition is complete.
inkCanvas.InkPresenter.StrokeContainer.Clear();
```

- Here's the click handler example, in full.

```
C#

// Handle button click to initiate recognition.
private async void Recognize_Click(object sender, RoutedEventArgs e)
{
    // Get all strokes on the InkCanvas.
    IReadOnlyList<InkStroke> currentStrokes =
```

```
        inkCanvas.InkPresenter.StrokeContainer.GetStrokes();

    // Ensure an ink stroke is present.
    if (currentStrokes.Count > 0)
    {
        // inkRecognizerContainer is null if a recognition engine is
not available.
        if (!(inkRecognizerContainer == null))
        {
            // Recognize all ink strokes on the ink canvas.
            IReadOnlyList<InkRecognitionResult> recognitionResults =
                await inkRecognizerContainer.RecognizeAsync(
                    inkCanvas.InkPresenter.StrokeContainer,
                    InkRecognitionTarget.All);
            // Process and display the recognition results.
            if (recognitionResults.Count > 0)
            {
                string str = "Recognition result\n";
                // Iterate through the recognition results.
                foreach (InkRecognitionResult result in
recognitionResults)
                {
                    // Get all recognition candidates from each
recognition result.
                    IReadOnlyList<string> candidates =
                        result.GetTextCandidates();
                    str += "Candidates: " + candidates.Count.ToString()
+ "\n";
                    foreach (string candidate in candidates)
                    {
                        str += candidate + " ";
                    }
                }
                // Display the recognition candidates.
                recognitionResult.Text = str;
                // Clear the ink canvas once recognition is complete.
                inkCanvas.InkPresenter.StrokeContainer.Clear();
            }
            else
            {
                recognitionResult.Text = "No recognition results.";
            }
        }
        else
        {
            Windows.UI.Popups.MessageDialog messageDialog =
                new Windows.UI.Popups.MessageDialog(
                    "You must install handwriting recognition
engine.");
            await messageDialog.ShowAsync();
        }
    }
    else
    {
        recognitionResult.Text = "No ink strokes to recognize.";
```

```
    }
}
```

Dynamic recognition

While, the previous two examples require the user to press a button to start recognition, you can also perform dynamic recognition using stroke input paired with a basic timing function.

For this example, we'll use the same UI and stroke settings as the previous international recognition example.

1. These global objects (InkAnalyzer, InkStroke, InkAnalysisResult, DispatcherTimer) are used throughout our app.

```
C#

// Stroke recognition globals.
InkAnalyzer inkAnalyzer;
DispatcherTimer recoTimer;
```

2. Instead of a button to initiate recognition, we add listeners for two **InkPresenter** stroke events (**StrokesCollected** and **StrokeStarted**), and set up a basic timer (**DispatcherTimer**) with a one second **Tick** interval.

```
C#

public MainPage()
{
    this.InitializeComponent();

    // Set supported inking device types.
    inkCanvas.InkPresenter.InputDeviceTypes =
        Windows.UI.Core.CoreInputDeviceTypes.Mouse |
        Windows.UI.Core.CoreInputDeviceTypes.Pen;

    // Listen for stroke events on the InkPresenter to
    // enable dynamic recognition.
    // StrokesCollected is fired when the user stops inking by
    // lifting their pen or finger, or releasing the mouse button.
    inkCanvas.InkPresenter.StrokesCollected +=
inkCanvas_StrokesCollected;
    // StrokeStarted is fired when ink input is first detected.
    inkCanvas.InkPresenter.StrokeInput.StrokeStarted +=
        inkCanvas_StrokeStarted;

    inkAnalyzer = new InkAnalyzer();
```

```
    // Timer to manage dynamic recognition.
    recoTimer = new DispatcherTimer();
    recoTimer.Interval = TimeSpan.FromSeconds(1);
    recoTimer.Tick += recoTimer_TickAsync;
}
```

3. We then define the handlers for the InkPresenter events we declared in the first step (we also override the **OnNavigatingFrom** page event to manage our timer).

   - **StrokesCollected**
     Add ink strokes (**AddDataForStrokes**) to the InkAnalyzer and start the recognition timer when the user stops inking (by lifting their pen or finger, or releasing the mouse button). After one second of no ink input, recognition is initiated.

     Use the **SetStrokeDataKind** method to specify whether you're interested only in text (including document structure amd bullet lists) or only in drawings (including shape recognition).

   - **StrokeStarted**
     If a new stroke starts before the next timer tick event, stop the timer as the new stroke is likely the continuation of a single handwriting entry.

C#

```
// Handler for the InkPresenter StrokeStarted event.
// Don't perform analysis while a stroke is in progress.
// If a new stroke starts before the next timer tick event,
// stop the timer as the new stroke is likely the continuation
// of a single handwriting entry.
private void inkCanvas_StrokeStarted(InkStrokeInput sender,
PointerEventArgs args)
{
    recoTimer.Stop();
}
// Handler for the InkPresenter StrokesCollected event.
// Stop the timer and add the collected strokes to the InkAnalyzer.
// Start the recognition timer when the user stops inking (by
// lifting their pen or finger, or releasing the mouse button).
// If ink input is not detected after one second, initiate recognition.
private void inkCanvas_StrokesCollected(InkPresenter sender,
InkStrokesCollectedEventArgs args)
{
    recoTimer.Stop();
    // If you're only interested in a specific type of recognition,
    // such as writing or drawing, you can constrain recognition
    // using the SetStrokDataKind method, which can improve both
    // efficiency and recognition results.
```

```
    // In this example, "InkAnalysisStrokeKind.Writing" is used.
    foreach (var stroke in args.Strokes)
    {
        inkAnalyzer.AddDataForStroke(stroke);
        inkAnalyzer.SetStrokeDataKind(stroke.Id,
InkAnalysisStrokeKind.Writing);
    }
    recoTimer.Start();
}
// Override the Page OnNavigatingFrom event handler to
// stop our timer if user leaves page.
protected override void OnNavigatingFrom(NavigatingCancelEventArgs e)
{
    recoTimer.Stop();
}
```

4. Finally, we perform the handwriting recognition. For this example, we use the **Tick** event handler of a **DispatcherTimer** to initiate the handwriting recognition.

   - Call **AnalyzeAsync** to initiate ink analysis and get the **InkAnalysisResult**.
   - If **Status** returns a state of **Updated**, call **FindNodes** for node types of **InkAnalysisNodeKind.InkWord**.
   - Iterate through the nodes and display the recognized text.
   - Finally, delete the recognized nodes from the InkAnalyzer and the corresponding ink strokes from the ink canvas.

C#

```
private async void recoTimer_TickAsync(object sender, object e)
{
    recoTimer.Stop();
    if (!inkAnalyzer.IsAnalyzing)
    {
        InkAnalysisResult result = await inkAnalyzer.AnalyzeAsync();

        // Have ink strokes on the canvas changed?
        if (result.Status == InkAnalysisStatus.Updated)
        {
            // Find all strokes that are recognized as handwriting and
            // create a corresponding ink analysis InkWord node.
            var inkwordNodes =
                inkAnalyzer.AnalysisRoot.FindNodes(
                    InkAnalysisNodeKind.InkWord);

            // Iterate through each InkWord node.
            // Display the primary recognized text (for this example,
            // we ignore alternatives), and then delete the
            // ink analysis data and recognized strokes.
            foreach (InkAnalysisInkWord node in inkwordNodes)
            {
                string recognizedText = node.RecognizedText;
```

```
                // Display the recognition candidates.
                recognitionResult.Text = recognizedText;

                foreach (var strokeId in node.GetStrokeIds())
                {
                    var stroke =

inkCanvas.InkPresenter.StrokeContainer.GetStrokeById(strokeId);
                    stroke.Selected = true;
                }
                inkAnalyzer.RemoveDataForStrokes(node.GetStrokeIds());
            }
            inkCanvas.InkPresenter.StrokeContainer.DeleteSelected();
        }
    }
    else
    {
        // Ink analyzer is busy. Wait a while and try again.
        recoTimer.Start();
    }
}
```

Related articles

- Pen and stylus interactions

Topic samples

- Ink analysis sample (basic) (C#)
- Ink handwriting recognition sample (C#)

Other samples

- Simple ink sample (C#/C++)
- Complex ink sample (C++)
- Ink sample (JavaScript)
- Get Started Tutorial: Support ink in your Windows app
- Coloring book sample
- Family notes sample

Store and retrieve Windows Ink stroke data

Article • 07/19/2021

Windows apps that support Windows Ink can serialize and deserialize ink strokes to an Ink Serialized Format (ISF) file. The ISF file is a GIF image with additional metadata for all ink stroke properties and behaviors. Apps that are not ink-enabled, can view the static GIF image, including alpha-channel background transparency.

> **Note**
>
> ISF is the most compact persistent representation of ink. It can be embedded within a binary document format, such as a GIF file, or placed directly on the Clipboard.
>
> The Ink Serialized Format (ISF) Specification can be downloaded from the **Microsoft Download Center**.

> **Important APIs**: InkCanvas, Windows.UI.Input.Inking

Save ink strokes to a file

Here, we demonstrate how to save ink strokes drawn on an **InkCanvas** control.

Download this sample from Save and load ink strokes from an Ink Serialized Format (ISF) file

1. First, we set up the UI.

   The UI includes "Save", "Load", and "Clear" buttons, and the **InkCanvas**.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <StackPanel x:Name="HeaderPanel" Orientation="Horizontal"
Grid.Row="0">
            <TextBlock x:Name="Header"
                       Text="Basic ink store sample"
                       Style="{ThemeResource HeaderTextBlockStyle}"
                       Margin="10,0,0,0" />
```

```
            <Button x:Name="btnSave"
                    Content="Save"
                    Margin="50,0,10,0"/>
            <Button x:Name="btnLoad"
                    Content="Load"
                    Margin="50,0,10,0"/>
            <Button x:Name="btnClear"
                    Content="Clear"
                    Margin="50,0,10,0"/>
        </StackPanel>
        <Grid Grid.Row="1">
            <InkCanvas x:Name="inkCanvas" />
        </Grid>
    </Grid>
```

2. We then set some basic ink input behaviors.

   The **InkPresenter** is configured to interpret input data from both pen and mouse as ink strokes (**InputDeviceTypes**), and listeners for the click events on the buttons are declared.

C#

```
public MainPage()
    {
        this.InitializeComponent();

        // Set supported inking device types.
        inkCanvas.InkPresenter.InputDeviceTypes =
            Windows.UI.Core.CoreInputDeviceTypes.Mouse |
            Windows.UI.Core.CoreInputDeviceTypes.Pen;

        // Listen for button click to initiate save.
        btnSave.Click += btnSave_Click;
        // Listen for button click to initiate load.
        btnLoad.Click += btnLoad_Click;
        // Listen for button click to clear ink canvas.
        btnClear.Click += btnClear_Click;
    }
```

3. Finally, we save the ink in the click event handler of the **Save** button.

   A **FileSavePicker** lets the user select both the file and the location where the ink data is saved.

   Once a file is selected, we open an **IRandomAccessStream** stream set to **ReadWrite**.

   We then call **SaveAsync** to serialize the ink strokes managed by the **InkStrokeContainer** to the stream.

C#

```
// Save ink data to a file.
    private async void btnSave_Click(object sender, RoutedEventArgs e)
    {
        // Get all strokes on the InkCanvas.
        IReadOnlyList<InkStroke> currentStrokes =
inkCanvas.InkPresenter.StrokeContainer.GetStrokes();

        // Strokes present on ink canvas.
        if (currentStrokes.Count > 0)
        {
            // Let users choose their ink file using a file picker.
            // Initialize the picker.
            Windows.Storage.Pickers.FileSavePicker savePicker =
                new Windows.Storage.Pickers.FileSavePicker();
            savePicker.SuggestedStartLocation =
                Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
            savePicker.FileTypeChoices.Add(
                "GIF with embedded ISF",
                new List<string>() { ".gif" });
            savePicker.DefaultFileExtension = ".gif";
            savePicker.SuggestedFileName = "InkSample";

            // Show the file picker.
            Windows.Storage.StorageFile file =
                await savePicker.PickSaveFileAsync();
            // When chosen, picker returns a reference to the selected file.
            if (file != null)
            {
                // Prevent updates to the file until updates are
                // finalized with call to CompleteUpdatesAsync.
                Windows.Storage.CachedFileManager.DeferUpdates(file);
                // Open a file stream for writing.
                IRandomAccessStream stream = await
file.OpenAsync(Windows.Storage.FileAccessMode.ReadWrite);
                // Write the ink strokes to the output stream.
                using (IOutputStream outputStream =
stream.GetOutputStreamAt(0))
                {
                    await
inkCanvas.InkPresenter.StrokeContainer.SaveAsync(outputStream);
                    await outputStream.FlushAsync();
                }
                stream.Dispose();

                // Finalize write so other apps can update file.
                Windows.Storage.Provider.FileUpdateStatus status =
                    await
Windows.Storage.CachedFileManager.CompleteUpdatesAsync(file);

                if (status ==
Windows.Storage.Provider.FileUpdateStatus.Complete)
                {
```

```
                    // File saved.
                }
                else
                {
                    // File couldn't be saved.
                }
            }
            // User selects Cancel and picker returns null.
            else
            {
                // Operation cancelled.
            }
        }
    }
```

> **Note**
>
> GIF is the only file format supported for saving ink data. However, the **LoadAsync** method (demonstrated in the next section) does support additional formats for backward compatibility.

Load ink strokes from a file

Here, we demonstrate how to load ink strokes from a file and render them on an **InkCanvas** control.

Download this sample from Save and load ink strokes from an Ink Serialized Format (ISF) file

1. First, we set up the UI.

   The UI includes "Save", "Load", and "Clear" buttons, and the **InkCanvas**.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <StackPanel x:Name="HeaderPanel" Orientation="Horizontal"
Grid.Row="0">
            <TextBlock x:Name="Header"
                       Text="Basic ink store sample"
                       Style="{ThemeResource HeaderTextBlockStyle}"
                       Margin="10,0,0,0" />
            <Button x:Name="btnSave"
```

```
                    Content="Save"
                    Margin="50,0,10,0"/>
            <Button x:Name="btnLoad"
                    Content="Load"
                    Margin="50,0,10,0"/>
            <Button x:Name="btnClear"
                    Content="Clear"
                    Margin="50,0,10,0"/>
        </StackPanel>
        <Grid Grid.Row="1">
            <InkCanvas x:Name="inkCanvas" />
        </Grid>
    </Grid>
```

2. We then set some basic ink input behaviors.

   The **InkPresenter** is configured to interpret input data from both pen and mouse as ink strokes (**InputDeviceTypes**), and listeners for the click events on the buttons are declared.

C#

```
public MainPage()
    {
        this.InitializeComponent();

        // Set supported inking device types.
        inkCanvas.InkPresenter.InputDeviceTypes =
            Windows.UI.Core.CoreInputDeviceTypes.Mouse |
            Windows.UI.Core.CoreInputDeviceTypes.Pen;

        // Listen for button click to initiate save.
        btnSave.Click += btnSave_Click;
        // Listen for button click to initiate load.
        btnLoad.Click += btnLoad_Click;
        // Listen for button click to clear ink canvas.
        btnClear.Click += btnClear_Click;
    }
```

3. Finally, we load the ink in the click event handler of the **Load** button.

   A **FileOpenPicker** lets the user select both the file and the location from where to retrieve the saved ink data.

   Once a file is selected, we open an **IRandomAccessStream** stream set to **Read**.

   We then call **LoadAsync** to read, de-serialize, and load the saved ink strokes into the **InkStrokeContainer**. Loading the strokes into the **InkStrokeContainer** causes the **InkPresenter** to immediately render them to the **InkCanvas**.

> **Note**
>
> All existing strokes in the InkStrokeContainer are cleared before new strokes are loaded.

C#

```
// Load ink data from a file.
private async void btnLoad_Click(object sender, RoutedEventArgs e)
{
    // Let users choose their ink file using a file picker.
    // Initialize the picker.
    Windows.Storage.Pickers.FileOpenPicker openPicker =
        new Windows.Storage.Pickers.FileOpenPicker();
    openPicker.SuggestedStartLocation =
        Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;
    openPicker.FileTypeFilter.Add(".gif");
    // Show the file picker.
    Windows.Storage.StorageFile file = await
openPicker.PickSingleFileAsync();
    // User selects a file and picker returns a reference to the selected
file.
    if (file != null)
    {
        // Open a file stream for reading.
        IRandomAccessStream stream = await
file.OpenAsync(Windows.Storage.FileAccessMode.Read);
        // Read from file.
        using (var inputStream = stream.GetInputStreamAt(0))
        {
            await
inkCanvas.InkPresenter.StrokeContainer.LoadAsync(inputStream);
        }
        stream.Dispose();
    }
    // User selects Cancel and picker returns null.
    else
    {
        // Operation cancelled.
    }
}
```

> **Note**
>
> GIF is the only file format supported for saving ink data. However, the **LoadAsync** method does support the following formats for backward compatibility.

Format	Description

Format	Description
InkSerializedFormat	Specifies ink that is persisted using ISF. This is the most compact persistent representation of ink. It can be embedded within a binary document format or placed directly on the Clipboard.
Base64InkSerializedFormat	Specifies ink that is persisted by encoding the ISF as a base64 stream. This format is provided so ink can be encoded directly in an XML or HTML file.
Gif	Specifies ink that is persisted by using a GIF file that contains ISF as metadata embedded within the file. This enables ink to be viewed in applications that are not ink-enabled and maintain its full ink fidelity when it returns to an ink-enabled application. This format is ideal when transporting ink content within an HTML file and for making it usable by ink and non-ink applications.
Base64Gif	Specifies ink that is persisted by using a base64-encoded fortified GIF. This format is provided when ink is to be encoded directly in an XML or HTML file for later conversion into an image. A possible use of this is in an XML format generated to contain all ink information and used to generate HTML through Extensible Stylesheet Language Transformations (XSLT).

Copy and paste ink strokes with the clipboard

Here, we demonstrate how to use the clipboard to transfer ink strokes between apps.

To support clipboard functionality, the built-in **InkStrokeContainer** cut and copy commands require one or more ink strokes be selected.

For this example, we enable stroke selection when input is modified with a pen barrel button (or right mouse button). For a complete example of how to implement stroke selection, see Pass-through input for advanced processing in Pen and stylus interactions.

Download this sample from Save and load ink strokes from the clipboard

1. First, we set up the UI.

   The UI includes "Cut", "Copy", "Paste", and "Clear" buttons, along with the **InkCanvas** and a selection canvas.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
```

```
        </Grid.RowDefinitions>
        <StackPanel x:Name="HeaderPanel" Orientation="Horizontal"
Grid.Row="0">
            <TextBlock x:Name="tbHeader"
                       Text="Basic ink store sample"
                       Style="{ThemeResource HeaderTextBlockStyle}"
                       Margin="10,0,0,0" />
            <Button x:Name="btnCut"
                    Content="Cut"
                    Margin="20,0,10,0"/>
            <Button x:Name="btnCopy"
                    Content="Copy"
                    Margin="20,0,10,0"/>
            <Button x:Name="btnPaste"
                    Content="Paste"
                    Margin="20,0,10,0"/>
            <Button x:Name="btnClear"
                    Content="Clear"
                    Margin="20,0,10,0"/>
        </StackPanel>
        <Grid x:Name="gridCanvas" Grid.Row="1">
            <!-- Canvas for displaying selection UI. -->
            <Canvas x:Name="selectionCanvas"/>
            <!-- Inking area -->
            <InkCanvas x:Name="inkCanvas"/>
        </Grid>
    </Grid>
```

2. We then set some basic ink input behaviors.

   The **InkPresenter** is configured to interpret input data from both pen and mouse as ink strokes (**InputDeviceTypes**). Listeners for the click events on the buttons as well as pointer and stroke events for selection functionality are also declared here.

   For a complete example of how to implement stroke selection, see Pass-through input for advanced processing in Pen and stylus interactions.

C#

```
public MainPage()
    {
        this.InitializeComponent();

        // Set supported inking device types.
        inkCanvas.InkPresenter.InputDeviceTypes =
            Windows.UI.Core.CoreInputDeviceTypes.Mouse |
            Windows.UI.Core.CoreInputDeviceTypes.Pen;

        // Listen for button click to cut ink strokes.
        btnCut.Click += btnCut_Click;
        // Listen for button click to copy ink strokes.
        btnCopy.Click += btnCopy_Click;
```

```
        // Listen for button click to paste ink strokes.
        btnPaste.Click += btnPaste_Click;
        // Listen for button click to clear ink canvas.
        btnClear.Click += btnClear_Click;

        // By default, the InkPresenter processes input modified by
        // a secondary affordance (pen barrel button, right mouse
        // button, or similar) as ink.
        // To pass through modified input to the app for custom processing
        // on the app UI thread instead of the background ink thread, set
        // InputProcessingConfiguration.RightDragAction to LeaveUnprocessed.
        inkCanvas.InkPresenter.InputProcessingConfiguration.RightDragAction
=
            InkInputRightDragAction.LeaveUnprocessed;

        // Listen for unprocessed pointer events from modified input.
        // The input is used to provide selection functionality.
        inkCanvas.InkPresenter.UnprocessedInput.PointerPressed +=
            UnprocessedInput_PointerPressed;
        inkCanvas.InkPresenter.UnprocessedInput.PointerMoved +=
            UnprocessedInput_PointerMoved;
        inkCanvas.InkPresenter.UnprocessedInput.PointerReleased +=
            UnprocessedInput_PointerReleased;

        // Listen for new ink or erase strokes to clean up selection UI.
        inkCanvas.InkPresenter.StrokeInput.StrokeStarted +=
            StrokeInput_StrokeStarted;
        inkCanvas.InkPresenter.StrokesErased +=
            InkPresenter_StrokesErased;
    }
```

3. Finally, after adding stroke selection support, we implement clipboard functionality in the click event handlers of the **Cut**, **Copy**, and **Paste** buttons.

   For cut, we first call **CopySelectedToClipboard** on the **InkStrokeContainer** of the **InkPresenter**.

   We then call **DeleteSelected** to remove the strokes from the ink canvas.

   Finally, we delete all selection strokes from the selection canvas.

C#

```
private void btnCut_Click(object sender, RoutedEventArgs e)
    {
        inkCanvas.InkPresenter.StrokeContainer.CopySelectedToClipboard();
        inkCanvas.InkPresenter.StrokeContainer.DeleteSelected();
        ClearSelection();
    }
```

C#

```
// Clean up selection UI.
    private void ClearSelection()
    {
        var strokes = inkCanvas.InkPresenter.StrokeContainer.GetStrokes();
        foreach (var stroke in strokes)
        {
            stroke.Selected = false;
        }
        ClearDrawnBoundingRect();
    }

    private void ClearDrawnBoundingRect()
    {
        if (selectionCanvas.Children.Any())
        {
            selectionCanvas.Children.Clear();
            boundingRect = Rect.Empty;
        }
    }
```

For copy, we simply call **CopySelectedToClipboard** on the **InkStrokeContainer** of the **InkPresenter**.

C#

```
private void btnCopy_Click(object sender, RoutedEventArgs e)
    {
        inkCanvas.InkPresenter.StrokeContainer.CopySelectedToClipboard();
    }
```

For paste, we call **CanPasteFromClipboard** to ensure that the content on the clipboard can be pasted to the ink canvas.

If so, we call **PasteFromClipboard** to insert the clipboard ink strokes into the **InkStrokeContainer** of the **InkPresenter**, which then renders the strokes to the ink canvas.

C#

```
private void btnPaste_Click(object sender, RoutedEventArgs e)
    {
        if (inkCanvas.InkPresenter.StrokeContainer.CanPasteFromClipboard())
        {
            inkCanvas.InkPresenter.StrokeContainer.PasteFromClipboard(
                new Point(0, 0));
        }
        else
        {
            // Cannot paste from clipboard.
```

```
        }
    }
```

Related articles

- Pen and stylus interactions

Topic samples

- Save and load ink strokes from an Ink Serialized Format (ISF) file
- Save and load ink strokes from the clipboard

Other samples

- Simple ink sample (C#/C++)
- Complex ink sample (C++)
- Ink sample (JavaScript)
- Get Started Tutorial: Support ink in your Windows app
- Coloring book sample
- Family notes sample

Add an InkToolbar to a Windows app

Article • 05/26/2022

There are two different controls that facilitate inking in Windows apps: **InkCanvas** and **InkToolbar**.

The **InkCanvas** control provides basic Windows Ink functionality. Use it to render pen input as either an ink stroke (using default settings for color and thickness) or an erase stroke.

> For InkCanvas implementation details, see Pen and stylus interactions in Windows apps.

As a completely transparent overlay, the InkCanvas does not provide any built-in UI for setting ink stroke properties. If you want to change the default inking experience, let users set ink stroke properties, and support other custom inking features, you have two options:

- In code-behind, use the underlying **InkPresenter** object bound to the InkCanvas.

  The InkPresenter APIs support extensive customization of the inking experience. For more detail, see Pen and stylus interactions in Windows apps.

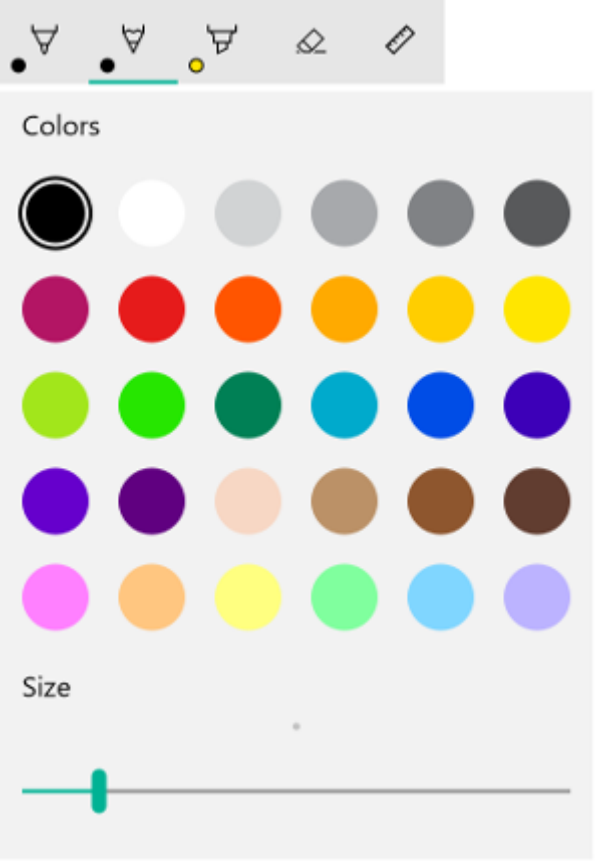
- Bind an **InkToolbar** to the InkCanvas. By default, the InkToolbar provides a customizable and extensible collection of buttons for activating ink-related features such as stroke size, ink color, and pen tip.

  We discuss the InkToolbar in this topic.

> **Important APIs**: **InkCanvas class**, **InkToolbar class**, **InkPresenter class**, **Windows.UI.Input.Inking**

Default InkToolbar

By default, the **InkToolbar** includes buttons for drawing, erasing, highlighting, and displaying a stencil (ruler or protractor). Depending on the feature, other settings and commands, such as ink color, stroke thickness, erase all ink, are provided in a flyout.



Default Windows Ink toolbar

To add a default **InkToolbar** to an inking app, just place it on the same page as your **InkCanvas** and associate the two controls.

1. In MainPage.xaml, declare a container object (for this example, we use a Grid control) for the inking surface.
2. Declare an InkCanvas object as a child of the container. (The InkCanvas size is inherited from the container.)
3. Declare an InkToolbar and use the TargetInkCanvas attribute to bind it to the InkCanvas.

> **Note**
>
> Ensure the InkToolbar is declared after the InkCanvas. If not, the InkCanvas overlay renders the InkToolbar inaccessible.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel x:Name="HeaderPanel" Orientation="Horizontal" Grid.Row="0">
        <TextBlock x:Name="Header"
```

```
                    Text="Basic ink sample"
                    Style="{ThemeResource HeaderTextBlockStyle}"
                    Margin="10,0,0,0" />
    </StackPanel>
    <Grid Grid.Row="1">
        <Image Source="Assets\StoreLogo.png" />
        <InkCanvas x:Name="inkCanvas" />
        <InkToolbar x:Name="inkToolbar"
          VerticalAlignment="Top"
          TargetInkCanvas="{x:Bind inkCanvas}" />
    </Grid>
</Grid>
```

Basic customization

In this section, we cover some basic Windows Ink toolbar customization scenarios.

Specify location and orientation

When you add an ink toolbar to your app, you can accept the default location and orientation of the toolbar or set them as required by your app or user.

XAML

Explicitly specify the location and orientation of the toolbar through its VerticalAlignment, HorizontalAlignment, and Orientation properties.

Expand table

Default	Explicit
Windows Ink toolbar default location and orientation	*Windows Ink toolbar explicit location and orientation*

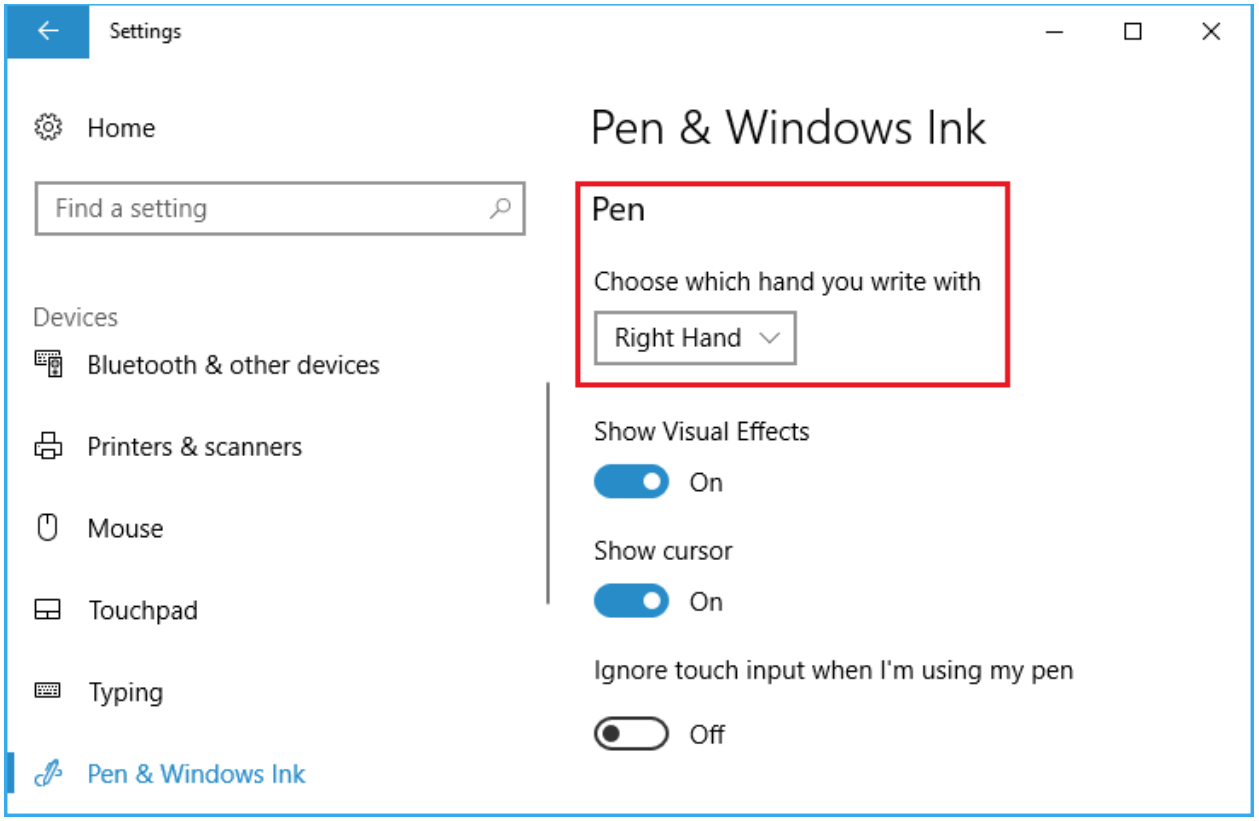
Here's the code for explicitly setting the location and orientation of the ink toolbar in XAML.

XAML

```
<InkToolbar x:Name="inkToolbar"
    VerticalAlignment="Center"
    HorizontalAlignment="Right"
    Orientation="Vertical"
    TargetInkCanvas="{x:Bind inkCanvas}" />
```

Initialize based on user preferences or device state

In some cases, you might want to set the location and orientation of the ink toolbar based on user preference or device state. The following example demonstrates how to set the location and orientation of the ink toolbar based on the left or right-hand writing preferences specified through **Settings** > **Devices** > **Pen & Windows Ink** > **Pen** > **Choose which hand you write with**.



Dominant hand setting

You can query this setting through the HandPreference property of Windows.UI.ViewManagement and set the HorizontalAlignment based on the value returned. In this example, we locate the toolbar on the left side of the app for a left-handed person and on the right side for a right-handed person.

Download this sample from Ink toolbar location and orientation sample (basic)

C#

```
public MainPage()
{
    this.InitializeComponent();

    Windows.UI.ViewManagement.UISettings settings =
        new Windows.UI.ViewManagement.UISettings();
    HorizontalAlignment alignment =
        (settings.HandPreference ==
            Windows.UI.ViewManagement.HandPreference.LeftHanded) ?
            HorizontalAlignment.Left : HorizontalAlignment.Right;
    inkToolbar.HorizontalAlignment = alignment;
}
```

Dynamically adjust to user or device state

You can also use binding to look after UI updates based on changes to user preferences, device settings, or device states. In the following example, we expand on the previous example and show how to dynamically position the ink toolbar based on device orientation using binding, a ViewMOdel object, and the INotifyPropertyChanged interface.

Download this sample from Ink toolbar location and orientation sample (dynamic)

1. First, let's add our ViewModel.

    a. Add a new folder to your project and call it **ViewModels**.

    b. Add a new class to the ViewModels folder (for this example, we called it **InkToolbarSnippetHostViewModel.cs**).

    > **Note**
    >
    > We used the Singleton pattern as we only need one object of this type for the life of the application

    c. Add `using System.ComponentModel` namespace to the file.

    d. Add a static member variable called **instance**, and a static read only property named **Instance**. Make the constructor private to ensure this class can only be accessed via the Instance property.

    > **Note**

This class inherits from INotifyPropertyChanged interface, which is used to notify clients, typically binding clients, that a property value has changed. We'll be using this to handle changes to the device orientation (we'll expand this code and explain further in a later step).

C#

```
using System.ComponentModel;

namespace locationandorientation.ViewModels
{
    public class InkToolbarSnippetHostViewModel :
INotifyPropertyChanged
    {
        private static InkToolbarSnippetHostViewModel instance;

        public static InkToolbarSnippetHostViewModel Instance
        {
            get
            {
                if (null == instance)
                {
                    instance = new InkToolbarSnippetHostViewModel();
                }
                return instance;
            }
        }
    }

    private InkToolbarSnippetHostViewModel() { }
}
```

e. Add two bool properties to the InkToolbarSnippetHostViewModel class: **LeftHandedLayout** (same functionality as the previous XAML-only example) and **PortraitLayout** (orientation of the device).

> ⓘ Note
>
> The PortraitLayout property is settable and includes the definition for the PropertyChanged event.

C#

```
public bool LeftHandedLayout
{
    get
    {
```

```
            bool leftHandedLayout = false;
            Windows.UI.ViewManagement.UISettings settings =
                new Windows.UI.ViewManagement.UISettings();
            leftHandedLayout = (settings.HandPreference ==
                Windows.UI.ViewManagement.HandPreference.LeftHanded);
            return leftHandedLayout;
        }
    }

    public bool portraitLayout = false;
    public bool PortraitLayout
    {
        get
        {
            Windows.UI.ViewManagement.ApplicationViewOrientation
    winOrientation =

    Windows.UI.ViewManagement.ApplicationView.GetForCurrentView().Orient
    ation;
            portraitLayout =
                (winOrientation ==

    Windows.UI.ViewManagement.ApplicationViewOrientation.Portrait);
            return portraitLayout;
        }
        set
        {
            if (value.Equals(portraitLayout)) return;
            portraitLayout = value;
            PropertyChanged?.Invoke(this, new
    PropertyChangedEventArgs("PortraitLayout"));
        }
    }
```

2. Now, let's add a couple of converter classes to our project. Each class contains a Convert object that returns an alignment value (either HorizontalAlignment or VerticalAlignment).

    a. Add a new folder to your project and call it **Converters**.

    b. Add two new classes to the Converters folder (for this example, we call them **HorizontalAlignmentFromHandednessConverter.cs** and **VerticalAlignmentFromAppViewConverter.cs**).

    c. Add `using Windows.UI.Xaml` and `using Windows.UI.Xaml.Data` namespaces to each file.

    d. Change each class to `public` and specify that it implements the IValueConverter interface.

e. Add the Convert and ConvertBack methods to each file, as shown here (we leave the ConvertBack method unimplemented).

- HorizontalAlignmentFromHandednessConverter positions the ink toolbar to the right side of the app for right-handed users and to the left side of the app for left-handed users.

C#

```
using System;

using Windows.UI.Xaml;
using Windows.UI.Xaml.Data;

namespace locationandorientation.Converters
{
    public class HorizontalAlignmentFromHandednessConverter :
IValueConverter
    {
        public object Convert(object value, Type targetType,
            object parameter, string language)
        {
            bool leftHanded = (bool)value;
            HorizontalAlignment alignment =
HorizontalAlignment.Right;
            if (leftHanded)
            {
                alignment = HorizontalAlignment.Left;
            }
            return alignment;
        }

        public object ConvertBack(object value, Type targetType,
            object parameter, string language)
        {
            throw new NotImplementedException();
        }
    }
}
```

- VerticalAlignmentFromAppViewConverter positions the ink toolbar to the center of the app for portrait orientation and to the top of the app for landscape orientation (while intended to improve usability, this is just an arbitrary choice for demonstration purposes).

C#

```
using System;

using Windows.UI.Xaml;
```

```
using Windows.UI.Xaml.Data;

namespace locationandorientation.Converters
{
    public class VerticalAlignmentFromAppViewConverter :
IValueConverter
    {
        public object Convert(object value, Type targetType,
            object parameter, string language)
        {
            bool portraitOrientation = (bool)value;
            VerticalAlignment alignment = VerticalAlignment.Top;
            if (portraitOrientation)
            {
                alignment = VerticalAlignment.Center;
            }
            return alignment;
        }

        public object ConvertBack(object value, Type targetType,
            object parameter, string language)
        {
            throw new NotImplementedException();
        }
    }
}
```

3. Now, open the MainPage.xaml.cs file.
   a. Add `using using locationandorientation.ViewModels` to the list of namespaces to associate our ViewModel.
   b. Add `using Windows.UI.ViewManagement` to the list of namespaces to enable listening for changes to the device orientation.
   c. Add the WindowSizeChangedEventHandler code.
   d. Set the DataContext for the view to the singleton instance of the InkToolbarSnippetHostViewModel class.

C#

```
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;

using locationandorientation.ViewModels;
using Windows.UI.ViewManagement;

namespace locationandorientation
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
```

```
            Window.Current.SizeChanged += (sender, args) =>
            {
                ApplicationView currentView =
ApplicationView.GetForCurrentView();

                if (currentView.Orientation ==
ApplicationViewOrientation.Landscape)
                {

InkToolbarSnippetHostViewModel.Instance.PortraitLayout = false;
                }
                else if (currentView.Orientation ==
ApplicationViewOrientation.Portrait)
                {

InkToolbarSnippetHostViewModel.Instance.PortraitLayout = true;
                }
            };

            DataContext = InkToolbarSnippetHostViewModel.Instance;
        }
    }
}
```

4. Next, open the MainPage.xaml file.

    a. Add `xmlns:converters="using:locationandorientation.Converters"` to the `Page` element for binding to our converters.

    XAML

    ```
    <Page
    x:Class="locationandorientation.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:locationandorientation"
    xmlns:converters="using:locationandorientation.Converters"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
    compatibility/2006"
    mc:Ignorable="d">
    ```

    b. Add a `PageResources` element and specify references to our converters.

    XAML

    ```
    <Page.Resources>
        <converters:HorizontalAlignmentFromHandednessConverter
    x:Key="HorizontalAlignmentConverter"/>
        <converters:VerticalAlignmentFromAppViewConverter
    ```

```
x:Key="VerticalAlignmentConverter"/>
</Page.Resources>
```

c. Add the InkCanvas and InkToolbar elements and bind the VerticalAlignment and HorizontalAlignment properties of the InkToolbar.

XAML

```
<InkCanvas x:Name="inkCanvas" />
<InkToolbar x:Name="inkToolbar"
            VerticalAlignment="{Binding PortraitLayout, Converter=
{StaticResource VerticalAlignmentConverter} }"
            HorizontalAlignment="{Binding LeftHandedLayout,
Converter={StaticResource HorizontalAlignmentConverter} }"
            Orientation="Vertical"
            TargetInkCanvas="{x:Bind inkCanvas}" />
```

5. Return to the InkToolbarSnippetHostViewModel.cs file to add our `PortraitLayout` and `LeftHandedLayout` bool properties to the `InkToolbarSnippetHostViewModel` class, along with support for rebinding `PortraitLayout` when that property value changes.

C#

```
public bool LeftHandedLayout
{
    get
    {
        bool leftHandedLayout = false;
        Windows.UI.ViewManagement.UISettings settings =
            new Windows.UI.ViewManagement.UISettings();
        leftHandedLayout = (settings.HandPreference ==
            Windows.UI.ViewManagement.HandPreference.LeftHanded);
        return leftHandedLayout;
    }
}

public bool portraitLayout = false;
public bool PortraitLayout
{
    get
    {
        Windows.UI.ViewManagement.ApplicationViewOrientation
winOrientation =

Windows.UI.ViewManagement.ApplicationView.GetForCurrentView().Orientati
on;
        portraitLayout =
            (winOrientation ==
```

```
Windows.UI.ViewManagement.ApplicationViewOrientation.Portrait);
        return portraitLayout;
    }
    set
    {
        if (value.Equals(portraitLayout)) return;
        portraitLayout = value;
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs("PortraitLayout"));
    }
}

#region INotifyPropertyChanged Members

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string property)
{
    PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(property));
}

#endregion
```

You should now have an inking app that adapts to both the dominant hand preference of the user and dynamically responds to the orientation of the user's device.

Specify the selected button

Windows Ink toolbar with pencil button selected at initialization

By default, the first (or leftmost) button is selected when your app is launched and the toolbar is initialized. In the default Windows Ink toolbar, this is the ballpoint pen button.

Because the framework defines the order of the built-in buttons, the first button might not be the pen or tool you want to activate by default.

You can override this default behavior and specify the selected button on the toolbar.

For this example, we initialize the default toolbar with the pencil button selected and the pencil activated (instead of the ballpoint pen).

1. Use the XAML declaration for the InkCanvas and InkToolbar from the previous example.
2. In code-behind, set up a handler for the Loaded event of the InkToolbar object.

C#

```
/// <summary>
/// An empty page that can be used on its own or navigated to within a
Frame.
/// Here, we set up InkToolbar event listeners.
/// </summary>
public MainPage_CodeBehind()
{
    this.InitializeComponent();
    // Add handlers for InkToolbar events.
    inkToolbar.Loaded += inkToolbar_Loaded;
}
```

3. In the handler for the Loaded event:
   a. Get a reference to the built-in InkToolbarPencilButton.

   Passing an InkToolbarTool.Pencil object in the GetToolButton method returns an InkToolbarToolButton object for the InkToolbarPencilButton.

   b. Set ActiveTool to the object returned in the previous step.

CSharp

```
/// <summary>
/// Handle the Loaded event of the InkToolbar.
/// By default, the active tool is set to the first tool on the toolbar.
/// Here, we set the active tool to the pencil button.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void inkToolbar_Loaded(object sender, RoutedEventArgs e)
{
    InkToolbarToolButton pencilButton =
inkToolbar.GetToolButton(InkToolbarTool.Pencil);
    inkToolbar.ActiveTool = pencilButton;
}
```

Specify the built-in buttons

Specific buttons included at initialization

As mentioned, the Windows Ink toolbar includes a collection of default, built-in buttons. These buttons are displayed in the following order (from left to right):

- InkToolbarBallpointPenButton

- InkToolbarPencilButton
- InkToolbarHighlighterButton
- InkToolbarEraserButton
- InkToolbarRulerButton

For this example, we initialize the toolbar with only the built-in ballpoint pen, pencil, and eraser buttons.

You can do this using either XAML or code-behind.

XAML

Modify the XAML declaration for the InkCanvas and InkToolbar from the first example.

- Add an InitialControls attribute and set its value to "None". This clears the default collection of built-in buttons.
- Add the specific InkToolbar buttons required by your app. Here, we add InkToolbarBallpointPenButton, InkToolbarPencilButton, and InkToolbarEraserButton only.

> **Note**
>
> Buttons are added to the toolbar in the order defined by the framework, not the order specified here.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel x:Name="HeaderPanel" Orientation="Horizontal" Grid.Row="0">
        <TextBlock x:Name="Header"
                   Text="Basic ink sample"
                   Style="{ThemeResource HeaderTextBlockStyle}"
                   Margin="10,0,0,0" />
    </StackPanel>
    <Grid Grid.Row="1">
        <Image Source="Assets\StoreLogo.png" />
        <!-- Clear the default InkToolbar buttons by setting InitialControls
to None. -->
        <!-- Set the active tool to the pencil button. -->
        <InkCanvas x:Name="inkCanvas" />
        <InkToolbar x:Name="inkToolbar"
                    VerticalAlignment="Top"
                    TargetInkCanvas="{x:Bind inkCanvas}"
                    InitialControls="None">
```

```
            <!--
             Add only the ballpoint pen, pencil, and eraser.
             Note that the buttons are added to the toolbar in the order
             defined by the framework, not the order we specify here.
            -->
            <InkToolbarEraserButton />
            <InkToolbarBallpointPenButton />
            <InkToolbarPencilButton/>
        </InkToolbar>
    </Grid>
</Grid>
```

Code-behind

1. Use the XAML declaration for the InkCanvas and InkToolbar from the first example.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel x:Name="HeaderPanel" Orientation="Horizontal" Grid.Row="0">
        <TextBlock x:Name="Header"
                   Text="Basic ink sample"
                   Style="{ThemeResource HeaderTextBlockStyle}"
                   Margin="10,0,0,0" />
    </StackPanel>
    <Grid Grid.Row="1">
        <Image Source="Assets\StoreLogo.png" />
        <InkCanvas x:Name="inkCanvas" />
        <InkToolbar x:Name="inkToolbar"
        VerticalAlignment="Top"
        TargetInkCanvas="{x:Bind inkCanvas}" />
    </Grid>
</Grid>
```

2. In code-behind, set up a handler for the Loading event of the InkToolbar object.

C#

```
/// <summary>
/// An empty page that can be used on its own or navigated to within a
Frame.
/// Here, we set up InkToolbar event listeners.
/// </summary>
public MainPage_CodeBehind()
{
    this.InitializeComponent();
    // Add handlers for InkToolbar events.
```

```
    inkToolbar.Loading += inkToolbar_Loading;
}
```

3. Set InitialControls to "None".
4. Create object references for the buttons required by your app. Here, we add InkToolbarBallpointPenButton, InkToolbarPencilButton, and InkToolbarEraserButton only.

> **Note**
>
> Buttons are added to the toolbar in the order defined by the framework, not the order specified here.

5. Add the buttons to the InkToolbar.

C#

```
/// <summary>
/// Handles the Loading event of the InkToolbar.
/// Here, we identify the buttons to include on the InkToolbar.
/// </summary>
/// <param name="sender">The InkToolbar</param>
/// <param name="args">The InkToolbar event data.
/// If there is no event data, this parameter is null</param>
private void inkToolbar_Loading(FrameworkElement sender, object args)
{
    // Clear all built-in buttons from the InkToolbar.
    inkToolbar.InitialControls = InkToolbarInitialControls.None;

    // Add only the ballpoint pen, pencil, and eraser.
    // Note that the buttons are added to the toolbar in the order
    // defined by the framework, not the order we specify here.
    InkToolbarBallpointPenButton ballpoint = new
InkToolbarBallpointPenButton();
    InkToolbarPencilButton pencil = new InkToolbarPencilButton();
    InkToolbarEraserButton eraser = new InkToolbarEraserButton();
    inkToolbar.Children.Add(eraser);
    inkToolbar.Children.Add(ballpoint);
    inkToolbar.Children.Add(pencil);
}
```

Custom buttons and inking features

You can customize and extend the collection of buttons (and associated inking features) that are provided through the InkToolbar.

The InkToolbar consists of two distinct groups of button types:

1. A group of "tool" buttons containing the built-in drawing, erasing, and highlighting buttons. Custom pens and tools are added here.

> **Note** Feature selection is mutually exclusive.

2. A group of "toggle" buttons containing the built-in ruler button. Custom toggles are added here.

> **Note** Features are not mutually exclusive and can be used concurrently with other active tools.

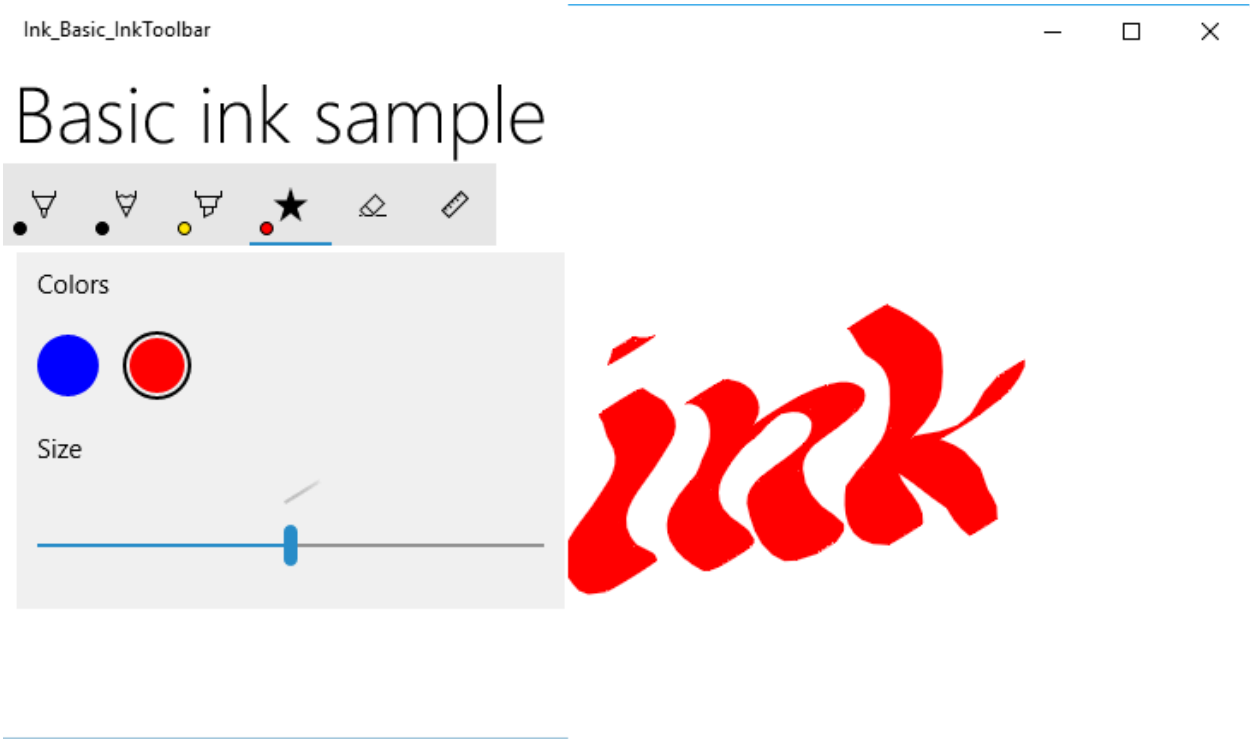
Depending on your application and the inking functionality required, you can add any of the following buttons (bound to your custom ink features) to the InkToolbar:

- Custom pen – a pen for which the ink color palette and pen tip properties, such as shape, rotation, and size, are defined by the host app.
- Custom tool – a non-pen tool, defined by the host app.
- Custom toggle – Sets the state of an app-defined feature to on or off. When turned on, the feature works in conjunction with the active tool.

> **Note** You cannot change the display order of the built-in buttons. The default display order is: Ballpoint pen, pencil, highlighter, eraser, and ruler. Custom pens are appended to the last default pen, custom tool buttons are added between the last pen button and the eraser button and custom toggle buttons are added after the ruler button. (Custom buttons are added in the order they are specified.)

Custom pen

You can create a custom pen (activated through a custom pen button) where you define the ink color palette and pen tip properties, such as shape, rotation, and size.



Custom calligraphic pen button

For this example, we define a custom pen with a broad tip that enables basic calligraphic ink strokes. We also customize the collection of brushes in the palette displayed on the button flyout.

Code-behind

First, we define our custom pen and specify the drawing attributes in code-behind. We reference this custom pen from XAML later.

1. Right click the project in Solution Explorer and select Add -> New item.
2. Under Visual C# -> Code, add a new Class file and call it CalligraphicPen.cs.
3. In Calligraphic.cs, replace the default using block with the following:

```
C#

using System.Numerics;
using Windows.UI;
using Windows.UI.Input.Inking;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;
```

4. Specify that the CalligraphicPen class is derived from InkToolbarCustomPen.

```
C#

class CalligraphicPen : InkToolbarCustomPen
{
```

```
}
```

5. Override CreateInkDrawingAttributesCore to specify your own brush and stroke size.

C#

```
class CalligraphicPen : InkToolbarCustomPen
{
    protected override InkDrawingAttributes
      CreateInkDrawingAttributesCore(Brush brush, double strokeWidth)
    {
    }
}
```

6. Create an InkDrawingAttributes object and set the pen tip shape, tip rotation, stroke size, and ink color.

C#

```
class CalligraphicPen : InkToolbarCustomPen
{
    protected override InkDrawingAttributes
      CreateInkDrawingAttributesCore(Brush brush, double strokeWidth)
    {
        InkDrawingAttributes inkDrawingAttributes =
          new InkDrawingAttributes();
        inkDrawingAttributes.PenTip = PenTipShape.Circle;
        inkDrawingAttributes.Size =
          new Windows.Foundation.Size(strokeWidth, strokeWidth * 20);
        SolidColorBrush solidColorBrush = brush as SolidColorBrush;
        if (solidColorBrush != null)
        {
            inkDrawingAttributes.Color = solidColorBrush.Color;
        }
        else
        {
            inkDrawingAttributes.Color = Colors.Black;
        }

        Matrix3x2 matrix = Matrix3x2.CreateRotation(45);
        inkDrawingAttributes.PenTipTransform = matrix;

        return inkDrawingAttributes;
    }
}
```

XAML

Next, we add the necessary references to the custom pen in MainPage.xaml.

1. We declare a local page resource dictionary that creates a reference to the custom pen (`CalligraphicPen`) defined in CalligraphicPen.cs, and a brush collection supported by the custom pen (`CalligraphicPenPalette`).

XAML

```
<Page.Resources>
    <!-- Add the custom CalligraphicPen to the page resources. -->
    <local:CalligraphicPen x:Key="CalligraphicPen" />
    <!-- Specify the colors for the palette of the custom pen. -->
    <BrushCollection x:Key="CalligraphicPenPalette">
        <SolidColorBrush Color="Blue" />
        <SolidColorBrush Color="Red" />
    </BrushCollection>
</Page.Resources>
```

2. We then add an InkToolbar with a child InkToolbarCustomPenButton element.

The custom pen button includes the two static resource references declared in the page resources: `CalligraphicPen` and `CalligraphicPenPalette`.

We also specify the range for the stroke size slider (MinStrokeWidth, MaxStrokeWidth, and SelectedStrokeWidth), the selected brush (SelectedBrushIndex), and the icon for the custom pen button (SymbolIcon).

XAML

```
<Grid Grid.Row="1">
    <InkCanvas x:Name="inkCanvas" />
    <InkToolbar x:Name="inkToolbar"
                VerticalAlignment="Top"
                TargetInkCanvas="{x:Bind inkCanvas}">
        <InkToolbarCustomPenButton
            CustomPen="{StaticResource CalligraphicPen}"
            Palette="{StaticResource CalligraphicPenPalette}"
            MinStrokeWidth="1" MaxStrokeWidth="3" SelectedStrokeWidth="2"
            SelectedBrushIndex ="1">
            <SymbolIcon Symbol="Favorite" />
            <InkToolbarCustomPenButton.ConfigurationContent>
                <InkToolbarPenConfigurationControl />
            </InkToolbarCustomPenButton.ConfigurationContent>
        </InkToolbarCustomPenButton>
    </InkToolbar>
</Grid>
```

Custom toggle

You can create a custom toggle (activated through a custom toggle button) to set the state of an app-defined feature to on or off. When turned on, the feature works in conjunction with the active tool.

In this example, we define a custom toggle button that enables inking with touch input (by default, touch inking is not enabled).

> ⓘ **Note**
>
> If you need to support inking with touch, we recommended that you enable it using a CustomToggleButton, with the icon and **tooltip** specified in this example.

Typically, touch input is used for direct manipulation of an object or the app UI. To demonstrate the differences in behavior when touch inking is enabled, we place the InkCanvas within a ScrollViewer container and set the dimensions of the ScrollViewer to be smaller than the InkCanvas.

When the app starts, only pen inking is supported and touch is used to pan or zoom the inking surface. When touch inking is enabled, the inking surface cannot be panned or zoomed through touch input.

> ⓘ **Note**
>
> See **Inking controls** for both **InkCanvas** and **InkToolbar** UX guidelines. The following recommendations are relevant to this example:
>
> - The **InkToolbar**, and inking in general, is best experienced through an active pen. However, inking with mouse and touch can be supported if required by your app.
> - If supporting inking with touch input, we recommend using the "ED5F" icon from the "Segoe MLD2 Assets" font for the toggle button, with a "Touch writing" tooltip.

XAML

1. First, we declare an **InkToolbarCustomToggleButton** element (toggleButton) with a Click event listener that specifies the event handler (Toggle_Custom).

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
```

```
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <StackPanel Grid.Row="0"
                x:Name="HeaderPanel"
                Orientation="Horizontal">
        <TextBlock x:Name="Header"
                   Text="Basic ink sample"
                   Style="{ThemeResource HeaderTextBlockStyle}"
                   Margin="10" />
    </StackPanel>

    <ScrollViewer Grid.Row="1"
                  HorizontalScrollBarVisibility="Auto"
                  VerticalScrollBarVisibility="Auto">

        <Grid HorizontalAlignment="Left" VerticalAlignment="Top">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>

            <InkToolbar Grid.Row="0"
                        Margin="10"
                        x:Name="inkToolbar"
                        VerticalAlignment="Top"
                        TargetInkCanvas="{x:Bind inkCanvas}">
                <InkToolbarCustomToggleButton
                x:Name="toggleButton"
                Click="CustomToggle_Click"
                ToolTipService.ToolTip="Touch Writing">
                    <SymbolIcon Symbol="{x:Bind TouchWritingIcon}"/>
                </InkToolbarCustomToggleButton>
            </InkToolbar>

            <ScrollViewer Grid.Row="1"
                          Height="500"
                          Width="500"
                          x:Name="scrollViewer"
                          ZoomMode="Enabled"
                          MinZoomFactor=".1"
                          VerticalScrollMode="Enabled"
                          VerticalScrollBarVisibility="Auto"
                          HorizontalScrollMode="Enabled"
                          HorizontalScrollBarVisibility="Auto">

                <Grid x:Name="outputGrid"
                      Height="1000"
                      Width="1000"
                      Background="{ThemeResource
SystemControlBackgroundChromeWhiteBrush}">
                    <InkCanvas x:Name="inkCanvas"/>
                </Grid>
```

```
            </ScrollViewer>
        </Grid>
    </ScrollViewer>
</Grid>
```

Code-behind

2. In the previous snippet, we declared a Click event listener and handler (Toggle_Custom) on the custom toggle button for touch inking (toggleButton). This handler simply toggles support for CoreInputDeviceTypes.Touch through the InputDeviceTypes property of the InkPresenter.

   We also specified an icon for the button using the SymbolIcon element and the {x:Bind} markup extension that binds it to a field defined in the code-behind file (TouchWritingIcon).

   The following snippet includes both the Click event handler and the definition of TouchWritingIcon.

C#

```
namespace Ink_Basic_InkToolbar
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a
Frame.
    /// </summary>
    public sealed partial class MainPage_AddCustomToggle : Page
    {
        Symbol TouchWritingIcon = (Symbol)0xED5F;

        public MainPage_AddCustomToggle()
        {
            this.InitializeComponent();
        }

        // Handler for the custom toggle button that enables touch inking.
        private void CustomToggle_Click(object sender, RoutedEventArgs e)
        {
            if (toggleButton.IsChecked == true)
            {
                inkCanvas.InkPresenter.InputDeviceTypes |=
CoreInputDeviceTypes.Touch;
            }
            else
            {
                inkCanvas.InkPresenter.InputDeviceTypes &=
~CoreInputDeviceTypes.Touch;
            }
        }
```

```
    }
}
```

Custom tool

You can create a custom tool button to invoke a non-pen tool that is defined by your app.

By default, an **InkPresenter** processes all input as either an ink stroke or an erase stroke. This includes input modified by a secondary hardware affordance such as a pen barrel button, a right mouse button, or similar. However, **InkPresenter** can be configured to leave specific input unprocessed, which can then be passed through to your app for custom processing.

In this example, we define a custom tool button that, when selected, causes subsequent strokes to be processed and rendered as a selection lasso (dashed line) instead of ink. All ink strokes within the bounds of the selection area are set to **Selected**.

> **Note**
>
> See Inking controls for both InkCanvas and InkToolbar UX guidelines. The following recommendation is relevant to this example:
>
> - If providing stroke selection, we recommend using the "EF20" icon from the "Segoe MLD2 Assets" font for the tool button, with a "Selection tool" tooltip.

XAML

1. First, we declare an **InkToolbarCustomToolButton** element (customToolButton) with a Click event listener that specifies the event handler (customToolButton_Click) where stroke selection is configured. (We've also added a set of buttons for copying, cutting, and pasting the stroke selection.)

2. We also add a Canvas element for drawing our selection stroke. Using a separate layer to draw the selection stroke ensures the **InkCanvas** and its content remain untouched.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
```

```
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel x:Name="HeaderPanel" Orientation="Horizontal" Grid.Row="0">
        <TextBlock x:Name="Header"
                   Text="Basic ink sample"
                   Style="{ThemeResource HeaderTextBlockStyle}"
                   Margin="10,0,0,0" />
    </StackPanel>
    <StackPanel x:Name="ToolPanel" Orientation="Horizontal" Grid.Row="1">
        <InkToolbar x:Name="inkToolbar"
                    VerticalAlignment="Top"
                    TargetInkCanvas="{x:Bind inkCanvas}">
            <InkToolbarCustomToolButton
                x:Name="customToolButton"
                Click="customToolButton_Click"
                ToolTipService.ToolTip="Selection tool">
                <SymbolIcon Symbol="{x:Bind SelectIcon}"/>
            </InkToolbarCustomToolButton>
        </InkToolbar>
        <Button x:Name="cutButton"
                Content="Cut"
                Click="cutButton_Click"
                Width="100"
                Margin="5,0,0,0"/>
        <Button x:Name="copyButton"
                Content="Copy"
                Click="copyButton_Click"
                Width="100"
                Margin="5,0,0,0"/>
        <Button x:Name="pasteButton"
                Content="Paste"
                Click="pasteButton_Click"
                Width="100"
                Margin="5,0,0,0"/>
    </StackPanel>
    <Grid Grid.Row="2" x:Name="outputGrid"
              Background="{ThemeResource
SystemControlBackgroundChromeWhiteBrush}"
              Height="Auto">
        <!-- Canvas for displaying selection UI. -->
        <Canvas x:Name="selectionCanvas"/>
        <!-- Canvas for displaying ink. -->
        <InkCanvas x:Name="inkCanvas" />
    </Grid>
</Grid>
```

Code-behind

2. We then handle the Click event for the **InkToolbarCustomToolButton** in the MainPage.xaml.cs code-behind file.

   This handler configures the **InkPresenter** to pass unprocessed input through to the app.

For a more detailed step through of this code: See the Pass-through input for advanced processing section of Pen interactions and Windows Ink in Windows apps.

We also specified an icon for the button using the SymbolIcon element and the {x:Bind} markup extension that binds it to a field defined in the code-behind file (SelectIcon).

The following snippet includes both the Click event handler and the definition of SelectIcon.

C#

```
namespace Ink_Basic_InkToolbar
{
    /// <summary>
    /// An empty page that can be used on its own or navigated to within a
Frame.
    /// </summary>
    public sealed partial class MainPage_AddCustomTool : Page
    {
        // Icon for custom selection tool button.
        Symbol SelectIcon = (Symbol)0xEF20;

        // Stroke selection tool.
        private Polyline lasso;
        // Stroke selection area.
        private Rect boundingRect;

        public MainPage_AddCustomTool()
        {
            this.InitializeComponent();

            // Listen for new ink or erase strokes to clean up selection UI.
            inkCanvas.InkPresenter.StrokeInput.StrokeStarted +=
                StrokeInput_StrokeStarted;
            inkCanvas.InkPresenter.StrokesErased +=
                InkPresenter_StrokesErased;
        }

        private void customToolButton_Click(object sender, RoutedEventArgs
e)
        {
            // By default, the InkPresenter processes input modified by
            // a secondary affordance (pen barrel button, right mouse
            // button, or similar) as ink.
            // To pass through modified input to the app for custom
processing
            // on the app UI thread instead of the background ink thread,
set
            // InputProcessingConfiguration.RightDragAction to
LeaveUnprocessed.
```

```
inkCanvas.InkPresenter.InputProcessingConfiguration.RightDragAction =
                InkInputRightDragAction.LeaveUnprocessed;

            // Listen for unprocessed pointer events from modified input.
            // The input is used to provide selection functionality.
            inkCanvas.InkPresenter.UnprocessedInput.PointerPressed +=
                UnprocessedInput_PointerPressed;
            inkCanvas.InkPresenter.UnprocessedInput.PointerMoved +=
                UnprocessedInput_PointerMoved;
            inkCanvas.InkPresenter.UnprocessedInput.PointerReleased +=
                UnprocessedInput_PointerReleased;
        }

        // Handle new ink or erase strokes to clean up selection UI.
        private void StrokeInput_StrokeStarted(
            InkStrokeInput sender, Windows.UI.Core.PointerEventArgs args)
        {
            ClearSelection();
        }

        private void InkPresenter_StrokesErased(
            InkPresenter sender, InkStrokesErasedEventArgs args)
        {
            ClearSelection();
        }

        private void cutButton_Click(object sender, RoutedEventArgs e)
        {

inkCanvas.InkPresenter.StrokeContainer.CopySelectedToClipboard();
            inkCanvas.InkPresenter.StrokeContainer.DeleteSelected();
            ClearSelection();
        }

        private void copyButton_Click(object sender, RoutedEventArgs e)
        {

inkCanvas.InkPresenter.StrokeContainer.CopySelectedToClipboard();
        }

        private void pasteButton_Click(object sender, RoutedEventArgs e)
        {
            if
(inkCanvas.InkPresenter.StrokeContainer.CanPasteFromClipboard())
            {
                inkCanvas.InkPresenter.StrokeContainer.PasteFromClipboard(
                    new Point(0, 0));
            }
            else
            {
                // Cannot paste from clipboard.
            }
        }
```

```
        // Clean up selection UI.
        private void ClearSelection()
        {
            var strokes =
inkCanvas.InkPresenter.StrokeContainer.GetStrokes();
            foreach (var stroke in strokes)
            {
                stroke.Selected = false;
            }
            ClearBoundingRect();
        }

        private void ClearBoundingRect()
        {
            if (selectionCanvas.Children.Any())
            {
                selectionCanvas.Children.Clear();
                boundingRect = Rect.Empty;
            }
        }

        // Handle unprocessed pointer events from modified input.
        // The input is used to provide selection functionality.
        // Selection UI is drawn on a canvas under the InkCanvas.
        private void UnprocessedInput_PointerPressed(
            InkUnprocessedInput sender, PointerEventArgs args)
        {
            // Initialize a selection lasso.
            lasso = new Polyline()
            {
                Stroke = new SolidColorBrush(Windows.UI.Colors.Blue),
                StrokeThickness = 1,
                StrokeDashArray = new DoubleCollection() { 5, 2 },
            };

            lasso.Points.Add(args.CurrentPoint.RawPosition);

            selectionCanvas.Children.Add(lasso);
        }

        private void UnprocessedInput_PointerMoved(
            InkUnprocessedInput sender, PointerEventArgs args)
        {
            // Add a point to the lasso Polyline object.
            lasso.Points.Add(args.CurrentPoint.RawPosition);
        }

        private void UnprocessedInput_PointerReleased(
            InkUnprocessedInput sender, PointerEventArgs args)
        {
            // Add the final point to the Polyline object and
            // select strokes within the lasso area.
            // Draw a bounding box on the selection canvas
            // around the selected ink strokes.
            lasso.Points.Add(args.CurrentPoint.RawPosition);
```

```
            boundingRect =
                inkCanvas.InkPresenter.StrokeContainer.SelectWithPolyLine(
                    lasso.Points);

            DrawBoundingRect();
        }

        // Draw a bounding rectangle, on the selection canvas, encompassing
        // all ink strokes within the lasso area.
        private void DrawBoundingRect()
        {
            // Clear all existing content from the selection canvas.
            selectionCanvas.Children.Clear();

            // Draw a bounding rectangle only if there are ink strokes
            // within the lasso area.
            if (!((boundingRect.Width == 0) ||
                (boundingRect.Height == 0) ||
                boundingRect.IsEmpty))
            {
                var rectangle = new Rectangle()
                {
                    Stroke = new SolidColorBrush(Windows.UI.Colors.Blue),
                    StrokeThickness = 1,
                    StrokeDashArray = new DoubleCollection() { 5, 2 },
                    Width = boundingRect.Width,
                    Height = boundingRect.Height
                };

                Canvas.SetLeft(rectangle, boundingRect.X);
                Canvas.SetTop(rectangle, boundingRect.Y);

                selectionCanvas.Children.Add(rectangle);
            }
        }
    }
}
```

Custom ink rendering

By default, ink input is processed on a low-latency background thread and rendered "wet" as it is drawn. When the stroke is completed (pen or finger lifted, or mouse button released), the stroke is processed on the UI thread and rendered "dry" to the **InkCanvas** layer (above the application content and replacing the wet ink).

The ink platform enables you to override this behavior and completely customize the inking experience by custom drying the ink input.

For more info on custom drying, see Pen interactions and Windows Ink in Windows apps.

> **Note**
>
> Custom drying and the **InkToolbar**
> If your app overrides the default ink rendering behavior of the **InkPresenter** with a custom drying implementation, the rendered ink strokes are no longer available to the InkToolbar and the built-in erase commands of the InkToolbar do not work as expected. To provide erase functionality, you must handle all pointer events, perform hit-testing on each stroke, and override the built-in "Erase all ink" command.

Related articles

- Pen and stylus interactions

Topic samples

- Ink toolbar location and orientation sample (basic)
- Ink toolbar location and orientation sample (dynamic)

Other samples

- Simple ink sample (C#/C++)
- Complex ink sample (C++)
- Ink sample (JavaScript)
- Get Started Tutorial: Support ink in your Windows app
- Coloring book sample
- Family notes sample

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

more information, see our contributor guide.

Provide product feedback

Touch interactions

Article • 01/19/2024

This topic provides design guidelines for building custom, touch-optimized experiences in Windows apps.

Overview

Touch is a primary form of input across Windows and Windows apps that involves the use of one or more fingers (or touch contacts). These touch contacts, and their movement, are interpreted as touch gestures and manipulations that support a variety of user interactions.

Both the Windows SDK and the Windows App SDK include comprehensive collections of touch-optimized controls that provide robust and consistent experiences across Windows apps.

Use these guidelines when creating custom controls, experiences, and frameworks for your Windows apps.

Design principles

Consider the following as you design the touch experience in your Windows app.

Touch Optimized

Windows app experiences should feel inviting to touch, allow for direct manipulation, and accommodate less-precise interactions. Consider touch accelerators including gestures and pen and voice integration.

Consistent Across Postures

Your app should have a consistent experience regardless of the input method or posture the user is in. Changes from traditional desktop posture to tablet posture (see Recommended settings for better tablet experiences), as well as changes in orientation, should not be disorienting, but rather subtle and only as necessary. Your app should reflow UI in subtle ways to create a familiar, cohesive experience that meets users where they are.

Responsive

Apps and interactions should provide users with feedback at every phase (Touch down, action, touch up) of an interaction using animations that respond to a user's existing state while indicating possible actions. Animations should also maintain at least 60 fps to feel smooth and modern.

Honoring Touch Conventions

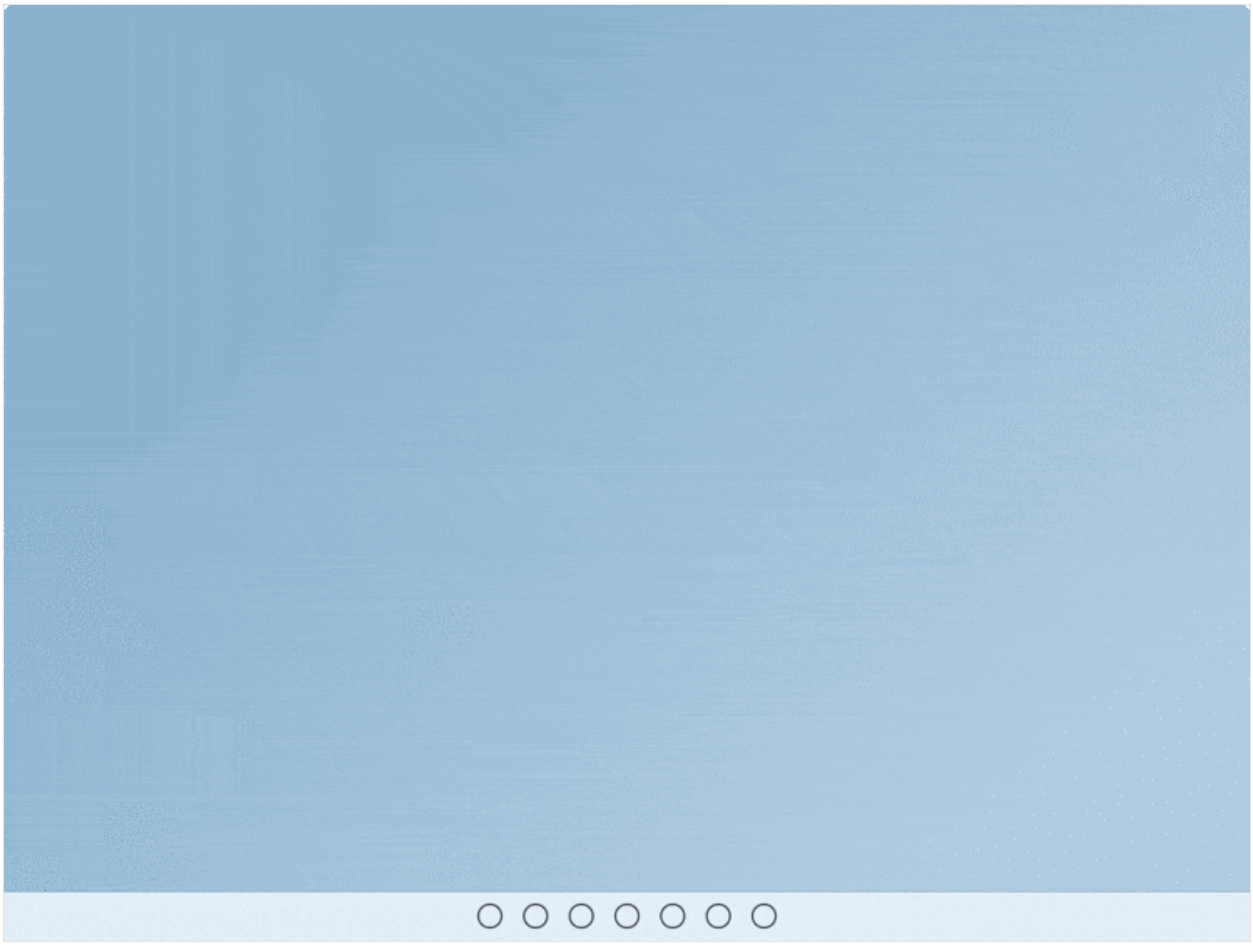
Responsive Feedback

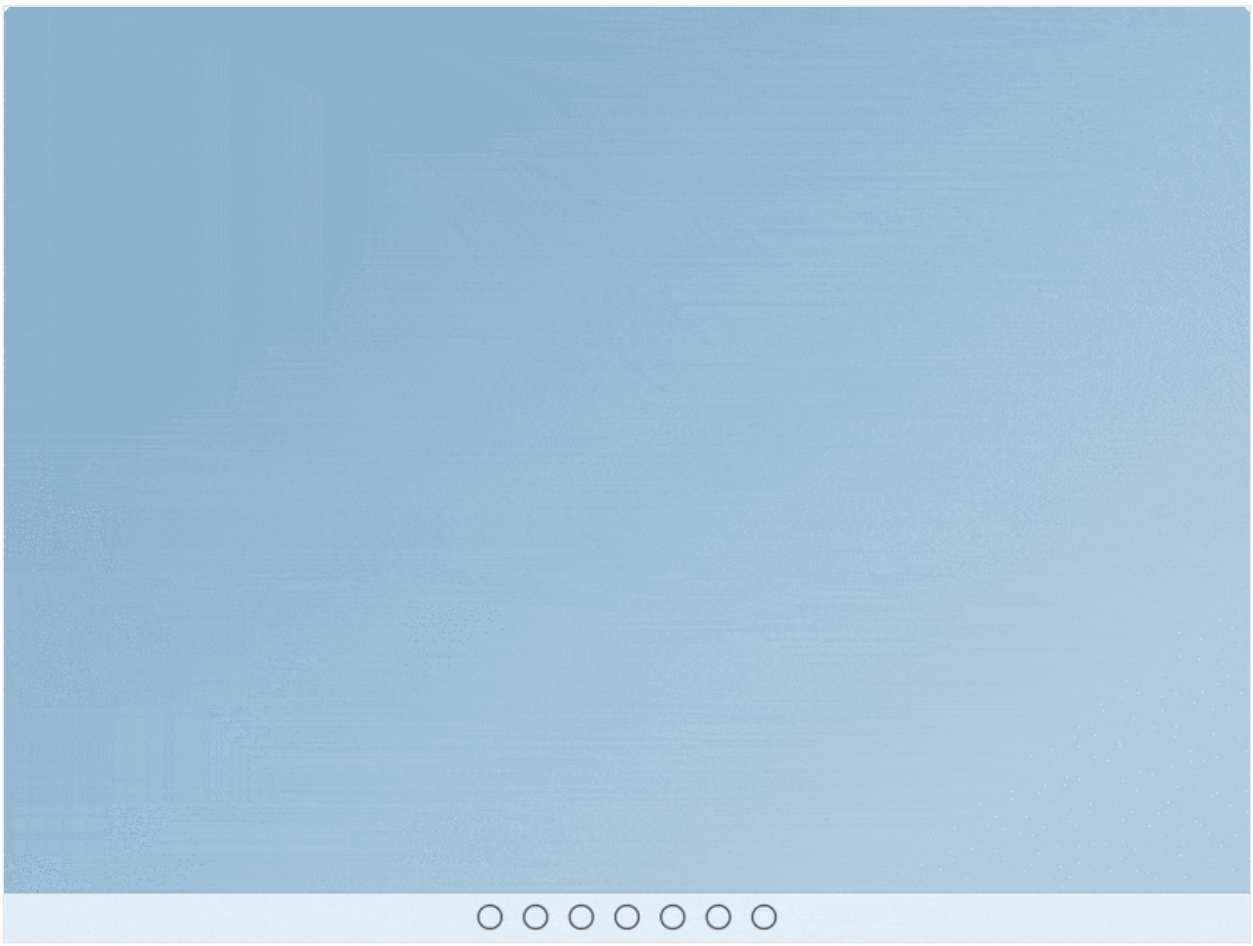
Appropriate visual feedback during interactions with your app helps users recognize, learn, and adapt to how their interactions are interpreted by both the app and the Windows platform. Provide immediate and continuous feedback in response to the user's touch, that is noticeable, comprehensible, and not lost by distractions. This immediate feedback is how users will learn and explore the interactive elements of your app.

- Feedback should be immediate on touch down and moving objects should stick to the user's finger.
- UI should respond to gestures by matching user speed and movements, avoid using keyframe animations.
- Visual feedback should convey possible outcomes before the user commits to an action.

Do

Don't





For more info, see Guidelines for visual feedback and Motion in Windows 11

Touch Interaction Patterns

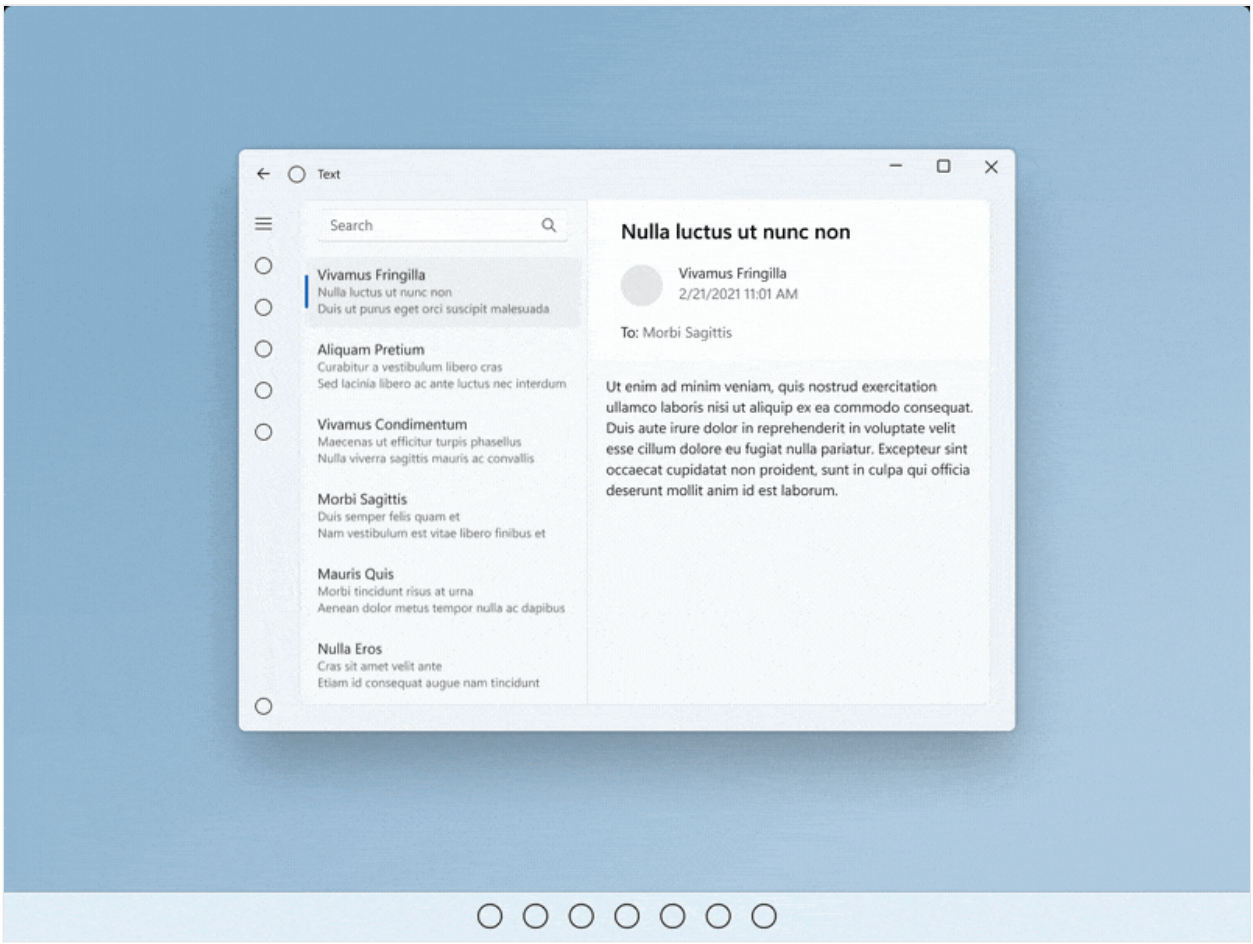
Honor these common interaction and gestures patterns to bring consistency and predictably to your experience.

Common Interactions

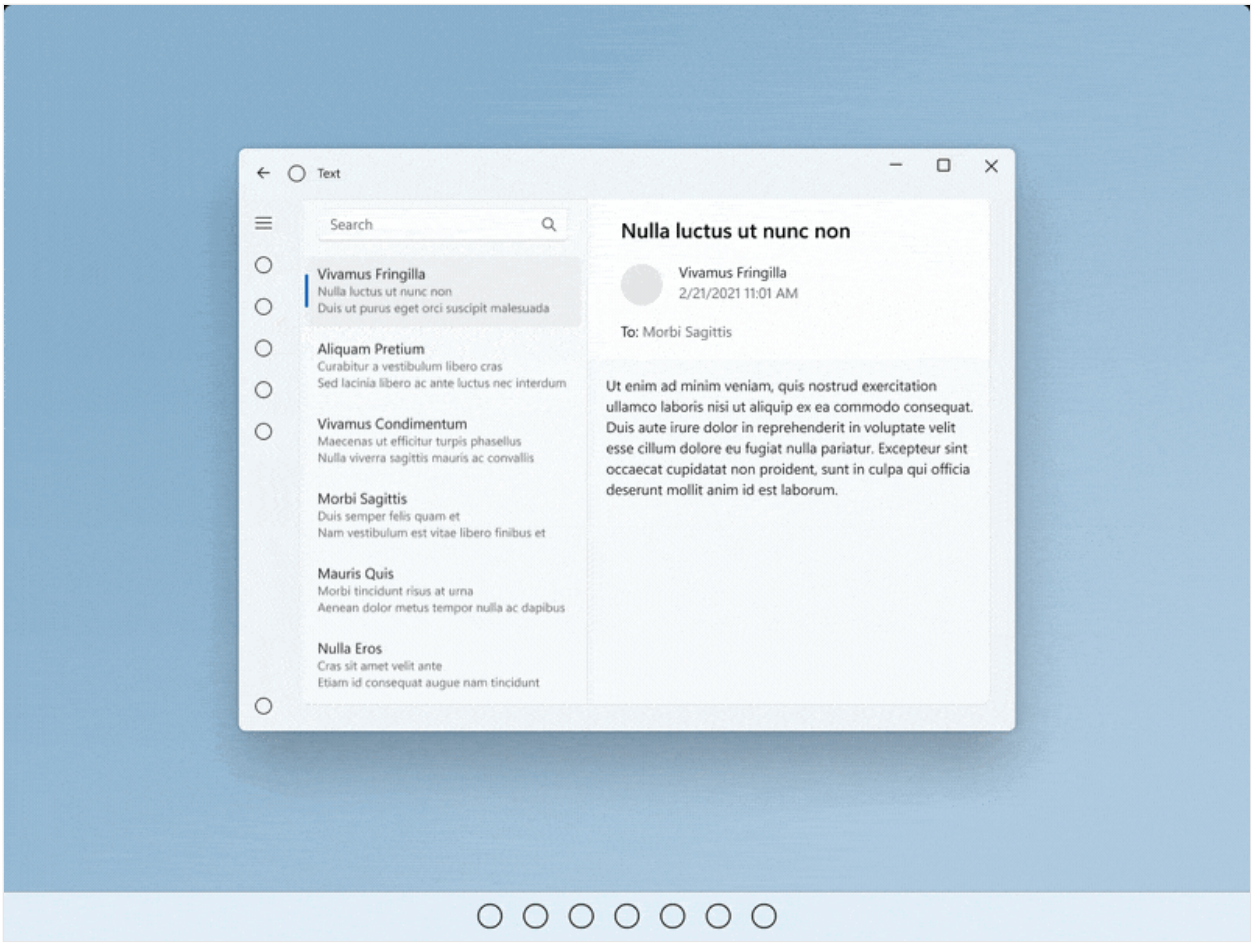
There are a set of common touch behaviors and gestures that users are familiar with and expect to work consistently across all Windows experiences.

- **Tap** to activate or select an item
- **Short press and drag** to move an object
- **Press and hold** to access a menu of secondary, contextual commands
- **Swipe (or drag and release)** for contextual commands
- **Rotate** clockwise or counterclockwise to pivot

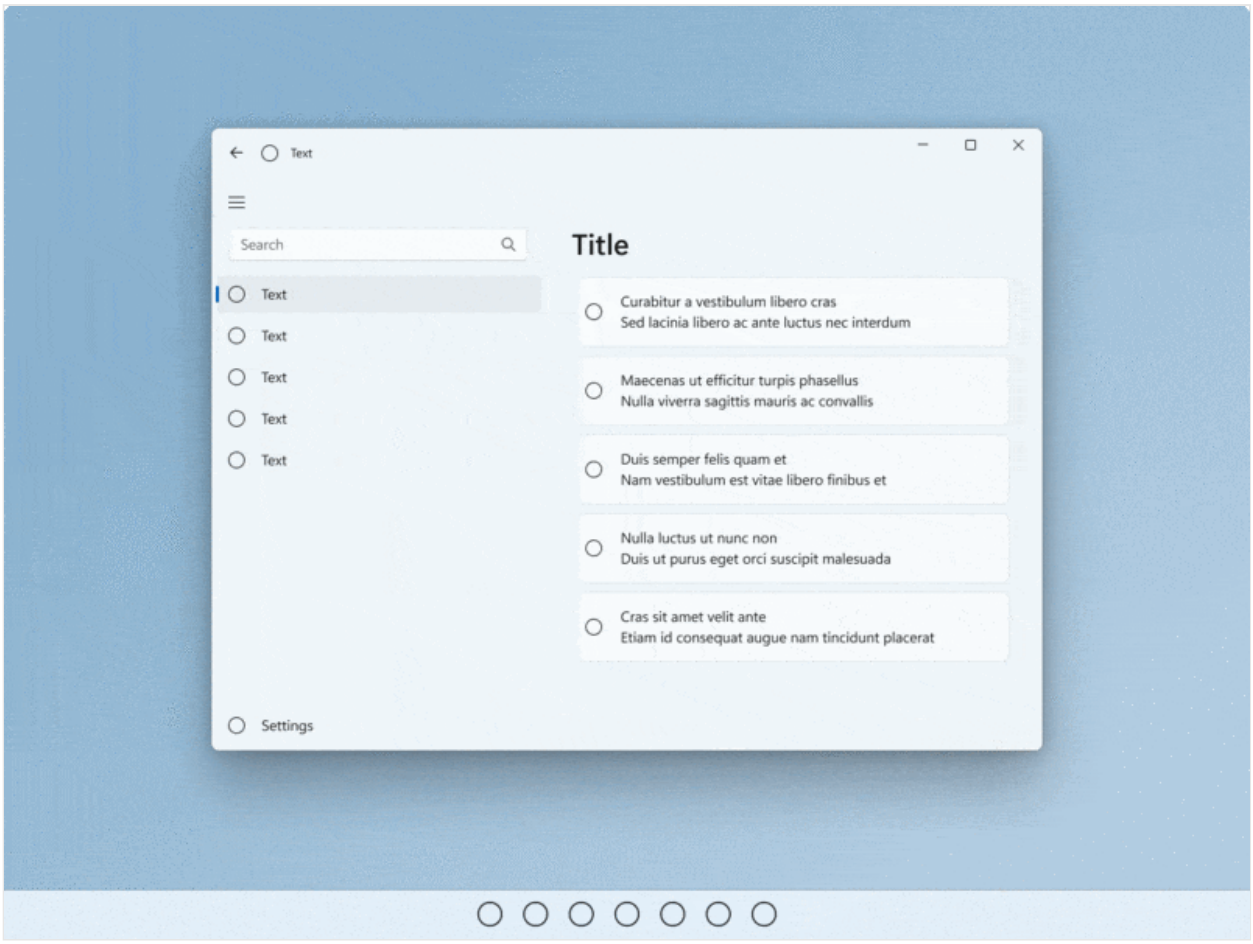
Interactions



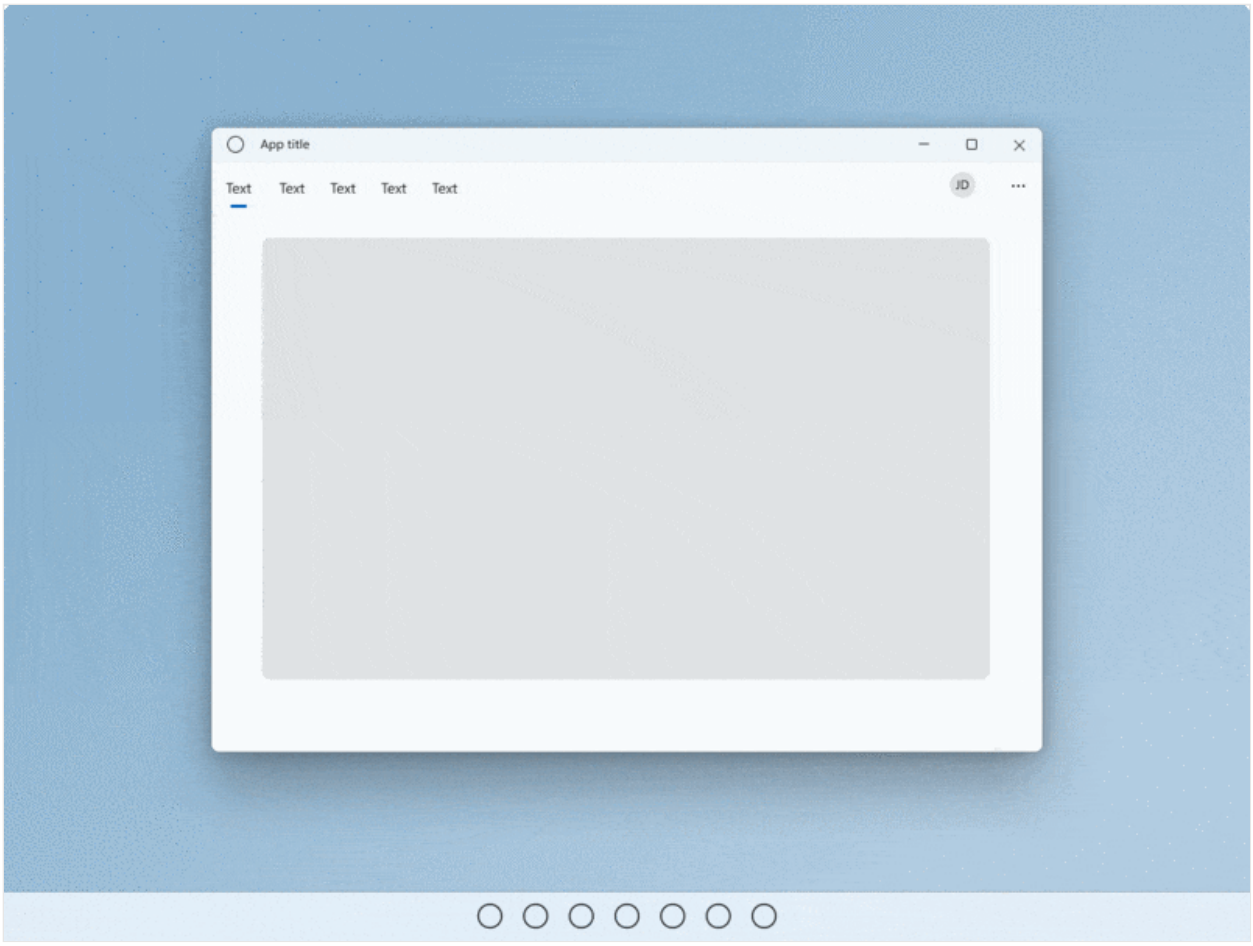
Tap



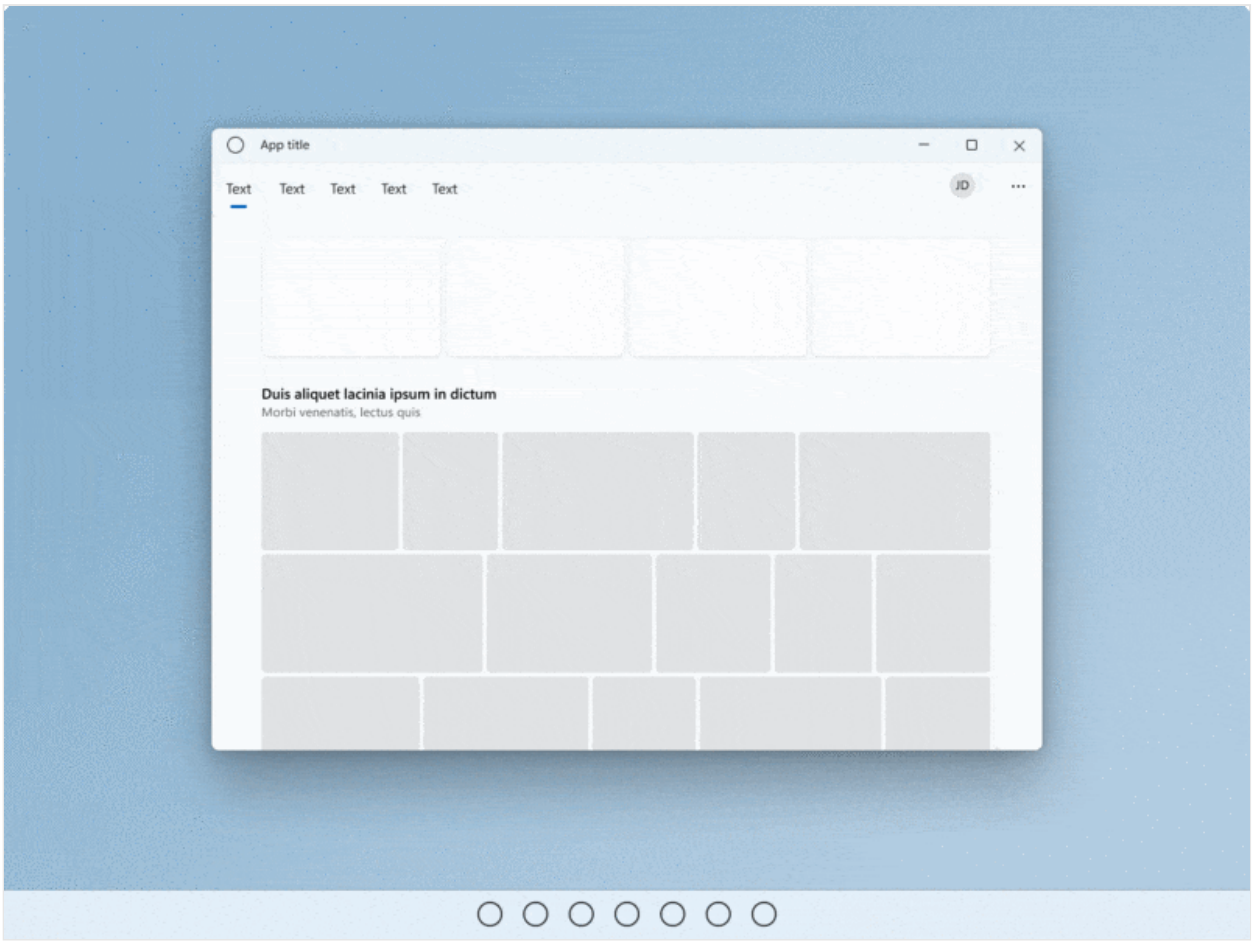
Swipe (or drag and release)



Short press and drag



Rotate



Press and hold

For more info, see Guidelines for visual feedback and Motion in Windows 11

Gestures

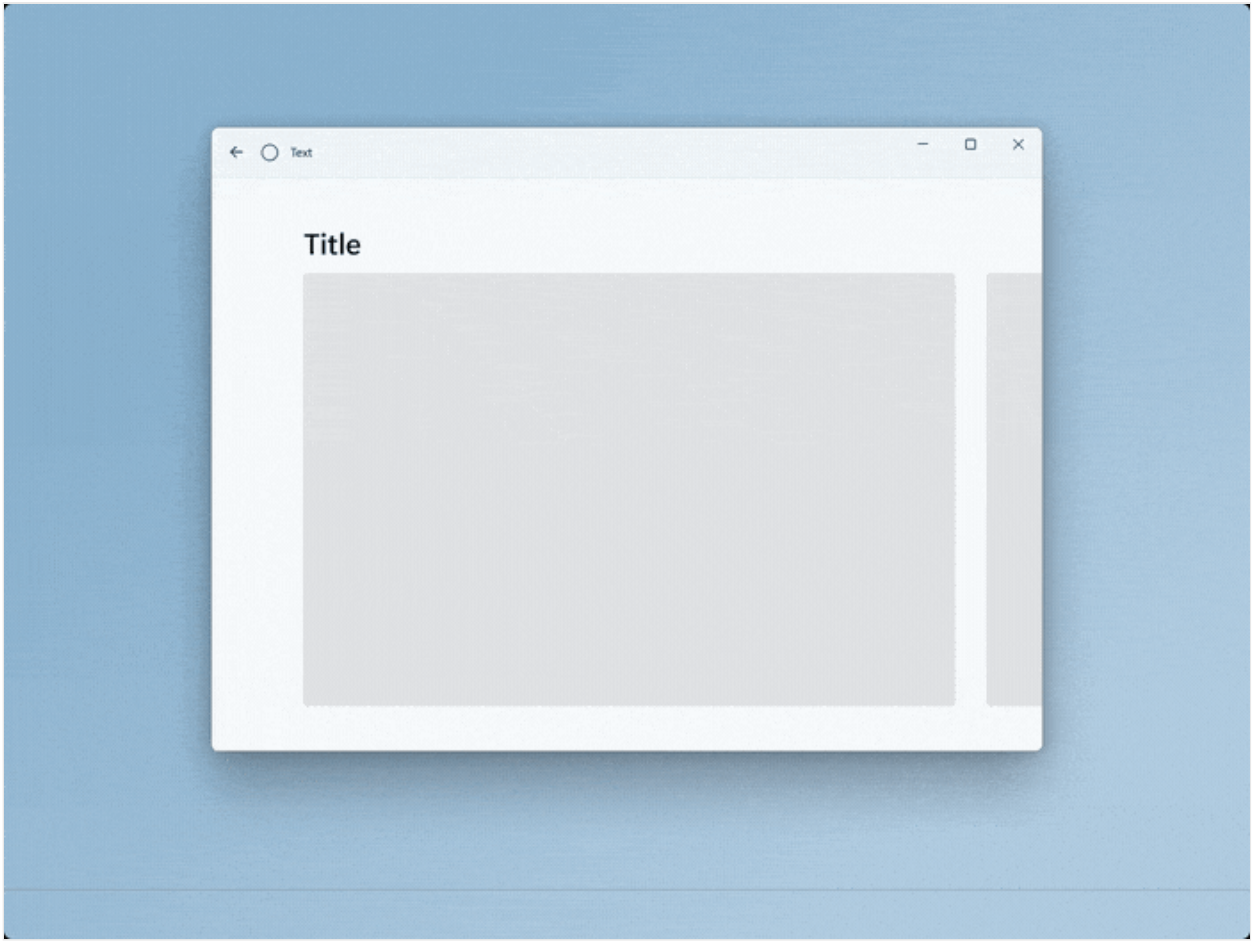
Gestures lower the effort required by users to navigate and act on common interactions. Where possible, support UI with touch gestures to make it easy for users to navigate and act in an app.

If navigating between views, use connected animations so existing and new states are both visible mid-drag. If interacting with UI, items should follow user movement, provide feedback and, on release, react with additional actions based on drag position thresholds.

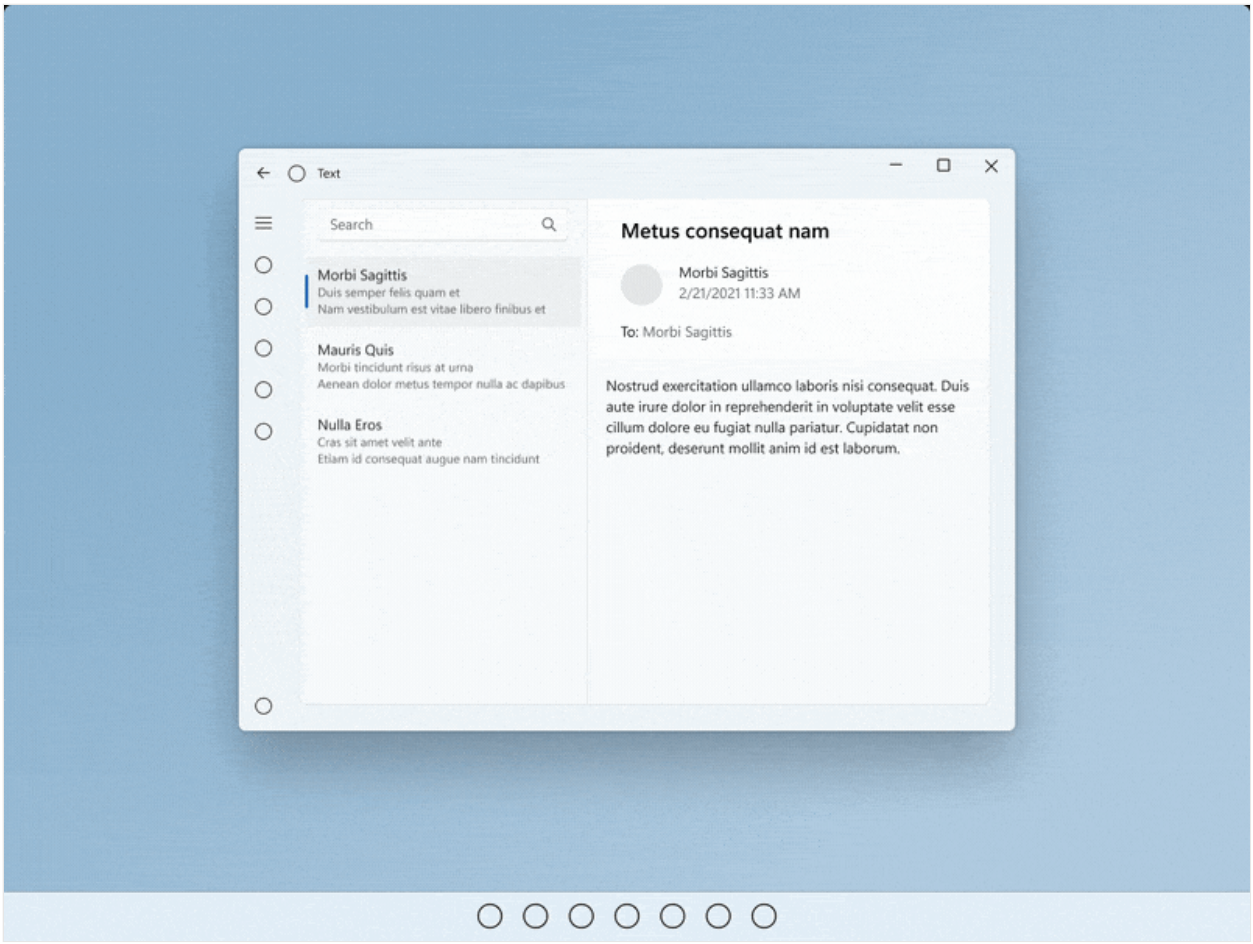
Gestures should also be actionable with flicks and swipes based on inertia and be within a comfortable range of motion.

- Drag or flick to go back and forth
- Drag to dismiss
- Pull to refresh

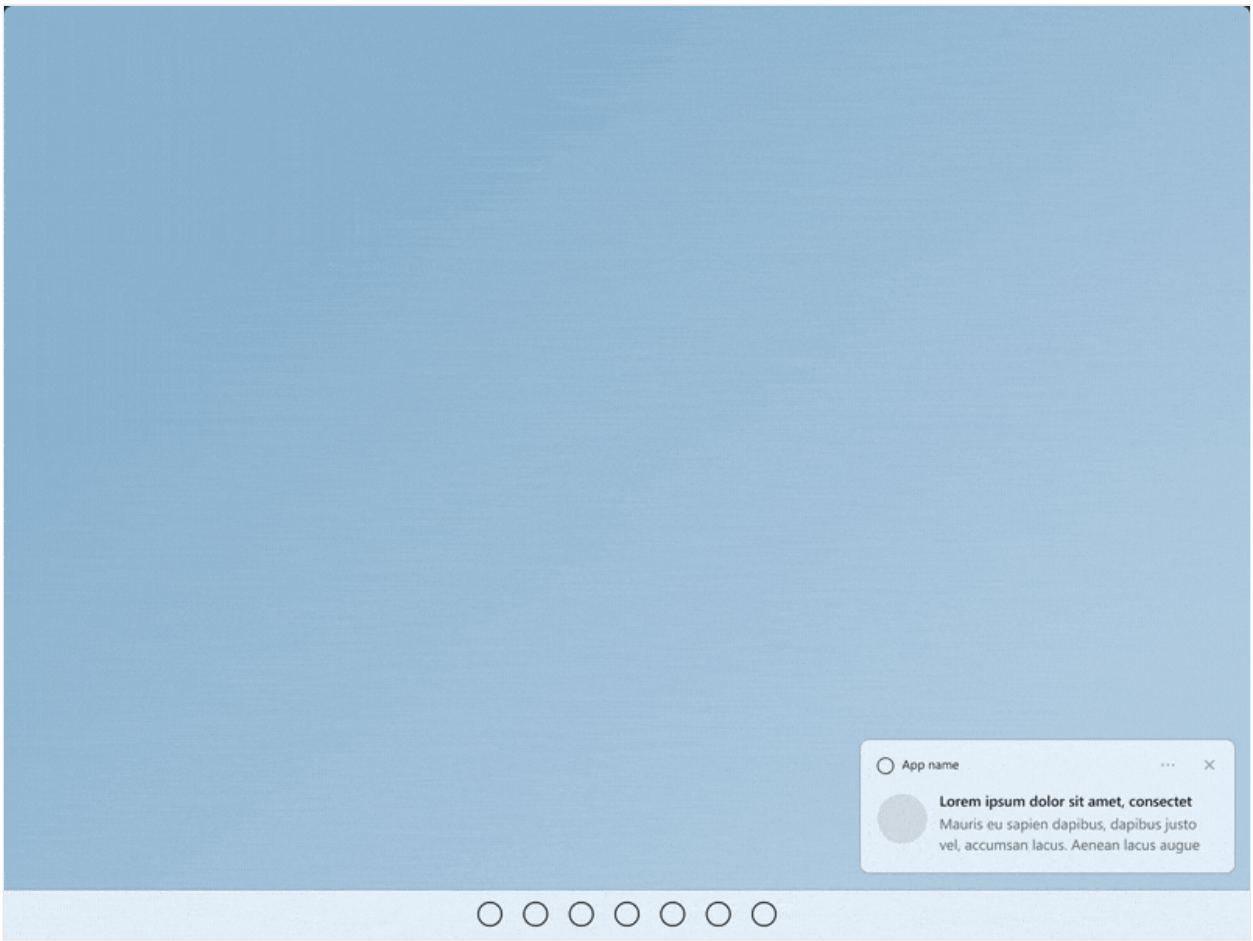
Gestures



Drag or flick to go back and forth



Pull to refresh



Drag to dismiss

For more info, see Page transitions and Pull to refresh.

Custom gestures

Use custom gestures to bring high frequency keyboard shortcut keys and trackpad gestures to a touch interaction. Aid discoverability and response through dedicated affordances with animations and visual states (for example, placing three fingers on screen causes windows to shrink for visual feedback).

- Do not override common gestures as this can cause confusion for users.
- Consider using multi-finger gestures for custom actions but be aware that the system has reserved some multi-finger gestures for rapid switching between apps and desktops.
- Be mindful of custom gestures originating near the edges of a screen as edge gestures are reserved for OS-level behaviors, which can be invoked accidentally.

Avoid accidental navigation

If you your app or game might involve frequent interactions near the edges of the screen, consider presenting your experience in a Fullscreen Exclusive (FSE) mode to avoid accidental activations of system flyouts (users will have to swipe directly on the temporary tab in order to pull in the associated system flyout).

> **Note**
>
> Avoid using this unless absolutely necessary as it will make it harder for users to navigate away from your app or use it in conjunction with others.

Touch keyboard experiences

The touch keyboard enables text entry for devices that support touch. Windows app text input controls invoke the touch keyboard by default when a user taps on an editable input field.

Invoke on text field tap

Touch keyboard should pop up when a user taps on a text input field – this will work automatically by using our system APIs to show and hide the keyboard. See Respond to the presence of the touch keyboard.

Use standard text input controls

Using common controls provides expected behavior and minimizes surprises for users.

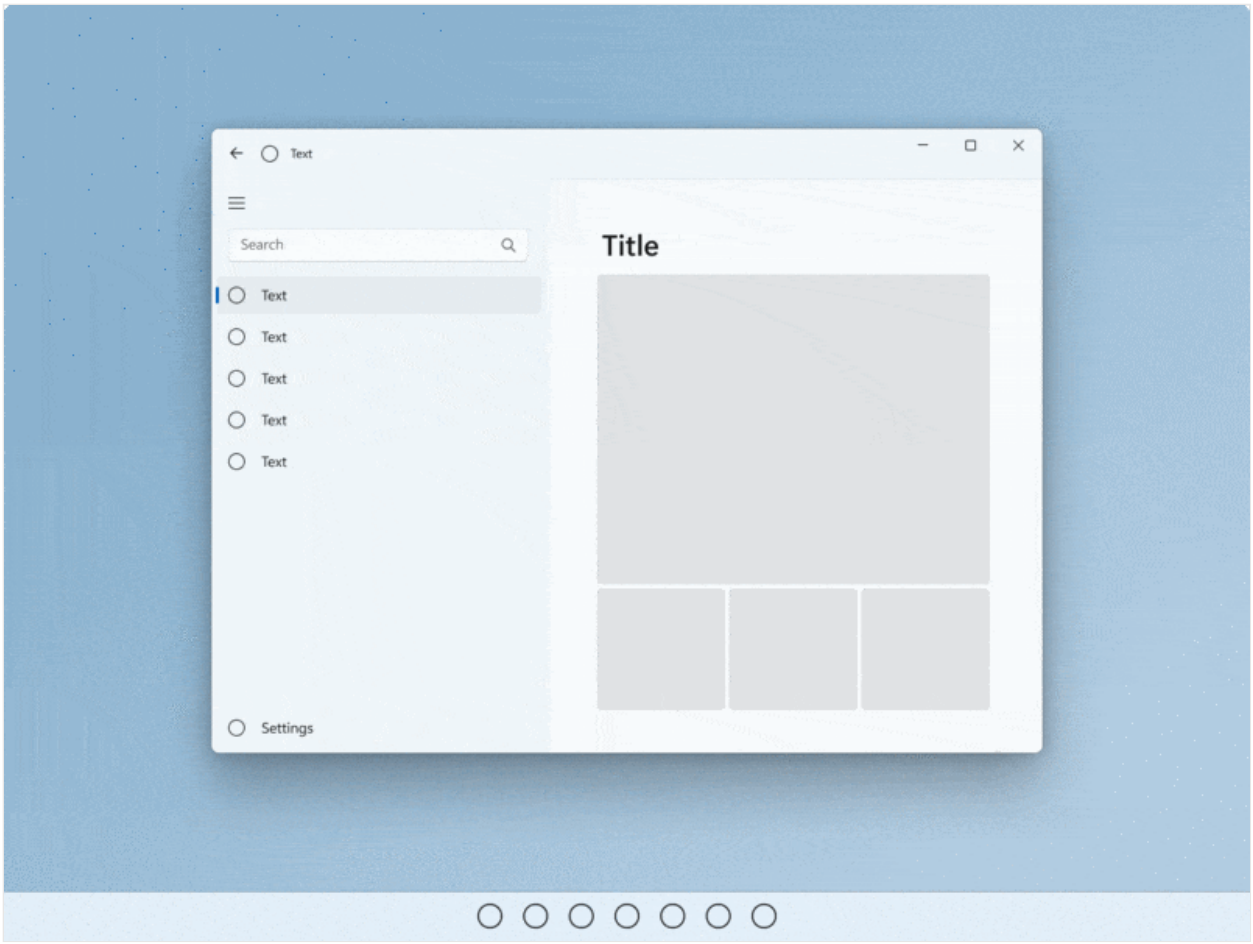
Text controls that support the Text Services Framework (TSF) provide shape-writing (swipe keyboard) capabilities.

Touch keyboard signals

Account for input, posture, hardware signals that make touch keyboard the main mode of input (hardware keyboard is detached, entrypoints are invoked with touch, clear user intent to type).

Reflow appropriately

- Be aware that the keyboard can take up 50% of the screen on smaller devices.
- Don't obscure the active text field with the touch keyboard.
- If the touch keyboard is obscuring the active text field, scroll the app content up (with animation) until the field is visible.
- If the touch keyboard is obscuring the active text field but the app content cannot scroll up, try to move the app container (with animation).



Hit Targets

Make sure hit targets are comfortable and inviting to touch. If hit targets are too small or crowded, users must be more precise, which is difficult with touch and can result in a poor experience.

Touchable

We define touchable as a minimum of 40 x 40 epx, even if the visual is smaller, or 32 epx tall if the width is at least 120 epx.

Our common controls conform to this standard (they are optimized for both mouse and touch users).

Touch-optimized

For a touch-optimized UI, consider increasing the target size to 44 x 44 epx with at least 4 epx of visible space between targets.

We recommend two default behaviors: Always touch optimized or transition based on device signals.

When an app can be optimized for touch without compromising mouse users, especially if the app is used primarily with touch, then always touch optimize.

If you transition the UI based on device signals for device posture, always provide consistent experiences across postures.

Match visuals to touch target

Consider updating visuals when touch target dimensions change. For example, if hit targets increase when users enter tablet posture, UI representing the hit targets should also update to help users understand the state change and updated affordance. For more info, see Content design basics for Windows apps, Guidelines for touch targets, Control size and density.

Portrait Optimization

Support responsive layouts that account for both tall and wide windows to ensure an app is optimized for both landscape and portrait orientations.

This will also ensure app windows display core UI visuals properly in multitasking scenarios (apps snapped side-by-side with portrait aspect ratios) regardless of orientation and screen sizes.

Touch interactions developer guide

Article • 01/19/2024

Design your app with the expectation that touch will be the primary input method of your users. If you use UWP controls, support for touchpad, mouse, and pen/stylus requires no additional programming, because UWP apps provide this for free.

However, keep in mind that a UI optimized for touch is not always superior to a traditional UI. Both provide advantages and disadvantages that are unique to a technology and application. In the move to a touch-first UI, it is important to understand the core differences between touch, touchpad, pen/stylus, mouse, and keyboard input.

> **Important APIs**: Windows.UI.Xaml.Input, Windows.UI.Core, Windows.Devices.Input

Many devices have multi-touch screens that support using one or more fingers (or touch contacts) as input. The touch contacts, and their movement, are interpreted as touch gestures and manipulations to support various user interactions.

The Windows app includes a number of different mechanisms for handling touch input, enabling you to create an immersive experience that your users can explore with confidence. Here, we cover the basics of using touch input in a Windows app.

Touch interactions require three things:

- A touch-sensitive display.
- The direct contact (or proximity to, if the display has proximity sensors and supports hover detection) of one or more fingers on that display.
- Movement of the touch contacts (or lack thereof, based on a time threshold).

The input data provided by the touch sensor can be:

- Interpreted as a physical gesture for direct manipulation of one or more UI elements, such as panning, rotating, resizing, or moving. (In contrast, interacting with an element through its properties window, dialog box, or other UI affordance is considered indirect manipulation.)
- Recognized as an alternative input method, such as mouse or pen.
- Used to complement or modify aspects of other input methods, such as smudging an ink stroke drawn with a pen.

Touch input typically involves the direct manipulation of an element on the screen. The element responds immediately to any touch contact within its hit test area, and reacts

appropriately to any subsequent movement of the touch contacts, including removal.

Custom touch gestures and interactions should be designed carefully. They should be intuitive, responsive, and discoverable, and they should let users explore your app with confidence.

Ensure that app functionality is exposed consistently across every supported input device type. If necessary, use some form of indirect input mode, such as text input for keyboard interactions, or UI affordances for mouse and pen.

Remember that traditional input devices (such as mouse and keyboard) are familiar and appealing to many users. They can offer speed, accuracy, and tactile feedback that touch cannot.

Providing unique and distinctive interaction experiences for all input devices will support the widest range of capabilities and preferences, appeal to the broadest possible audience, and attract more customers to your app.

Compare touch interaction requirements

The following table shows some of the differences between input devices that you should consider when you design touch-optimized Windows apps.

Expand table

Factor	Touch interactions	Mouse, keyboard, pen/stylus interactions	Touchpad
Precision	The contact area of a fingertip is greater than a single x-y coordinate, which increases the chances of unintended command activations.	The mouse and pen/stylus supply a precise x-y coordinate.	Same as mouse.
	The shape of the contact area changes throughout the movement.	Mouse movements and pen/stylus strokes supply precise x-y coordinates. Keyboard focus is explicit.	Same as mouse.
	There is no mouse cursor to assist with targeting.	The mouse cursor, pen/stylus cursor, and keyboard focus all assist with targeting.	Same as mouse.
Human anatomy	Fingertip movements are imprecise, because a straight-line motion with one or more fingers	It's easier to perform a straight-line motion with the mouse or pen/stylus because	Same as mouse.

	is difficult. This is due to the curvature of hand joints and the number of joints involved in the motion.	the hand that controls them travels a shorter physical distance than the cursor on the screen.	
	Some areas on the touch surface of a display device can be difficult to reach due to finger posture and the user's grip on the device.	The mouse and pen/stylus can reach any part of the screen while any control should be accessible by the keyboard through tab order.	Finger posture and grip can be an issue.
	Objects might be obscured by one or more fingertips or the user's hand. This is known as occlusion.	Indirect input devices do not cause occlusion.	Same as mouse.
Object state	Touch uses a two-state model: the touch surface of a display device is either touched (on) or not (off). There is no hover state that can trigger additional visual feedback.	A mouse, pen/stylus, and keyboard all expose a three-state model: up (off), down (on), and hover (focus). Hover lets users explore and learn through tooltips associated with UI elements. Hover and focus effects can relay which objects are interactive and also help with targeting.	Same as mouse.
Rich interaction	Supports multi-touch: multiple input points (fingertips) on a touch surface.	Supports a single input point.	Same as touch.
	Supports direct manipulation of objects through gestures such as tapping, dragging, sliding, pinching, and rotating.	No support for direct manipulation as mouse, pen/stylus, and keyboard are indirect input devices.	Same as mouse.

Note

Indirect input has had the benefit of more than 25 years of refinement. Features such as hover-triggered tooltips have been designed to solve UI exploration specifically for touchpad, mouse, pen/stylus, and keyboard input. UI features like this have been re-designed for the rich experience provided by touch input, without compromising the user experience for these other devices.

Use touch feedback

Appropriate visual feedback during interactions with your app helps users recognize, learn, and adapt to how their interactions are interpreted by both the app and the Windows platform. Visual feedback can indicate successful interactions, relay system status, improve the sense of control, reduce errors, help users understand the system and input device, and encourage interaction.

Visual feedback is critical when the user relies on touch input for activities that require accuracy and precision based on location. Display feedback whenever and wherever touch input is detected, to help the user understand any custom targeting rules that are defined by your app and its controls.

Targeting

Targeting is optimized through:

- Touch target sizes

  Clear size guidelines ensure that applications provide a comfortable UI that contains objects and controls that are easy and safe to target.

- Contact geometry

  The entire contact area of the finger determines the most likely target object.

- Scrubbing

  Items within a group are easily re-targeted by dragging the finger between them (for example, radio buttons). The current item is activated when the touch is released.

- Rocking

  Densely packed items (for example, hyperlinks) are easily re-targeted by pressing the finger down and, without sliding, rocking it back and forth over the items. Due to occlusion, the current item is identified through a tooltip or the status bar and is activated when the touch is released.

Accuracy

Design for sloppy interactions by using:

- Snap-points that can make it easier to stop at desired locations when users interact with content.
- Directional "rails" that can assist with vertical or horizontal panning, even when the hand moves in a slight arc. For more information, see Guidelines for panning.

Occlusion

Finger and hand occlusion is avoided through:

- Size and positioning of UI

  Make UI elements big enough so that they cannot be completely covered by a fingertip contact area.

  Position menus and pop-ups above the contact area whenever possible.

- Tooltips

  Show tooltips when a user maintains finger contact on an object. This is useful for describing object functionality. The user can drag the fingertip off the object to avoid invoking the tooltip.

  For small objects, offset tooltips so they are not covered by the fingertip contact area. This is helpful for targeting.

- Handles for precision

  Where precision is required (for example, text selection), provide selection handles that are offset to improve accuracy. For more information, see Guidelines for selecting text and images (Windows Runtime apps).

Timing

Avoid timed mode changes in favor of direct manipulation. Direct manipulation simulates the direct, real-time physical handling of an object. The object responds as the fingers are moved.

A timed interaction, on the other hand, occurs after a touch interaction. Timed interactions typically depend on invisible thresholds like time, distance, or speed to determine what command to perform. Timed interactions have no visual feedback until the system performs the action.

Direct manipulation provides a number of benefits over timed interactions:

- Instant visual feedback during interactions make users feel more engaged, confident, and in control.
- Direct manipulations make it safer to explore a system because they are reversible—users can easily step back through their actions in a logical and intuitive manner.
- Interactions that directly affect objects and mimic real world interactions are more intuitive, discoverable, and memorable. They don't rely on obscure or abstract interactions.
- Timed interactions can be difficult to perform, as users must reach arbitrary and invisible thresholds.

In addition, the following are strongly recommended:

- Manipulations should not be distinguished by the number of fingers used.
- Interactions should support compound manipulations. For example, pinch to zoom while dragging the fingers to pan.
- Interactions should not be distinguished by time. The same interaction should have the same outcome regardless of the time taken to perform it. Time-based activations introduce mandatory delays for users and detract from both the immersive nature of direct manipulation and the perception of system responsiveness.

  > **Note**
  >
  > An exception to this is where you use specific timed interactions to assist in learning and exploration (for example, press and hold).

- Appropriate descriptions and visual cues have a great effect on the use of advanced interactions.

App views

Tweak the user interaction experience through the pan/scroll and zoom settings of your app views. An app view dictates how a user accesses and manipulates your app and its content. Views also provide behaviors such as inertia, content boundary bounce, and snap points.

Pan and scroll settings of the ScrollViewer control dictate how users navigate within a single view, when the content of the view doesn't fit within the viewport. A single view, for example, can be a page of a magazine or book, the folder structure of a computer, a library of documents, or a photo album.

Zoom settings apply to both optical zoom (supported by the **ScrollViewer** control) and the **Semantic Zoom** control. Semantic Zoom is a touch-optimized technique for presenting and navigating large sets of related data or content within a single view. It works by using two distinct modes of classification, or zoom levels. This is analogous to panning and scrolling within a single view. Panning and scrolling can be used in conjunction with Semantic Zoom.

Use app views and events to modify the pan/scroll and zoom behaviors. This can provide a smoother interaction experience than is possible through the handling of pointer and gesture events.

For more info about app views, see Controls, layouts, and text.

Custom touch interactions

If you implement your own interaction support, keep in mind that users expect an intuitive experience involving direct interaction with the UI elements in your app. We recommend that you model your custom interactions on the platform control libraries to keep things consistent and discoverable. The controls in these libraries provide the full user interaction experience, including standard interactions, animated physics effects, visual feedback, and accessibility. Create custom interactions only if there is a clear, well-defined requirement and basic interactions don't support your scenario.

> ⓘ **Important**
>
> **Windows 11 and newer**
>
> *Some three- and four-finger touch interactions will no longer work in Windows apps by default.*
>
> By default, three- and four-finger touch interactions are now consumed by the system for operations such as switching or minimizing windows and changing virtual desktops. As these interactions are now handled at the system level, your app's functionality could be affected by this change.
>
> To support three- or four-finger interactions within an application, a new user setting has been introduced that specifies whether or not the system handles these interactions:
>
> **Bluetooth & devices > Touch > "Three- and four-finger touch gestures"**
>
> When set to "On" (default), the system will handle all three- and four-finger interactions (apps will not be able to support them).

When set to "Off", three- and four-finger interactions can be supported by apps (they will not be handled by the system).

If your application must support these interactions, we recommend that you inform users of this setting and provide a link that launches the Settings app to the relevant page (ms-settings:devices-touch). For more details, see **Launch the Windows Settings app**.

To provide customized touch support, you can handle various **UIElement** events. These events are grouped into three levels of abstraction.

- Static gesture events are triggered after an interaction is complete. Gesture events include **Tapped**, **DoubleTapped**, **RightTapped**, and **Holding**.

  You can disable gesture events on specific elements by setting **IsTapEnabled**, **IsDoubleTapEnabled**, **IsRightTapEnabled**, and **IsHoldingEnabled** to **false**.

- Pointer events such as **PointerPressed** and **PointerMoved** provide low-level details for each touch contact, including pointer motion and the ability to distinguish press and release events.

  A pointer is a generic input type with a unified event mechanism. It exposes basic info, such as screen position, on the active input source, which can be touch, touchpad, mouse, or pen.

- Manipulation gesture events, such as **ManipulationStarted**, indicate an ongoing interaction. They start firing when the user touches an element and continue until the user lifts their finger(s), or the manipulation is canceled.

  Manipulation events include multi-touch interactions such as zooming, panning, or rotating, and interactions that use inertia and velocity data such as dragging. The information provided by the manipulation events doesn't identify the form of the interaction that was performed, but rather includes data such as position, translation delta, and velocity. You can use this touch data to determine the type of interaction that should be performed.

Here is the basic set of touch gestures supported by UWP.

Expand table

Name	Type	Description
Tap	Static gesture	One finger touches the screen and lifts up.

Name	Type	Description
Press and hold	Static gesture	One finger touches the screen and stays in place.
Slide	Manipulation gesture	One or more fingers touch the screen and move in the same direction.
Swipe	Manipulation gesture	One or more fingers touch the screen and move a short distance in the same direction.
Turn	Manipulation gesture	Two or more fingers touch the screen and move in a clockwise or counter-clockwise arc.
Pinch	Manipulation gesture	Two or more fingers touch the screen and move closer together.
Stretch	Manipulation gesture	Two or more fingers touch the screen and move farther apart.

Gesture events

For details about individual controls, see Controls list.

Pointer events

Pointer events are raised by a variety of active input sources, including touch, touchpad, pen, and mouse (they replace traditional mouse events.)

Pointer events are based on a single input point (finger, pen tip, mouse cursor) and do not support velocity-based interactions.

Here is a list of pointer events and their related event argument.

Expand table

Event or class	Description
PointerPressed	Occurs when a single finger touches the screen.
PointerReleased	Occurs when that same touch contact is lifted.
PointerMoved	Occurs when the pointer is dragged across the screen.
PointerEntered	Occurs when a pointer enters the hit test area of an element.
PointerExited	Occurs when a pointer exits the hit test area of an element.

Event or class	Description
PointerCanceled	Occurs when a touch contact is abnormally lost.
PointerCaptureLost	Occurs when a pointer capture is taken by another element.
PointerWheelChanged	Occurs when the delta value of a mouse wheel changes and when the touchpad is pinched.
PointerRoutedEventArgs	Provides data for all pointer events.

The following example shows how to use the PointerPressed, PointerReleased, and PointerExited events to handle a tap interaction on a Rectangle object.

First, a Rectangle named `touchRectangle` is created in Extensible Application Markup Language (XAML).

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Rectangle Name="touchRectangle"
           Height="100" Width="200" Fill="Blue" />
</Grid>
```

Next, listeners for the PointerPressed, PointerReleased, and PointerExited events are specified.

C++

```
MainPage::MainPage()
{
    InitializeComponent();

    // Pointer event listeners.
    touchRectangle->PointerPressed += ref new PointerEventHandler(this,
&MainPage::touchRectangle_PointerPressed);
    touchRectangle->PointerReleased += ref new PointerEventHandler(this,
&MainPage::touchRectangle_PointerReleased);
    touchRectangle->PointerExited += ref new PointerEventHandler(this,
&MainPage::touchRectangle_PointerExited);
}
```

cs

```
public MainPage()
{
    this.InitializeComponent();

    // Pointer event listeners.
```

```
    touchRectangle.PointerPressed += touchRectangle_PointerPressed;
    touchRectangle.PointerReleased += touchRectangle_PointerReleased;
    touchRectangle.PointerExited += touchRectangle_PointerExited;
}
```

VB

```
Public Sub New()

    ' This call is required by the designer.
    InitializeComponent()

    ' Pointer event listeners.
    AddHandler touchRectangle.PointerPressed, AddressOf
touchRectangle_PointerPressed
    AddHandler touchRectangle.PointerReleased, AddressOf
Me.touchRectangle_PointerReleased
    AddHandler touchRectangle.PointerExited, AddressOf
touchRectangle_PointerExited

End Sub
```

Finally, the **PointerPressed** event handler increases the **Height** and **Width** of the **Rectangle**, while the **PointerReleased** and **PointerExited** event handlers set the **Height** and **Width** back to their starting values.

C++

```
// Handler for pointer exited event.
void MainPage::touchRectangle_PointerExited(Object^ sender,
PointerRoutedEventArgs^ e)
{
    Rectangle^ rect = (Rectangle^)sender;

    // Pointer moved outside Rectangle hit test area.
    // Reset the dimensions of the Rectangle.
    if (nullptr != rect)
    {
        rect->Width = 200;
        rect->Height = 100;
    }
}

// Handler for pointer released event.
void MainPage::touchRectangle_PointerReleased(Object^ sender,
PointerRoutedEventArgs^ e)
{
    Rectangle^ rect = (Rectangle^)sender;

    // Reset the dimensions of the Rectangle.
    if (nullptr != rect)
```

```
    {
        rect->Width = 200;
        rect->Height = 100;
    }
}

// Handler for pointer pressed event.
void MainPage::touchRectangle_PointerPressed(Object^ sender,
PointerRoutedEventArgs^ e)
{
    Rectangle^ rect = (Rectangle^)sender;

    // Change the dimensions of the Rectangle.
    if (nullptr != rect)
    {
        rect->Width = 250;
        rect->Height = 150;
    }
}
```

cs

```
// Handler for pointer exited event.
private void touchRectangle_PointerExited(object sender,
PointerRoutedEventArgs e)
{
    Rectangle rect = sender as Rectangle;

    // Pointer moved outside Rectangle hit test area.
    // Reset the dimensions of the Rectangle.
    if (null != rect)
    {
        rect.Width = 200;
        rect.Height = 100;
    }
}
// Handler for pointer released event.
private void touchRectangle_PointerReleased(object sender,
PointerRoutedEventArgs e)
{
    Rectangle rect = sender as Rectangle;

    // Reset the dimensions of the Rectangle.
    if (null != rect)
    {
        rect.Width = 200;
        rect.Height = 100;
    }
}

// Handler for pointer pressed event.
private void touchRectangle_PointerPressed(object sender,
PointerRoutedEventArgs e)
```

```
{
    Rectangle rect = sender as Rectangle;

    // Change the dimensions of the Rectangle.
    if (null != rect)
    {
        rect.Width = 250;
        rect.Height = 150;
    }
}
```

VB

```
' Handler for pointer exited event.
Private Sub touchRectangle_PointerExited(sender As Object, e As
PointerRoutedEventArgs)
    Dim rect As Rectangle = CType(sender, Rectangle)

    ' Pointer moved outside Rectangle hit test area.
    ' Reset the dimensions of the Rectangle.
    If (rect IsNot Nothing) Then
        rect.Width = 200
        rect.Height = 100
    End If
End Sub

' Handler for pointer released event.
Private Sub touchRectangle_PointerReleased(sender As Object, e As
PointerRoutedEventArgs)
    Dim rect As Rectangle = CType(sender, Rectangle)

    ' Reset the dimensions of the Rectangle.
    If (rect IsNot Nothing) Then
        rect.Width = 200
        rect.Height = 100
    End If
End Sub

' Handler for pointer pressed event.
Private Sub touchRectangle_PointerPressed(sender As Object, e As
PointerRoutedEventArgs)
    Dim rect As Rectangle = CType(sender, Rectangle)

    ' Change the dimensions of the Rectangle.
    If (rect IsNot Nothing) Then
        rect.Width = 250
        rect.Height = 150
    End If
End Sub
```

Manipulation events

Use manipulation events in your app if you need to support multiple finger interactions or interactions that require velocity data.

You can use manipulation events to detect interactions such as drag, zoom, and hold.

> **Note**
>
> The touchpad does not raise Manipulation events. Instead, pointer events will be raised for touchpad input.

Here is a list of manipulation events and related event arguments.

Expand table

Event or class	Description
ManipulationStarting event	Occurs when the manipulation processor is first created.
ManipulationStarted event	Occurs when an input device begins a manipulation on the UIElement.
ManipulationDelta event	Occurs when the input device changes position during a manipulation.
ManipulationInertiaStarting event	Occurs when the input device loses contact with the UIElement object during a manipulation and inertia begins.
ManipulationCompleted event	Occurs when a manipulation and inertia on the UIElement are complete.
ManipulationStartingRoutedEventArgs	Provides data for the ManipulationStarting event.
ManipulationStartedRoutedEventArgs	Provides data for the ManipulationStarted event.
ManipulationDeltaRoutedEventArgs	Provides data for the ManipulationDelta event.
ManipulationInertiaStartingRoutedEventArgs	Provides data for the ManipulationInertiaStarting event.
ManipulationVelocities	Describes the speed at which manipulations occur.
ManipulationCompletedRoutedEventArgs	Provides data for the ManipulationCompleted event.

A gesture consists of a series of manipulation events. Each gesture starts with a ManipulationStarted event, such as when a user touches the screen.

Next, one or more ManipulationDelta events are fired. For example, if you touch the screen and then drag your finger across the screen. Finally, a ManipulationCompleted event is raised when the interaction finishes.

> **Note**
>
> If you don't have a touch-screen monitor, you can test your manipulation event code in the simulator using a mouse and mouse wheel interface.

The following example shows how to use the ManipulationDelta events to handle a slide interaction on a Rectangle and move it across the screen.

First, a Rectangle named `touchRectangle` is created in XAML with a Height and Width of 200.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Rectangle Name="touchRectangle"
               Width="200" Height="200" Fill="Blue"
               ManipulationMode="All"/>
</Grid>
```

Next, a global TranslateTransform named `dragTranslation` is created for translating the Rectangle. A ManipulationDelta event listener is specified on the **Rectangle**, and `dragTranslation` is added to the RenderTransform of the **Rectangle**.

C++

```
// Global translation transform used for changing the position of
// the Rectangle based on input data from the touch contact.
Windows::UI::Xaml::Media::TranslateTransform^ dragTranslation;
```

cs

```
// Global translation transform used for changing the position of
// the Rectangle based on input data from the touch contact.
private TranslateTransform dragTranslation;
```

VB

```
' Global translation transform used for changing the position of
' the Rectangle based on input data from the touch contact.
Private dragTranslation As TranslateTransform
```

C++

```
MainPage::MainPage()
{
    InitializeComponent();

    // Listener for the ManipulationDelta event.
    touchRectangle->ManipulationDelta +=
        ref new ManipulationDeltaEventHandler(
            this,
            &MainPage::touchRectangle_ManipulationDelta);
    // New translation transform populated in
    // the ManipulationDelta handler.
    dragTranslation = ref new TranslateTransform();
    // Apply the translation to the Rectangle.
    touchRectangle->RenderTransform = dragTranslation;
}
```

cs

```
public MainPage()
{
    this.InitializeComponent();

    // Listener for the ManipulationDelta event.
    touchRectangle.ManipulationDelta += touchRectangle_ManipulationDelta;
    // New translation transform populated in
    // the ManipulationDelta handler.
    dragTranslation = new TranslateTransform();
    // Apply the translation to the Rectangle.
    touchRectangle.RenderTransform = this.dragTranslation;
}
```

VB

```
Public Sub New()

    ' This call is required by the designer.
    InitializeComponent()

    ' Listener for the ManipulationDelta event.
    AddHandler touchRectangle.ManipulationDelta,
        AddressOf testRectangle_ManipulationDelta
    ' New translation transform populated in
    ' the ManipulationDelta handler.
    dragTranslation = New TranslateTransform()
```

```
    ' Apply the translation to the Rectangle.
    touchRectangle.RenderTransform = dragTranslation

End Sub
```

Finally, in the **ManipulationDelta** event handler, the position of the **Rectangle** is updated by using the **TranslateTransform** on the **Delta** property.

C++

```
// Handler for the ManipulationDelta event.
// ManipulationDelta data is loaded into the
// translation transform and applied to the Rectangle.
void MainPage::touchRectangle_ManipulationDelta(Object^ sender,
    ManipulationDeltaRoutedEventArgs^ e)
{
    // Move the rectangle.
    dragTranslation->X += e->Delta.Translation.X;
    dragTranslation->Y += e->Delta.Translation.Y;

}
```

cs

```
// Handler for the ManipulationDelta event.
// ManipulationDelta data is loaded into the
// translation transform and applied to the Rectangle.
void touchRectangle_ManipulationDelta(object sender,
    ManipulationDeltaRoutedEventArgs e)
{
    // Move the rectangle.
    dragTranslation.X += e.Delta.Translation.X;
    dragTranslation.Y += e.Delta.Translation.Y;
}
```

VB

```
' Handler for the ManipulationDelta event.
' ManipulationDelta data Is loaded into the
' translation transform And applied to the Rectangle.
Private Sub testRectangle_ManipulationDelta(
    sender As Object,
    e As ManipulationDeltaRoutedEventArgs)

    ' Move the rectangle.
    dragTranslation.X = (dragTranslation.X + e.Delta.Translation.X)
    dragTranslation.Y = (dragTranslation.Y + e.Delta.Translation.Y)

End Sub
```

Routed events

All of the pointer events, gesture events and manipulation events mentioned here are implemented as *routed events*. This means that the event can potentially be handled by objects other than the one that originally raised the event. Successive parents in an object tree, such as the parent containers of a **UIElement** or the root **Page** of your app, can choose to handle these events even if the original element does not. Conversely, any object that does handle the event can mark the event handled so that it no longer reaches any parent element. For more info about the routed event concept and how it affects how you write handlers for routed events, see Events and routed events overview.

> **ⓘ Important**
>
> If you need to handle pointer events for a **UIElement** in a scrollable view (such as a ScrollViewer or ListView), you must explicitly disable support for manipulation events on the element in the view by calling **UIElement.CancelDirectmanipulation()**. To re-enable manipulation events in the view, call **UIElement.TryStartDirectManipulation()**.

Dos and don'ts

- Design applications with touch interaction as the primary expected input method.
- Provide visual feedback for interactions of all types (touch, pen, stylus, mouse, etc.)
- Optimize targeting by adjusting touch target size, contact geometry, scrubbing and rocking.
- Optimize accuracy through the use of snap points and directional "rails".
- Provide tooltips and handles to help improve touch accuracy for tightly packed UI items.
- Don't use timed interactions whenever possible (example of appropriate use: touch and hold).
- Don't use the number of fingers used to distinguish the manipulation whenever possible.

Related articles

- Handle pointer input
- Identify input devices

Samples

- Basic input sample
- Low latency input sample
- User interaction mode sample
- Focus visuals sample

Archive Samples

- Input: Device capabilities sample
- Input: XAML user input events sample
- XAML scrolling, panning, and zooming sample
- Input: Gestures and manipulations with GestureRecognizer

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Mouse interactions

Article • 06/24/2021

Optimize your Windows app design for touch input and get basic mouse support by default.

Mouse input is best suited for user interactions that require precision when pointing and clicking. This inherent precision is naturally supported by the UI of Windows, which is optimized for the imprecise nature of touch.

Where mouse and touch input diverge is the ability for touch to more closely emulate the direct manipulation of UI elements through physical gestures performed directly on those objects (such as swiping, sliding, dragging, rotating, and so on). Manipulations with a mouse typically require some other UI affordance, such as the use of handles to resize or rotate an object.

This topic describes design considerations for mouse interactions.

The UWP app mouse language

A concise set of mouse interactions are used consistently throughout the system.

Term	Description
Hover to learn	Hover over an element to display more detailed info or teaching visuals (such as a tooltip) without a commitment to an action.
Left-click for primary action	Left-click an element to invoke its primary action (such as launching an app or executing a command).

Term	Description
Scroll to change view	Display scroll bars to move up, down, left, and right within a content area. Users can scroll by clicking scroll bars or rotating the mouse wheel. Scroll bars can indicate the location of the current view within the content area (panning with touch displays a similar UI).
Right-click to select and command	Right-click to display the navigation bar (if available) and the app bar with global commands. Right-click an element to select it and display the app bar with contextual commands for the selected element. **Note** Right-click to display a context menu if selection or app bar commands are not appropriate UI behaviors. But we strongly recommend that you use the app bar for all command behaviors.
UI commands to zoom	Display UI commands in the app bar (such as + and -), or press Ctrl and rotate mouse wheel, to emulate pinch and stretch gestures for zooming.
UI commands to rotate	Display UI commands in the app bar, or press Ctrl+Shift and rotate mouse wheel, to emulate the turn gesture for rotating. Rotate the device itself to rotate the entire screen.
Left-click and drag to rearrange	Left-click and drag an element to move it.
Left-click and drag to select text	Left-click within selectable text and drag to select it. Double-click to select a word.

Mouse input events

Most mouse input can be handled through the common routed input events supported by all **UIElement** objects. These include:

- **BringIntoViewRequested**
- **CharacterReceived**
- **ContextCanceled**

- ContextRequested
- DoubleTapped
- DragEnter
- DragLeave
- DragOver
- DragStarting
- Drop
- DropCompleted
- GettingFocus
- GotFocus
- Holding
- KeyDown
- KeyUp
- LosingFocus
- LostFocus
- ManipulationCompleted
- ManipulationDelta
- ManipulationInertiaStarting
- ManipulationStarted
- ManipulationStarting
- NoFocusCandidateFound
- PointerCanceled
- PointerCaptureLost
- PointerEntered
- PointerExited
- PointerMoved
- PointerPressed
- PointerReleased
- PointerWheelChanged
- PreviewKeyDown
- PreviewKeyUp
- RightTapped
- Tapped

However, you can take advantage of the specific capabilities of each device (such as mouse wheel events) using the pointer, gesture, and manipulation events in Windows.UI.Input.

Samples: See our BasicInput sample, for .

Guidelines for visual feedback

- When a mouse is detected (through move or hover events), show mouse-specific UI to indicate functionality exposed by the element. If the mouse doesn't move for a certain amount of time, or if the user initiates a touch interaction, make the mouse UI gradually fade away. This keeps the UI clean and uncluttered.
- Don't use the cursor for hover feedback, the feedback provided by the element is sufficient (see Cursors below).
- Don't display visual feedback if an element doesn't support interaction (such as static text).
- Don't use focus rectangles with mouse interactions. Reserve these for keyboard interactions.
- Display visual feedback concurrently for all elements that represent the same input target.
- Provide buttons (such as + and -) for emulating touch-based manipulations such as panning, rotating, zooming, and so on.

For more general guidance on visual feedback, see Guidelines for visual feedback.

Cursors

A set of standard cursors is available for a mouse pointer. These are used to indicate the primary action of an element.

Each standard cursor has a corresponding default image associated with it. The user or an app can replace the default image associated with any standard cursor at any time. Specify a cursor image through the **PointerCursor** function.

If you need to customize the mouse cursor:

- Always use the arrow cursor ( ) for clickable elements. don't use the pointing hand cursor ( ) for links or other interactive elements. Instead, use hover effects (described earlier).
- Use the text cursor ( ) for selectable text.
- Use the move cursor ( ) when moving is the primary action (such as dragging or cropping). Don't use the move cursor for elements where the primary action is navigation (such as Start tiles).
- Use the horizontal, vertical and diagonal resize cursors ( ), when an object is resizable.
- Use the grasping hand cursors ( ) when panning content within a fixed canvas (such as a map).

Related articles

- Handle pointer input
- Identify input devices
- Events and routed events overview

Samples

- Basic input sample
- Low latency input sample
- User interaction mode sample
- Focus visuals sample

Keyboard interactions

Article • 06/11/2024

Learn how to design and optimize your Windows apps so they provide the best experiences for both keyboard power users and those with disabilities and other accessibility requirements.

Across devices, keyboard input is an important part of the overall Windows app interaction experience. A well-designed keyboard experience lets users efficiently navigate the UI of your app and access its full functionality without ever lifting their hands from the keyboard.

Common interaction patterns are shared between keyboard and gamepad

In this topic, we focus specifically on Windows app design for keyboard input on PCs. However, a well-designed keyboard experience is important for supporting accessibility tools such as Windows Narrator, using software keyboards such as the touch keyboard and the On-Screen Keyboard (OSK), and for handling other input device types, such as a game pad or remote control.

Many of the guidelines and recommendations discussed here, including focus visuals, access keys, and UI navigation, are also applicable to these other scenarios.

NOTE While both hardware and software keyboards are used for text input, the focus of this topic is navigation and interaction.

Built-in support

Along with the mouse, the keyboard is the most widely used peripheral on PCs and, as such, is a fundamental part of the PC experience. PC users expect a comprehensive and consistent experience from both the system and individual apps in response to keyboard input.

All UWP controls include built-in support for rich keyboard experiences and user interactions, while the platform itself provides an extensive foundation for creating keyboard experiences that you feel are best suited to both your custom controls and apps.

UWP supports keyboard with any device

Basic experiences

As mentioned previously, input devices such as a game pad and remote control, and accessibility tools such as Narrator, share much of the keyboard input experience for navigation and commanding. This common experience across input types and tools minimizes additional work from you and contributes to the "build once, run anywhere" goal of the Universal Windows Platform.

Where necessary, we'll identify key differences you should be aware of and describe any mitigations you should consider.

Here are the devices and tools discussed in this topic:

Expand table

Device/tool	Description
Keyboard (hardware and software)	In addition to the standard hardware keyboard, Windows applications support two software keyboards: the touch (or software) keyboard and the On-Screen Keyboard.
Game pad and remote control	A game pad and remote control are fundamental input devices in the 10-foot experience. For specific details on Windows support for game pad and remote control, see Game pad and remote control interactions.
Screen readers (Narrator)	Narrator is a built-in screen reader for Windows that provides unique interaction experiences and functionality, but still relies on basic keyboard navigation and input. For Narrator details, see Getting started with Narrator.

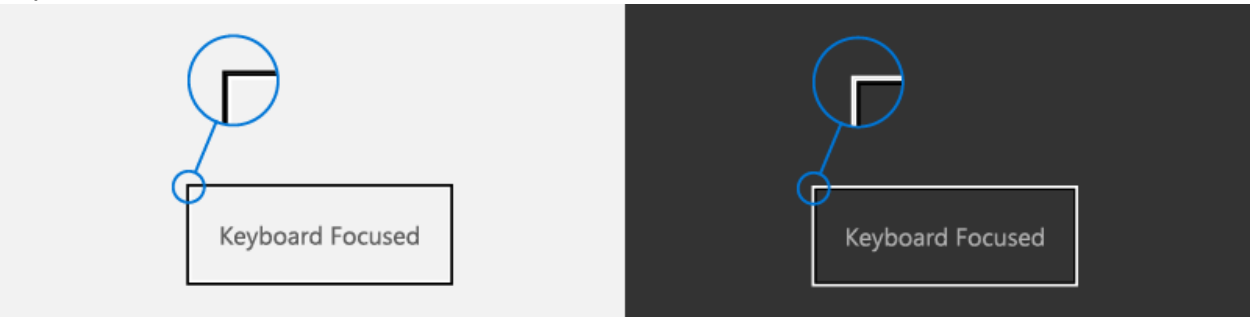
Custom experiences and efficient keyboarding

As mentioned, keyboard support is integral to ensuring your applications work great for users with different skills, abilities, and expectations. We recommend that you prioritize the following.

- Support keyboard navigation and interaction
  - Ensure actionable items are identified as tab stops (and non-actionable items are not), and navigation order is logical and predictable (see Tab stops)
  - Set initial focus on the most logical element (see Initial focus)
  - Provide arrow key navigation for "inner navigations" (see Navigation)
- Support keyboard shortcuts
  - Provide accelerator keys for quick actions (see Accelerators)
  - Provide access keys to navigate your application's UI (see Access keys)

Focus visuals

The UWP supports a single focus visual design that works well for all input types and experiences.



A focus visual:

- Is shown when a UI element receives focus from a keyboard and/or gamepad/remote control
- Is rendered as a highlighted border around the UI element to indicate an action can be taken
- Helps a user navigate an app UI without getting lost
- Can be customized for your app (See High visibility focus visuals)

NOTE The UWP focus visual is not the same as the Narrator focus rectangle.

Tab stops

To use a control (including navigation elements) with the keyboard, the control must have focus. One way for a control to receive keyboard focus is to make it accessible through tab navigation by identifying it as a tab stop in your application's tab order.

For a control to be included in the tab order, the IsEnabled property must be set to **true** and the IsTabStop property must be set to **true**.

To specifically exclude a control from the tab order, set the IsTabStop property to **false**.

By default, tab order reflects the order in which UI elements are created. For example, if a `StackPanel` contains a `Button`, a `Checkbox`, and a `TextBox`, tab order is `Button`, `Checkbox`, and `TextBox`.

You can override the default tab order by setting the TabIndex property.

Tab order should be logical and predictable

A well-designed keyboard navigation model, using a logical and predictable tab order, makes your app more intuitive and helps users explore, discover, and access functionality more efficiently and effectively.

All interactive controls should have tab stops (unless they are in a group), while non-interactive controls, such as labels, should not.

Avoid a custom tab order that makes the focus jump around in your application. For example, a list of controls in a form should have a tab order that flows from top to bottom and left to right (depending on locale).

See Keyboard accessibility for more details about customizing tab stops.

Try to coordinate tab order and visual order

Coordinating tab order and visual order (also referred to as reading order or display order) helps reduce confusion for users as they navigate through your application's UI.

Try to rank and present the most important commands, controls, and content first in both the tab order and the visual order. However, the actual display position can depend on the parent layout container and certain properties of the child elements that influence the layout. Specifically, layouts that use a grid metaphor or a table metaphor can have a visual order quite different from the tab order.

NOTE Visual order is also dependent on locale and language.

Initial focus

Initial focus specifies the UI element that receives focus when an application or a page is first launched or activated. When using a keyboard, it is from this element that a user starts interacting with your application's UI.

For UWP apps, initial focus is set to the element with the highest TabIndex that can receive focus. Child elements of container controls are ignored. In a tie, the first element in the visual tree receives focus.

Set initial focus on the most logical element

Set initial focus on the UI element for the first, or primary, action that users are most likely to take when launching your app or navigating to a page. Some examples include:

- A photo app where focus is set to the first item in a gallery
- A music app where focus is set to the play button

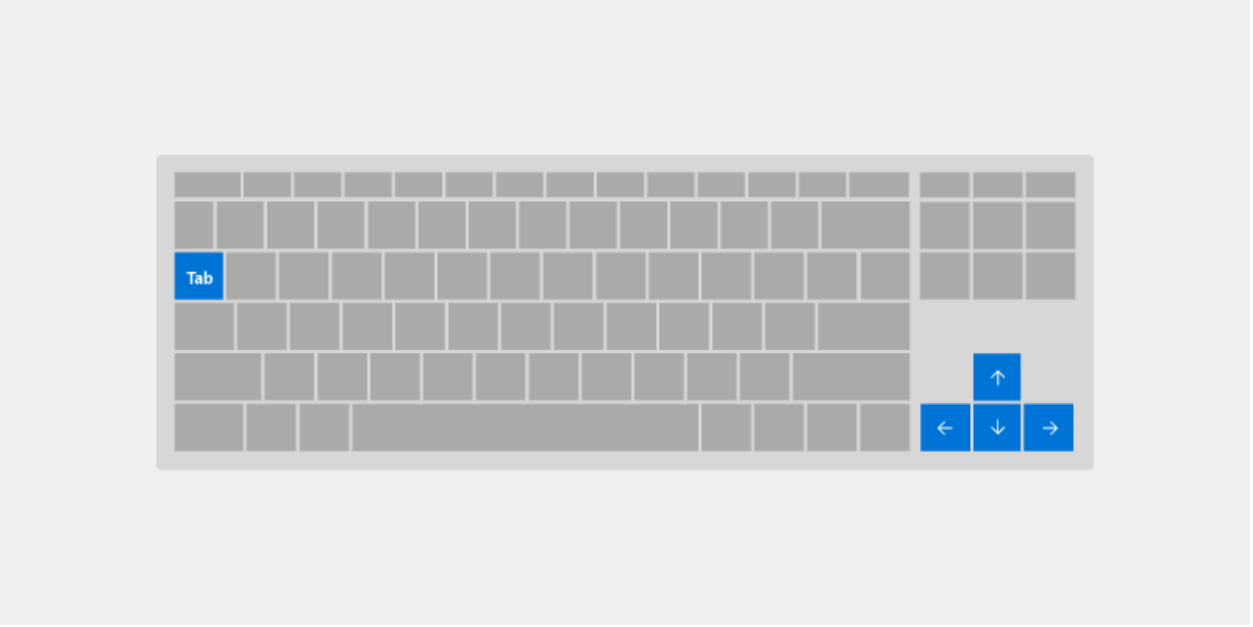
Don't set initial focus on an element that exposes a potentially negative, or even disastrous, outcome

This level of functionality should be a user's choice. Setting initial focus to an element with a significant outcome might result in unintended data loss or system access. For example, don't set focus to the delete button when navigating to an e-mail.

See Focus navigation for more details about overriding tab order.

Navigation

Keyboard navigation is typically supported through the Tab keys and the Arrow keys.



By default, UWP controls follow these basic keyboard behaviors:

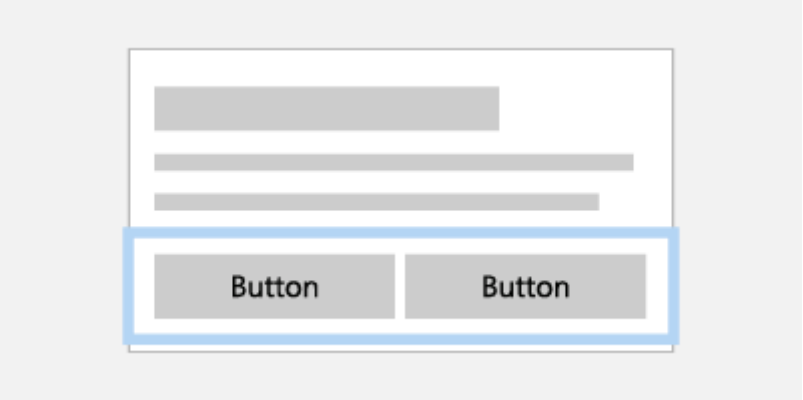
- **Tab keys** navigate between actionable/active controls in tab order.
- **Shift + Tab** navigate controls in reverse tab order. If user has navigated inside the control using arrow key, focus is set to the last known value inside the control.
- **Arrow keys** expose control-specific "inner navigation" When user enters "inner navigation,"" arrow keys do not navigate out of a control. Some examples include:
  - Up/Down arrow key moves focus inside `ListView` and `MenuFlyout`
  - Modify currently selected values for `Slider` and `RatingsControl`
  - Move caret inside `TextBox`
  - Expand/collapse items inside `TreeView`

Use these default behaviors to optimize your application's keyboard navigation.

Use "inner navigation" with sets of related controls

Providing arrow key navigation into a set of related controls reinforces their relationship within the overall organization of your application's UI.

For example, the `ContentDialog` control shown here provides inner navigation by default for a horizontal row of buttons (for custom controls, see the Control Group section).



Interaction with a collection of related buttons is made easier with arrow key navigation

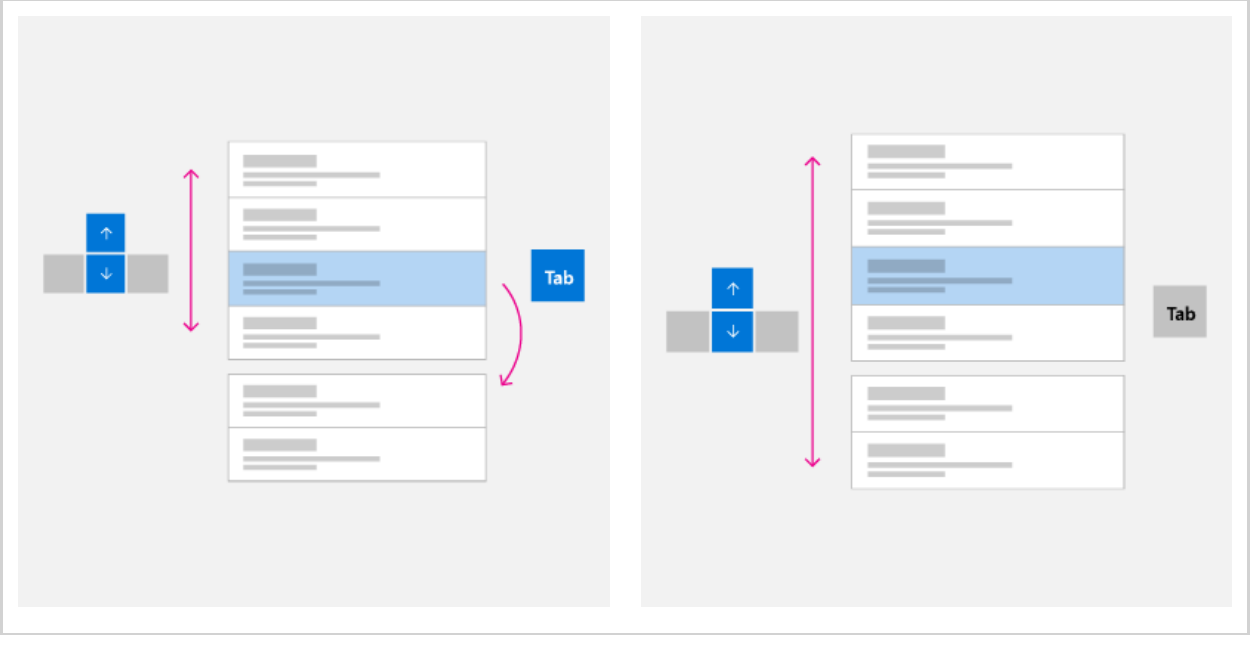
If items are displayed in a single column, Up/Down arrow key navigates items. If items are displayed in a single row, Right/Left arrow key navigates items. If items are multiple columns, all 4 arrow keys navigate.

Define a single tab stop for a collection of related controls

By defining a single tab stop for a collection of related, or complementary, controls, you can minimize the number of overall tab stops in your app.

For example, the following images show two stacked `ListView` controls. The image on the left shows arrow key navigation used with a tab stop to navigate between `ListView` controls, while the image on the right shows how navigation between child elements could be made easier and more efficient by eliminating the need for to traverse parent controls with a tab key.

Expand table



Interaction with two stacked ListView controls can be made easier and more efficient by eliminating the tab stop and navigating with just arrow keys.

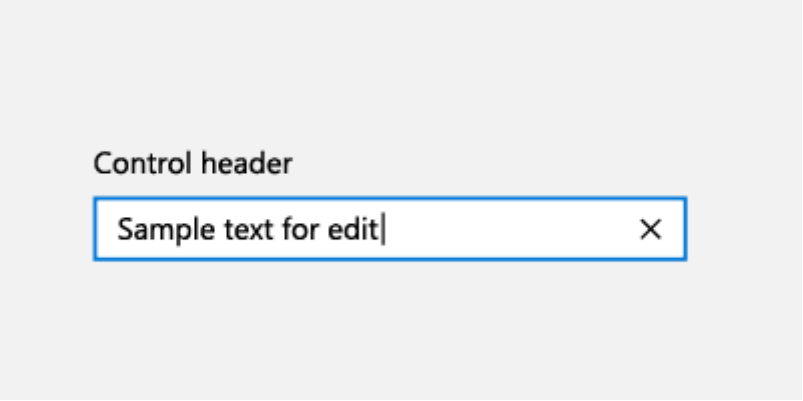
Visit Control Group section to learn how to apply the optimization examples to your application UI.

Interaction and commanding

Once a control has focus, a user can interact with it and invoke any associated functionality using specific keyboard input.

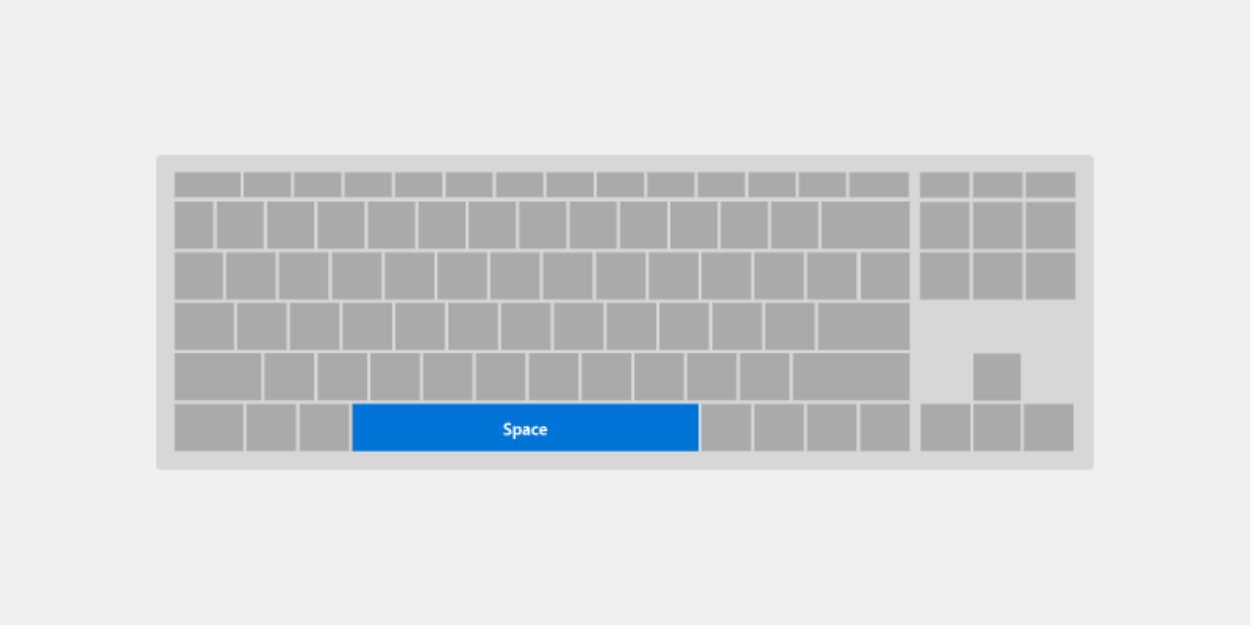
Text entry

For those controls specifically designed for text input such as `TextBox` and `RichEditBox`, all keyboard input is used for entering or navigating text, which takes priority over other keyboard commands. For example, the drop down menu for an `AutoSuggestBox` control does not recognize the **Space** key as a selection command.



Space key

When not in text entry mode, the **Space** key invokes the action or command associated with the focused control (just like a tap with touch or a click with a mouse).

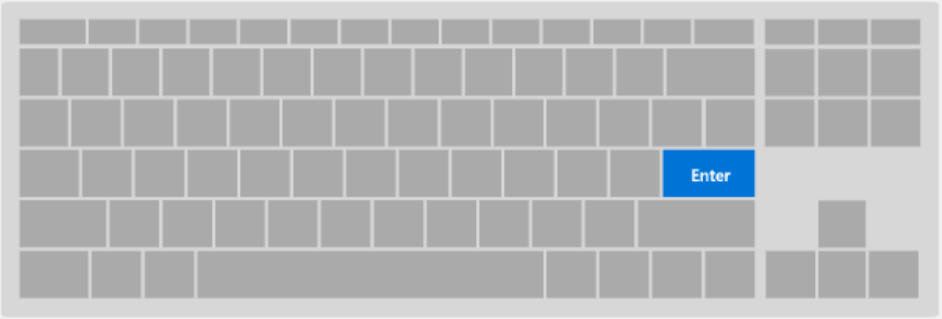


Enter key

The **Enter** key can perform a variety of common user interactions, depending on the control with focus:

- Activates command controls such as a `Button` or `Hyperlink`. To avoid end user confusion, the **Enter** key also activates controls that look like command controls such as `ToggleButton` or `AppBarToggleButton`.
- Displays the picker UI for controls such as `ComboBox` and `DatePicker`. The **Enter** key also commits and closes the picker UI.
- Activates list controls such as `ListView`, `GridView`, and `ComboBox`.
  - The **Enter** key performs the selection action as the **Space** key for list and grid items, unless there is an additional action associated with these items (opening a new window).
  - If an additional action is associated with the control, the **Enter** key performs the additional action and the **Space** key performs the selection action.

NOTE The **Enter** key and **Space** key do not always perform the same action, but often do.

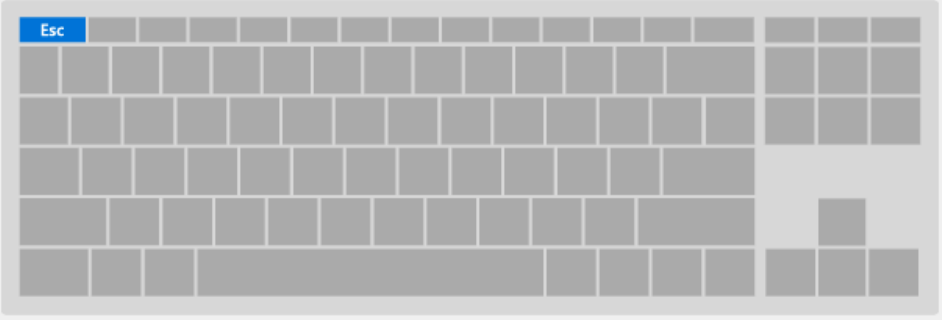


Esc key

The Esc key lets a user cancel transient UI (along with any ongoing actions in that UI).

Examples of this experience include:

- User opens a `ComboBox` with a selected value and uses the arrow keys to move the focus selection to a new value. Pressing the Esc key closes the `ComboBox` and resets the selected value back to the original value.
- User invokes a permanent delete action for an email and is prompted with a `ContentDialog` to confirm the action. The user decides this is not the intended action and presses the **Esc** key to close the dialog. As the **Esc** key is associated with the **Cancel** button, the dialog is closed and the action is canceled. The **Esc** key only affects transient UI, it does not close, or back navigate through, app UI.

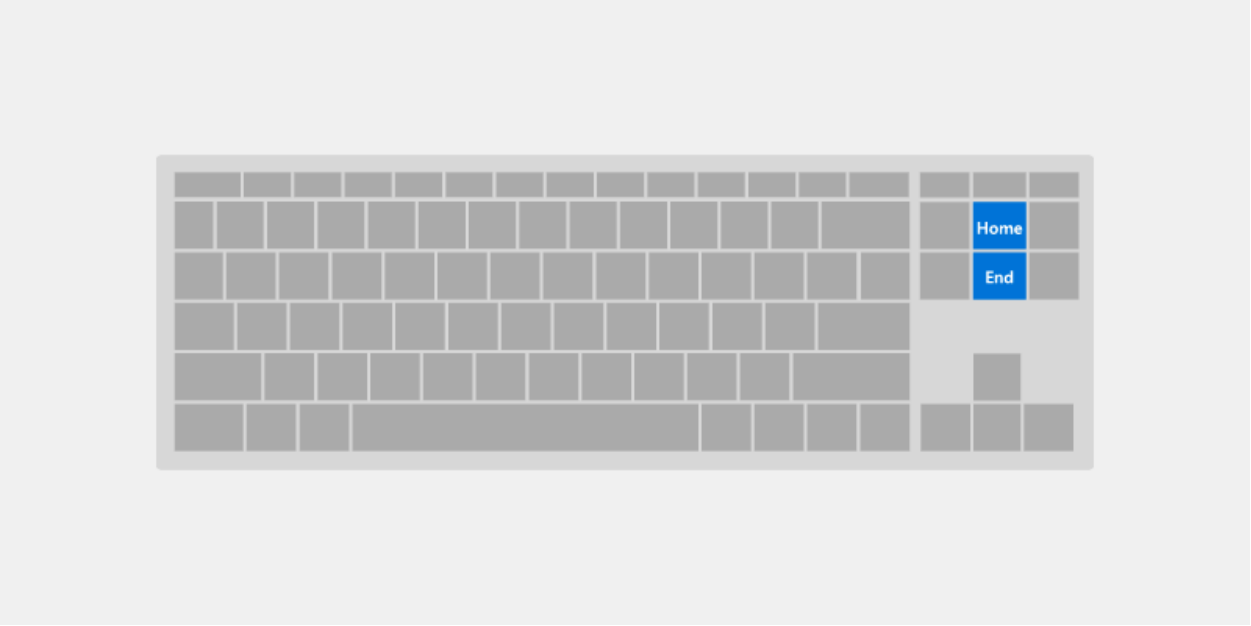


Home and End keys

The **Home** and **End** keys let a user scroll to the beginning or end of a UI region.

Examples of this experience include:

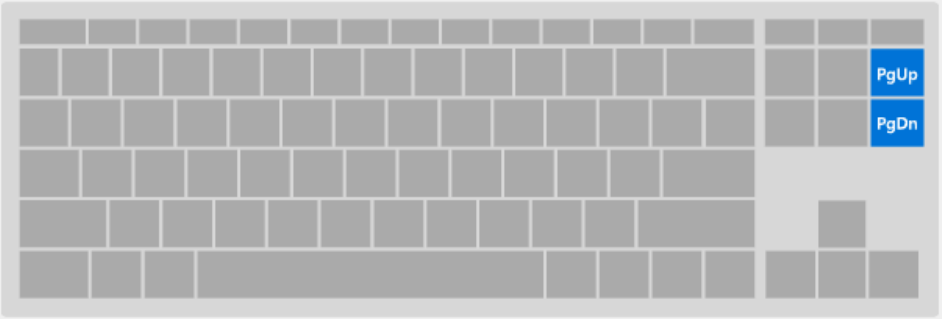
- For `ListView` and `GridView` controls, the **Home** key moves focus to the first element and scrolls it into view, whereas the **End** key moves focus to the last element and scrolls it into view.
- For a `ScrollView` control, the **Home** key scrolls to the top of the region, whereas the **End** key scrolls to the bottom of the region (focus is not changed).



Page up and Page down keys

The **Page** keys let a user scroll a UI region in discrete increments.

For example, for `ListView` and `GridView` controls, the **Page up** key scrolls the region up by a "page" (typically the viewport height) and moves focus to the top of the region. Alternately, the **Page down** key scrolls the region down by a page and moves focus to the bottom of the region.



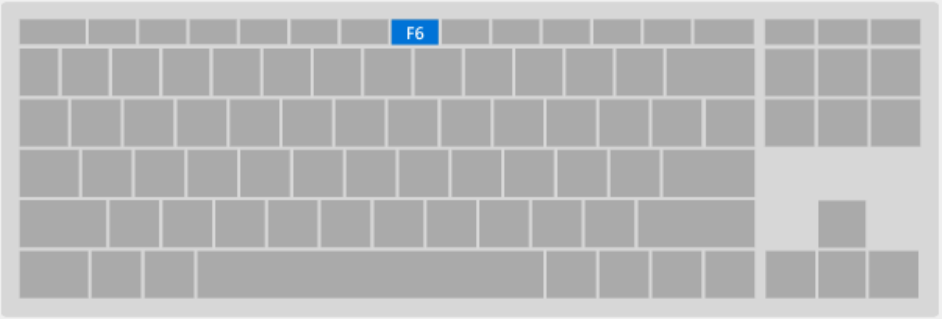
F6 key

The **F6** key lets a user cycle between panes or important sections of your app or UI. **Shift-F6** typically cycles backwards (see Keyboard accessibility).

These are often related to landmarks and headings, but do not need to correspond directly.

For example:

- In Edge, pressing F6 will cycle between the tab bar, the address bar/app bar, and the page content.
- In File Explorer, pressing F6 will cycle between the sections of the app.
- On the desktop, pressing F6 will cycle between parts of the taskbar and the desktop.



Keyboard shortcuts

In addition to implementing keyboard navigation and activation, it is also good practice to implement keyboard shortcuts such as keyboard accelerators and access keys for important or frequently used functionality.

Keyboard shortcuts can make your app easier to use by providing both enhanced support for accessibility and improved efficiency for keyboard users.

A shortcut is a keyboard combination that enhances productivity by providing an efficient way for the user to access app functionality. There are two kinds of shortcut:

- Accelerators are shortcuts that invoke an app command. Your app may or may not provide specific UI that corresponds to the command. Accelerators typically consist of the Ctrl key plus a letter key.
- Access keys are shortcuts that set focus to specific UI in your application. Access keys typically consist of the Alt key plus a letter key.

Providing consistent keyboard shortcuts that support similar tasks across applications makes them much more useful and powerful and helps users remember them.

Accelerators

Accelerators help users perform common actions in an application much more quickly and efficiently.

Examples of Accelerators:

- Pressing Ctrl + N key anywhere in the **Mail** app launches a new mail item.
- Pressing Ctrl + E key anywhere in Microsoft Edge (and many Microsoft Store applications) launches search.

Accelerators have the following characteristics:

- They primarily use Ctrl and Function key sequences (Windows system shortcut keys also use Alt + non-alphanumeric keys and the Windows logo key).
- They are assigned only to the most commonly used commands.
- They are intended to be memorized, and are documented only in menus, tooltips, and Help.
- They have effect throughout the entire application, when supported.
- They should be assigned consistently as they are memorized and not directly documented.

Access keys

See Access keys page for more in-depth information for supporting access keys with UWP.

Access keys help users with motor function disabilities an ability to press one key at a time to action on a specific item in the UI. Moreover, access keys can be used to communicate additional shortcut keys to help advanced users perform actions quickly.

Access keys have the following characteristics:

- They use the Alt key plus an alphanumeric key.
- They are primarily for accessibility.
- They are documented directly in the UI, adjacent to the control, through Key Tips.
- They have effect only in the current window, and navigate to the corresponding menu item or control.
- Access keys should be assigned consistently to commonly used commands (especially commit buttons), whenever possible.
- They are localized.

Common keyboard shortcuts

The following table is a small sample of frequently used keyboard shortcuts.

Expand table

Action	Key command
Select all	Ctrl+A
Continuously select	Shift+Arrow key
Save	Ctrl+S
Find	Ctrl+F
Print	Ctrl+P
Copy	Ctrl+C
Cut	Ctrl+X
Paste	Ctrl+V
Undo	Ctrl+Z
Next tab	Ctrl+Tab

Action	Key command
Close tab	Ctrl+F4 or Ctrl+W
Semantic zoom	Ctrl++ or Ctrl+-

For a comprehensive list of Windows system shortcuts, see keyboard shortcuts for Windows. For common application shortcuts, see keyboard shortcuts for Microsoft applications.

Advanced experiences

In this section, we discuss some of the more complex keyboard interaction experiences supported by UWP apps, along with some of the behaviors you should be aware of when your app is used on different devices and with different tools.

Control group

You can group a set of related, or complementary, controls in a "control group" (or directional area), which enables "inner navigation" using the arrow keys. The control group can be a single tab stop, or you can specify multiple tab stops within the control group.

Arrow key navigation

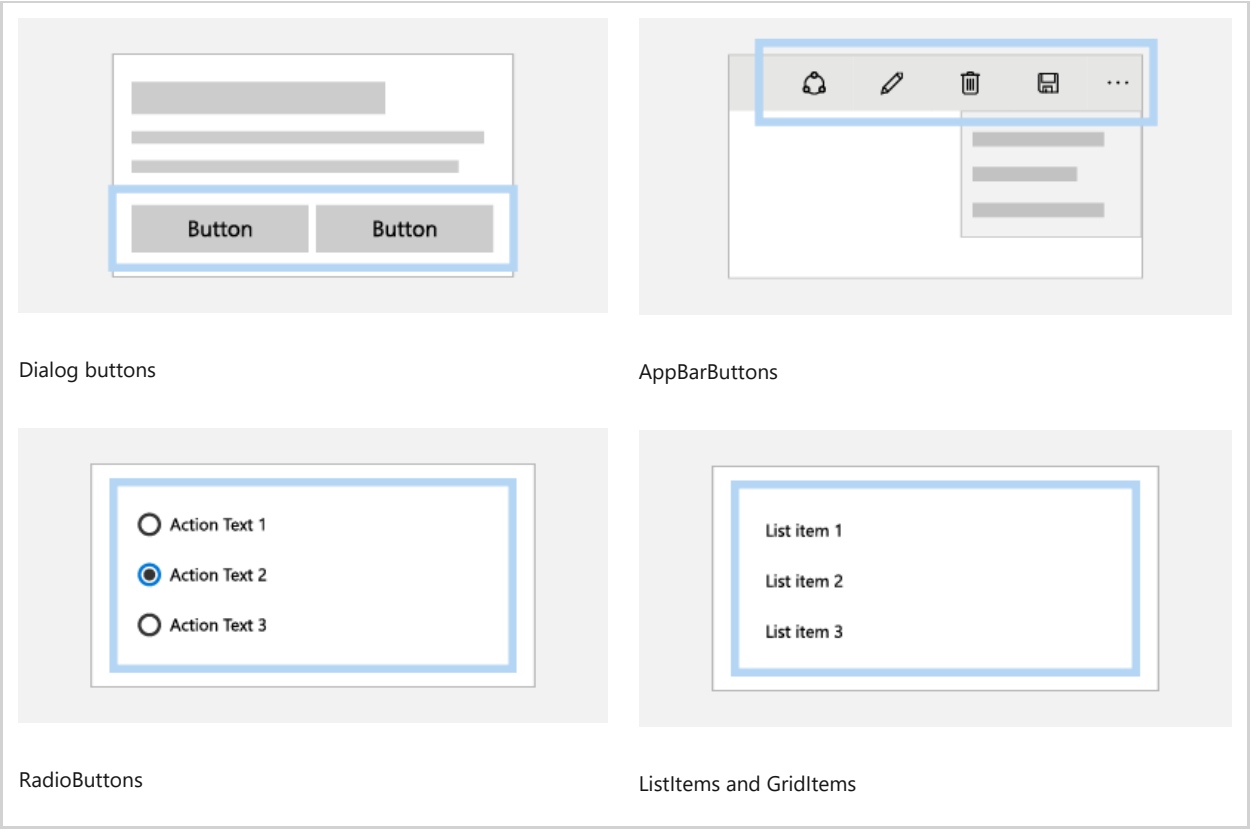
Users expect support for arrow key navigation when there is a group of similar, related controls in a UI region:

- `AppBarButtons` in a `CommandBar`
- `ListItems` or `GridItems` inside `ListView` or `GridView`
- `Buttons` inside `ContentDialog`

UWP controls support arrow key navigation by default. For custom layouts and control groups, use `XYFocusKeyboardNavigation="Enabled"` to provide similar behavior.

Consider adding support for arrow key navigation when using the following controls:

Expand table



Tab stops

Depending on your application's functionality and layout, the best navigation option for a control group might be a single tab stop with arrow navigation to child elements, multiple tab stops, or some combination.

Use multiple tab stops and arrow keys for buttons

Accessibility users rely on well-established keyboard navigation rules, which do not typically use arrow keys to navigate a collection of buttons. However, users without visual impairments might feel that the behavior is natural.

An example of default UWP behavior in this case is the `ContentDialog`. While arrow keys can be used to navigate between buttons, each button is also a tab stop.

Assign single tab stop to familiar UI patterns

In cases where your layout follows a well-known UI pattern for control groups, assigning a single tab stop to the group can improve navigation efficiency for users.

Examples include:

- `RadioButtons`
- Multiple `ListViews` that look like and behave like a single `ListView`

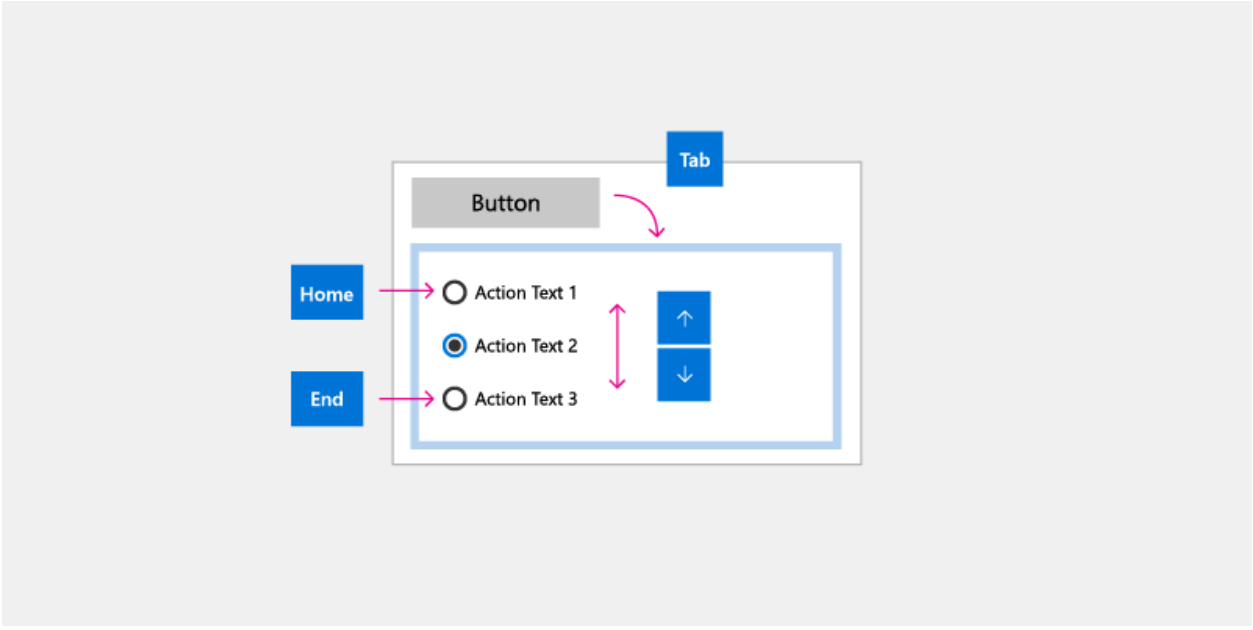
- Any UI made to look and behave like grid of tiles (such as the Start menu tiles)

Specifying control group behavior

Use the following APIs to support custom control group behavior (all are discussed in more detail later in this topic):

- XYFocusKeyboardNavigation enables arrow key navigation between controls
- TabFocusNavigation indicates whether there are multiple tab stops or single tab stop
- FindFirstFocusableElement and FindLastFocusableElement sets focus on the first item with **Home** key and the last item with **End** key

The following image shows an intuitive keyboard navigation behavior for a control group of associated radio buttons. In this case, we recommend a single tab stop for the control group, inner navigation between the radio buttons using the arrow keys, **Home** key bound to the first radio button, and **End** key bound to the last radio button.



Keyboard and Narrator

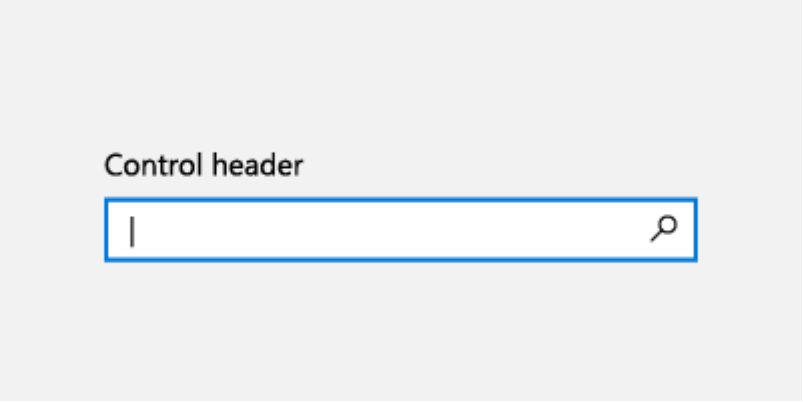
Narrator is a UI accessibility tool geared towards keyboard users (other input types are also supported). However, Narrator functionality goes beyond the keyboard interactions supported by UWP apps and extra care is required when designing your UWP app for Narrator. (The Narrator basics page guides you through the Narrator user experience.)

Some of the differences between UWP keyboard behaviors and those supported by Narrator include:

- Extra key combinations for navigation to UI elements that are not exposed through standard keyboard navigation, such as Caps lock + arrow keys to read control labels.
- Navigation to disabled items. By default, disabled items are not exposed through standard keyboard navigation.
- Control "views" for quicker navigation based on UI granularity. Users can navigate to items, characters, word, lines, paragraphs, links, headings, tables, landmarks, and suggestions. Standard keyboard navigation exposes these objects as a flat list, which might make navigation cumbersome unless you provide shortcut keys.

Case Study – AutoSuggestBox control

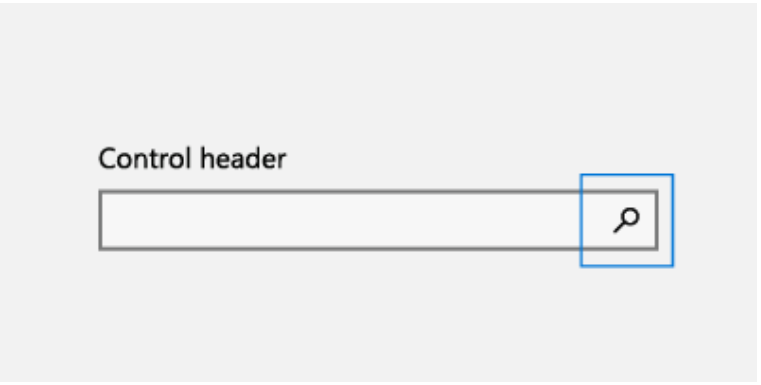
The search button for the `AutoSuggestBox` is not accessible to standard keyboard navigation using tab and arrow keys because the user can press the **Enter** key to submit the search query. However, it is accessible through Narrator when the user presses Caps Lock + an arrow key.



*With keyboard, users press the **Enter** key to submit search query*

Expand table

*With Narrator, users press the **Enter** key to submit search query*



*With Narrator, users are also able to access the search button using the **Caps Lock + Right arrow key**, then pressing **Space** key*

Keyboard, game pad, and remote control

Game pads and remote controls support many UWP keyboard behaviors and experiences. However, due to the lack of various key options available on a keyboard, game pad and remote control lack many keyboard optimizations (remote control is even more limited than game pad).

See Game pad and remote control interactions for more detail on UWP support for game pad and remote control input.

The following shows some key mappings between keyboard, game pad, and remote control.

Expand table

Keyboard	Game pad	Remote control
Space	A button	Select button
Enter	A button	Select button
Escape	B button	Back button
Home/End	N/A	N/A
Page Up/Down	Trigger button for vertical scroll, Bumper button for horizontal scroll	N/A

Some key differences you should be aware of when designing your UWP app for use with game pad and remote control usage include:

- Text entry requires the user to press A to activate a text control.
- Focus navigation is not limited to control groups, users can navigate freely to any focusable UI element in the app.

  NOTE Focus can move to any focusable UI element in the key press direction unless it is in an overlay UI or focus engagement is specified, which prevents focus from entering/exiting a region until engaged/disengaged with the A button. For more info, see the directional navigation section.

- D-pad and left stick buttons are used to move focus between controls and for inner navigation.

  NOTE Gamepad and remote control only navigate to items that are in the same visual order as the directional key pressed. Navigation is disabled in that direction

when there is no subsequent element that can receive focus. Depending on the situation, keyboard users do not always have that constraint. See the Built in keyboard optimization section for more info.

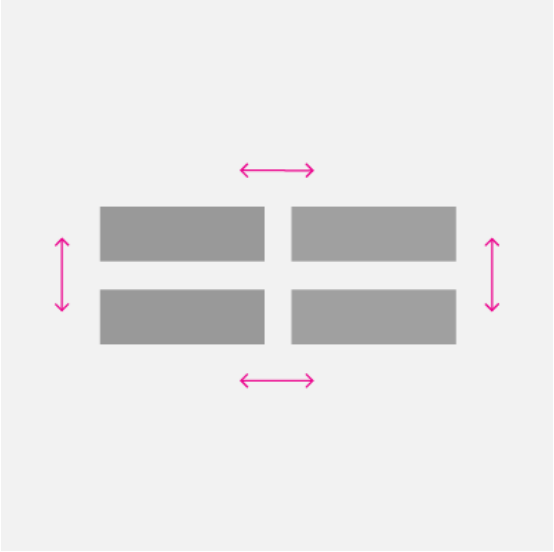
Directional navigation

Directional navigation is managed by a UWP Focus Manager helper class, which takes the directional key pressed (arrow key, D-pad) and attempts to move focus in the corresponding visual direction.

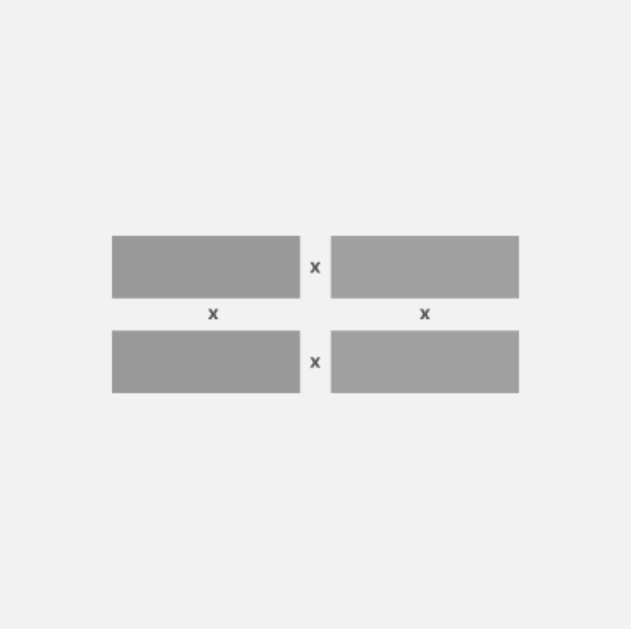
Unlike the keyboard, when an app opts out of Mouse Mode, directional navigation is applied across the entire application for gamepad and remote control. See Gamepad and remote control interactions for more detail on directional navigation optimization.

NOTE Navigation using the keyboard Tab key is not considered directional navigation. For more info, see the Tab stops section.

Expand table



Directional navigation supported
Using directional keys (keyboard arrows, gamepad and remote control D-pad), user can navigate between different controls.



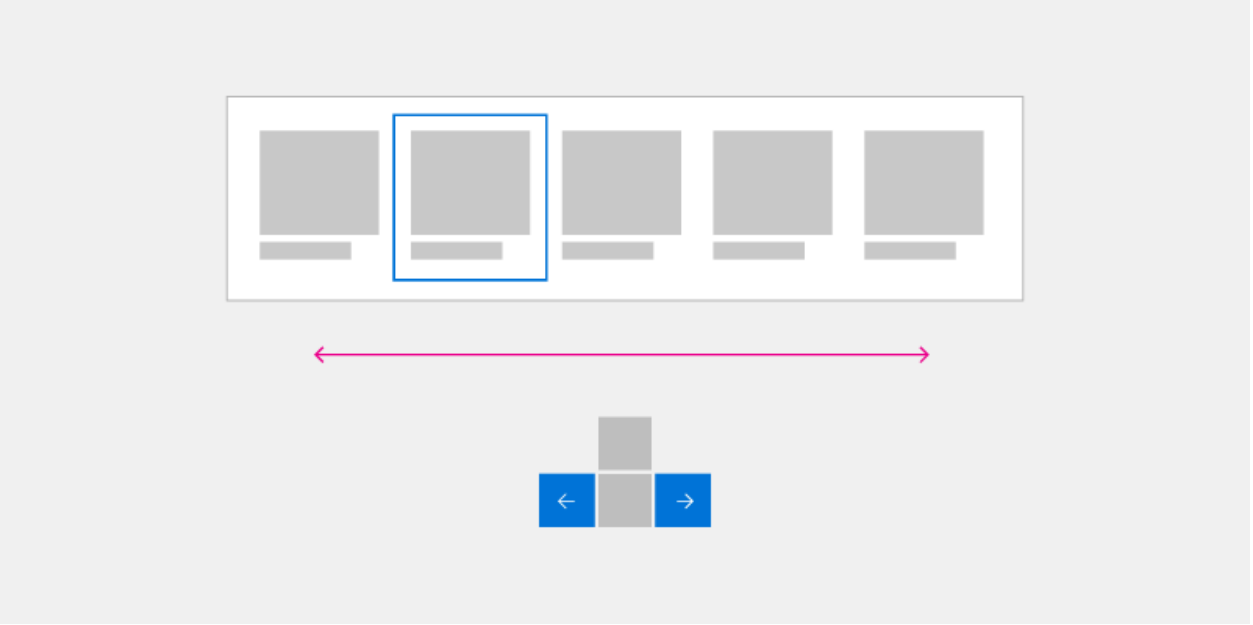
Directional navigation not supported
User cannot navigate between different controls using directional keys. Other methods of navigating between controls (tab key) are not impacted.

Built in keyboard optimization

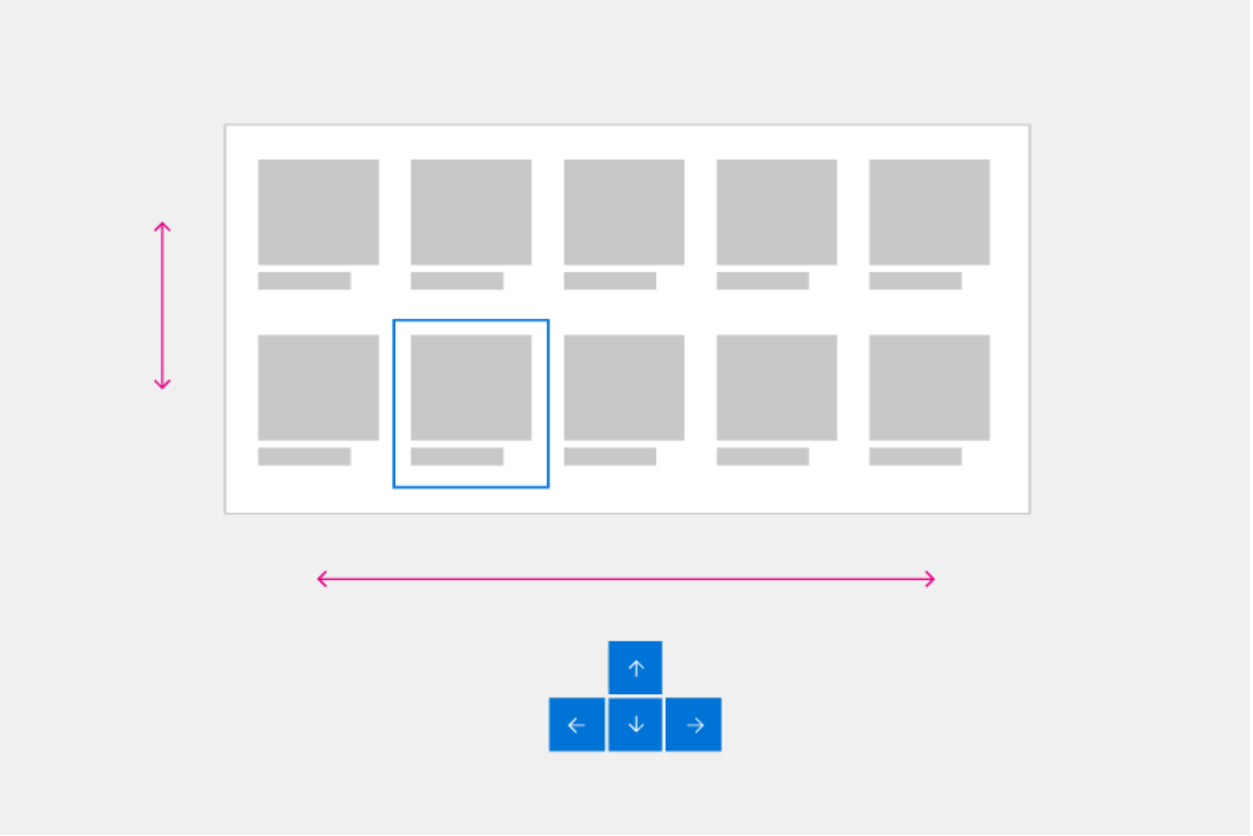
Depending on the layout and controls used, UWP apps can be optimized specifically for keyboard input.

The following example shows a group of list items, grid items, and menu items that have been assigned to a single tab stop (see the Tab stops section). When the group has focus, inner navigation is performed with the directional arrow keys in the corresponding visual order (see Navigation section).

Single Column Arrow Key Navigation



Single Row Arrow Key Navigation



Multiple Column/Row Arrow Key Navigation

Wrapping homogeneous List and Grid View Items

Directional navigation is not always the most efficient way to navigate multiple rows and columns of List and GridView items.

NOTE Menu items are typically single column lists, but special focus rules might apply in some cases (see Popup UI).

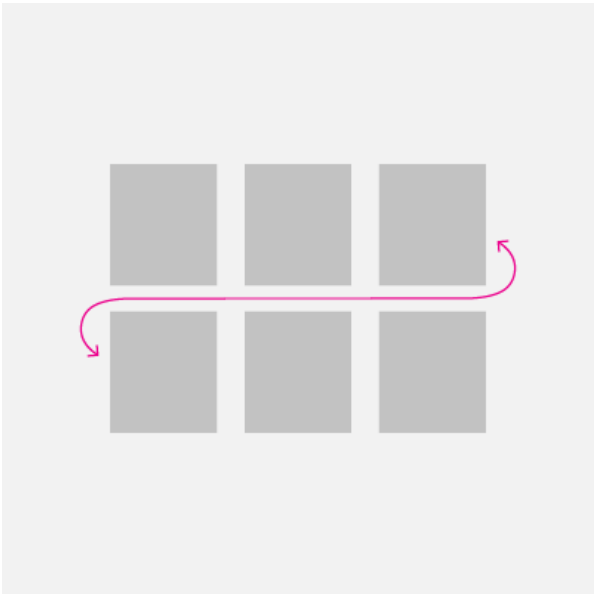
List and Grid objects can be created with multiple rows and columns. These are typically in row-major (where items fill entire row first before filling in the next row) or column-major (where items fill entire column first before filling in the next column) order. Row or column major order depends on scroll direction and you should ensure that item order does not conflict with this direction.

In row-major order (where items fill in left to right, top to bottom), when the focus is on the last item in a row and the Right arrow key is pressed, focus is moved to the first item in the next row. This same behavior occurs in reverse: When focus is set to the first item in a row and the Left arrow key is pressed, focus is moved to the last item in the previous row.

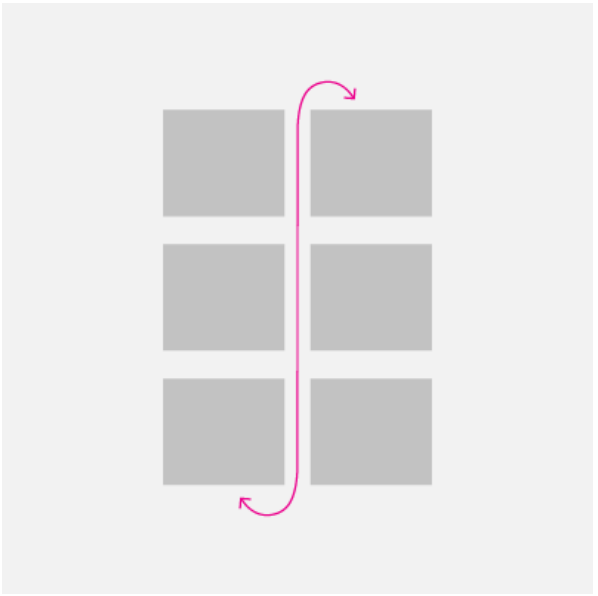
In column-major order (where items fill in top to bottom, left to right), when the focus is on the last item in a column and user presses the Down arrow key, focus is moved to the first item in the next column. This same behavior occurs in reverse: When focus is set

to the first item in a column and the Up arrow key is pressed, focus is moved to the last item in the previous column.

Expand table



Row major keyboard navigation



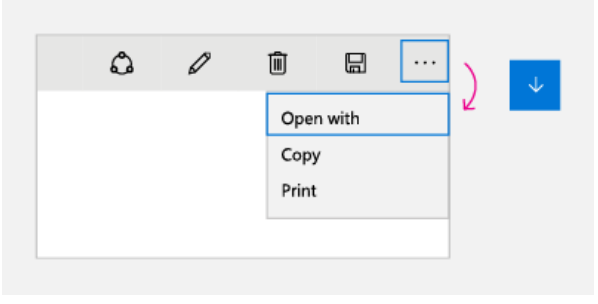
Column major keyboard navigation

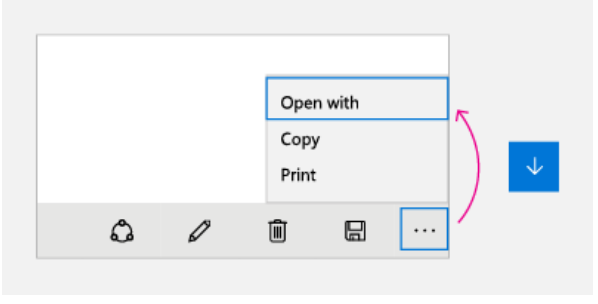
Popup UI

As mentioned, you should try to ensure directional navigation corresponds to the visual order of the controls in your application's UI.

Some controls (such as the context menu, CommandBar overflow menu, and AutoSuggest menu) display a menu popup in a location and direction (downwards by default) relative to the primary control and available screen space. Note that the opening direction can be affected by a variety of factors at run time.

Expand table





For these controls, when the menu is first opened (and no item has been selected by the user), the Down arrow key always sets focus to the first item while the Up arrow key always sets focus to the last item on the menu.

If the last item has focus and the Down arrow key is pressed, focus moves to the first item on the menu. Similarly, if the first item has focus and the Up arrow key is pressed, focus moves to the last item on the menu. This behavior is referred to as *cycling* and is useful for navigating popup menus that can open in unpredictable directions.

> **Note**
>
> Cycling should be avoided in non-popup UIs where users might come to feel trapped in an endless loop.

We recommend that you emulate these same behaviors in your custom controls. Code sample on how to implement this behavior can be found in Programmatic focus navigation documentation.

Test your app

Test your app with all supported input devices to ensure UI elements can be navigated to in a coherent and intuitive way and that no unexpected elements interfere with the desired tab order.

Related articles

- Keyboard events
- Identify input devices
- Respond to the presence of the touch keyboard
- Focus visuals sample
- NavigationView control keyboarding specifics
- Keyboard accessibility

Appendix

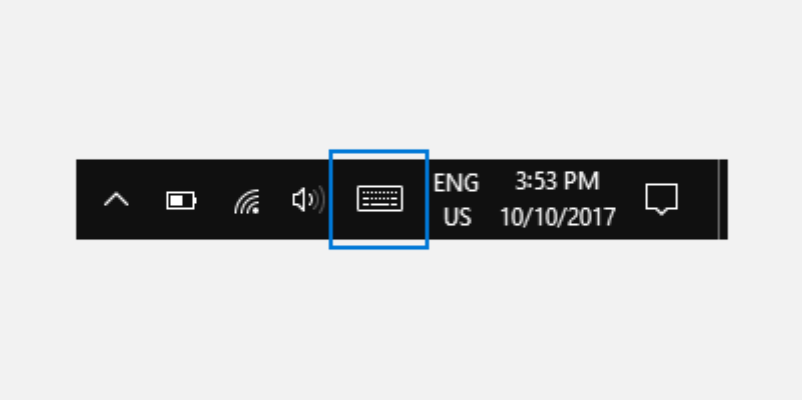
Software keyboard

A software keyboard is displayed on screen and used instead of the physical keyboard to type and enter data using touch, mouse, pen/stylus or other pointing device. On gaming devices, individual keys need to be selected by moving focus visual or using shortcut keys on a game pad or remote control.

Touch keyboard

Windows 11 Touch Keyboard

Depending on the device, the touch keyboard appears when a text field or other editable text control gets focus, or when the user manually enables it through the **Notification Center**:



If your app sets focus programmatically to a text input control, the touch keyboard is not invoked. This eliminates unexpected behaviors not instigated directly by the user. However, the keyboard does automatically hide when focus is moved programmatically to a non-text input control.

The touch keyboard typically remains visible while the user navigates between controls in a form. This behavior can vary based on the other control types within the form.

The following is a list of non-edit controls that can receive focus during a text entry session using the touch keyboard without dismissing the keyboard. Rather than needlessly churn the UI and potentially disorient the user, the touch keyboard remains in view because the user is likely to go back and forth between these controls and text entry with the touch keyboard.

- Check box
- Combo box
- Radio button
- Scroll bar
- Tree
- Tree item
- Menu
- Menu bar
- Menu item
- Toolbar
- List
- List item

Here are examples of different modes for the touch keyboard. The first image is the default layout, the second is the expanded layout (which might not be available in all languages).

The touch keyboard in default layout mode

The touch keyboard in expanded layout mode

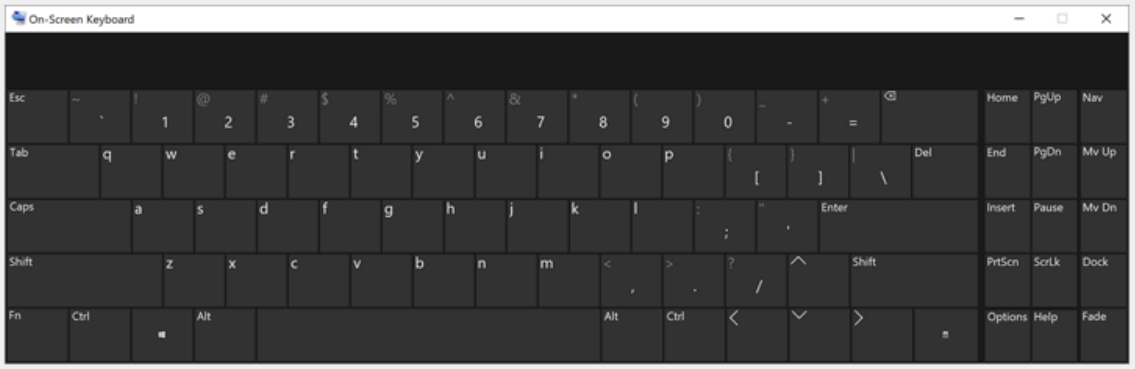
Successful keyboard interactions enable users to accomplish basic app scenarios using only the keyboard; that is, users can reach all interactive elements and activate default functionality. A number of factors can affect the degree of success, including keyboard navigation, access keys for accessibility, and accelerator (or shortcut) keys for advanced users.

On-Screen Keyboard

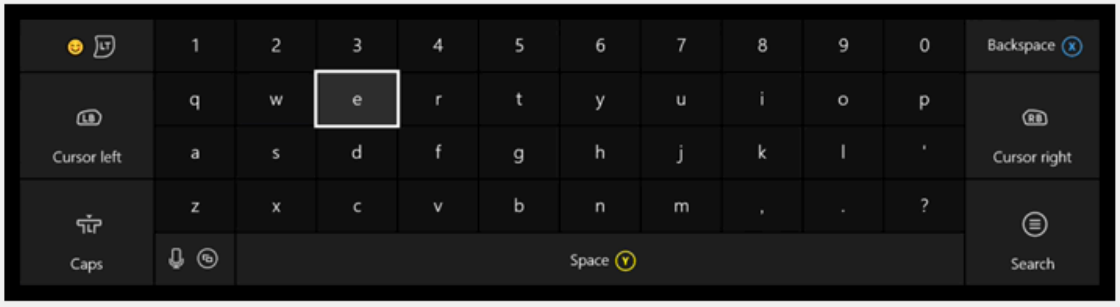
Like the touch keyboard, the On-Screen Keyboard (OSK) is a visual, software keyboard used instead of the physical keyboard to type and enter data using touch, mouse, pen/stylus or other pointing device (a touch screen is not required). The OSK is provided for systems that don't have a physical keyboard, or for users whose mobility impairments prevent them from using traditional physical input devices. The OSK emulates most, if not all, the functionality of a hardware keyboard.

The OSK can be turned on from the Keyboard page in Settings > Ease of access.

NOTE The OSK has priority over the touch keyboard, which won't be shown if the OSK is present.



On-Screen Keyboard



Xbox One On-Screen Keyboard

For more details, see Use the On-Screen Keyboard to type.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Access keys

Article • 06/11/2024

Access keys are keyboard shortcuts that improve the usability and the accessibility of your Windows applications by providing an intuitive way for users to quickly navigate and interact with an app's visible UI through a keyboard instead of a pointer device (such as touch or mouse).

The Windows app provides built-in support across platform controls for both keyboard-based access keys and associated UI feedback through visual cues called keytips.

> **Note**
>
> A keyboard is indispensable for users with certain disabilities (see Keyboard accessibility), and is also an important tool for users who prefer it as a more efficient way to interact with an app.

See the Accelerator keys topic for details on invoking common actions in a Windows application with keyboard shortcuts.

To create your own custom keyboard shortcuts, see the Keyboard events topic.

Overview

An access key is a combination of the Alt key and one or more alphanumeric keys—sometimes called a *mnemonic*—typically pressed sequentially, rather than simultaneously.

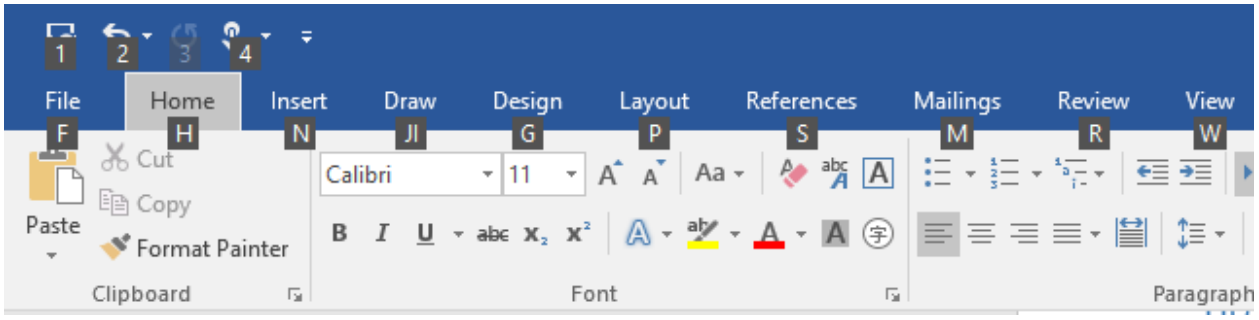
Keytips are badges displayed next to controls that support access keys when the user presses the Alt key. Each keytip contains the alphanumeric keys that activate the associated control.

> **Note**
>
> Keyboard shortcuts are automatically supported for access keys with a single alphanumeric character. For example, simultaneously pressing Alt+F in Word opens the File menu without displaying keytips.

Pressing the Alt key initializes access key functionality and displays all currently available key combinations in keytips. Subsequent keystrokes are handled by the access key

framework, which rejects invalid keys until either a valid access key is pressed, or the Enter, Esc, Tab, or Arrow keys are pressed to deactivate access keys and return keystroke handling to the app.

Microsoft Office apps provide extensive support for access keys. The following image shows the Home tab of Word with access keys activated (note the support for both numbers and multiple keystrokes).



KeyTip badges for access keys in Microsoft Word

To add an access key to a control, use the **AccessKey property**. The value of this property specifies the access key sequence, the shortcut (if a single alphanumeric), and the keytip.

XAML

```
<Button Content="Accept" AccessKey="A" Click="AcceptButtonClick" />
```

When to use access keys

We recommend that you specify access keys wherever appropriate in your UI, and support access keys in all custom controls.

1. **Access keys make your app more accessible** for users with motor disabilities, including those users who can press only one key at a time or have difficulty using a mouse.

   A well-designed keyboard UI is an important aspect of software accessibility. It enables users with vision impairments or who have certain motor disabilities to navigate an app and interact with its features. Such users might not be able to operate a mouse and instead rely on various assistive technologies such as keyboard enhancement tools, on-screen keyboards, screen enlargers, screen readers, and voice input utilities. For these users, comprehensive command coverage is crucial.

2. **Access keys make your app more usable** for power users who prefer to interact through the keyboard.

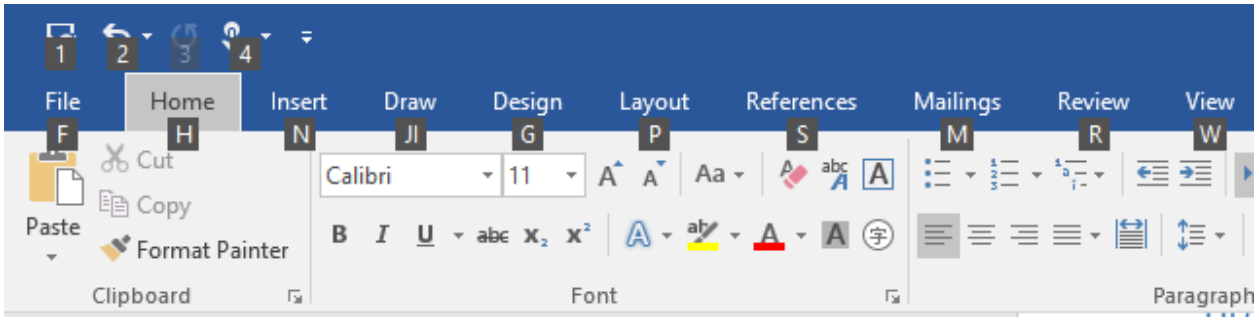
   Experienced users often have a strong preference for using the keyboard because keyboard-based commands can be entered more quickly and don't require them to remove their hands from the keyboard. For these users, efficiency and consistency are crucial; comprehensiveness is important only for the most frequently used commands.

Set access key scope

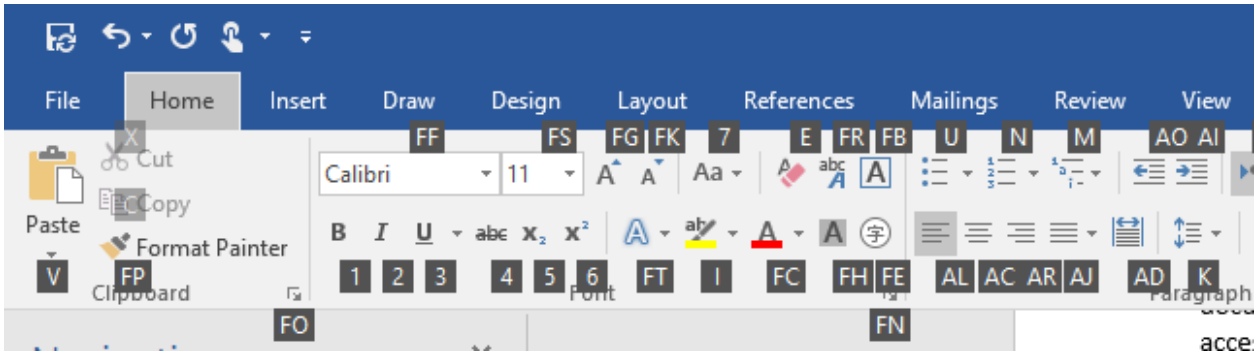
When there are many elements on the screen that support access keys, we recommend scoping the access keys to reduce **cognitive load**. This minimizes the number of access keys on the screen, which makes them easier to locate, and improves efficiency and productivity.

For example, Microsoft Word provides two access key scopes: a primary scope for the Ribbon tabs and a secondary scope for commands on the selected tab.

The following images demonstrate the two access key scopes in Word. The first shows the primary access keys that let a user select a tab and other top level commands, and the second shows the secondary access keys for the Home tab.



Primary access keys in Microsoft Word



Secondary access keys in Microsoft Word

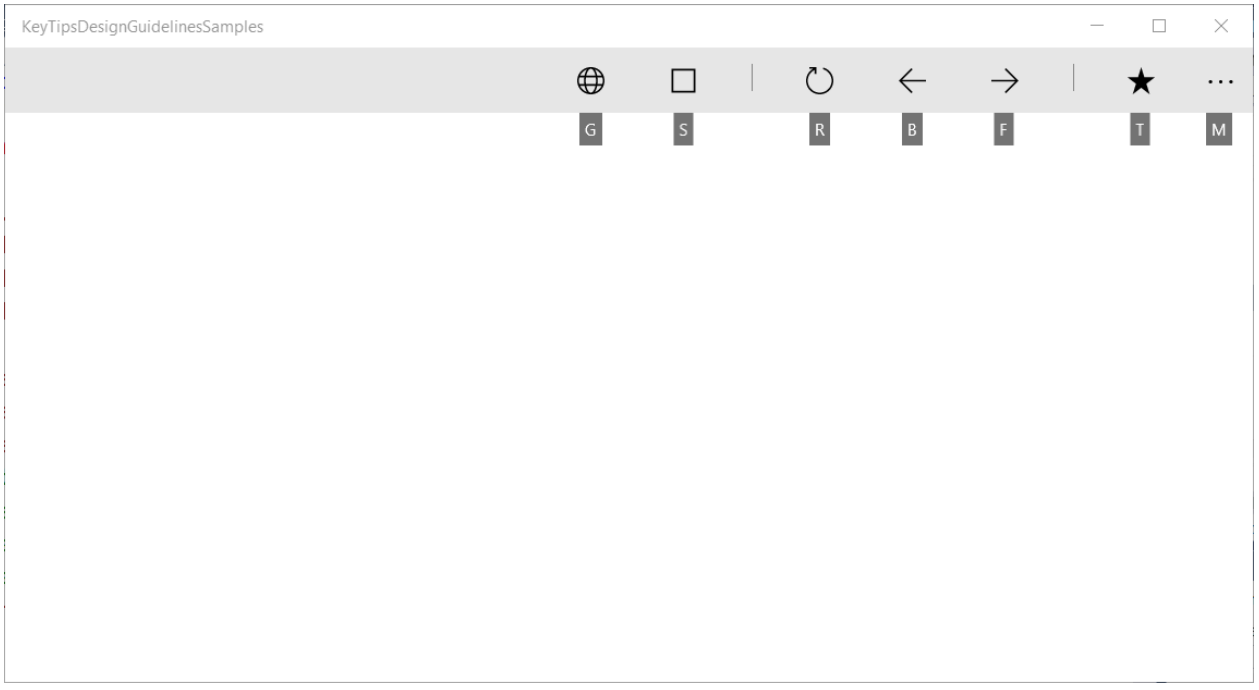
Access keys can be duplicated for elements in different scopes. In the preceding example, "2" is the access key for Undo in the primary scope, and also "Italics" in the

secondary scope.

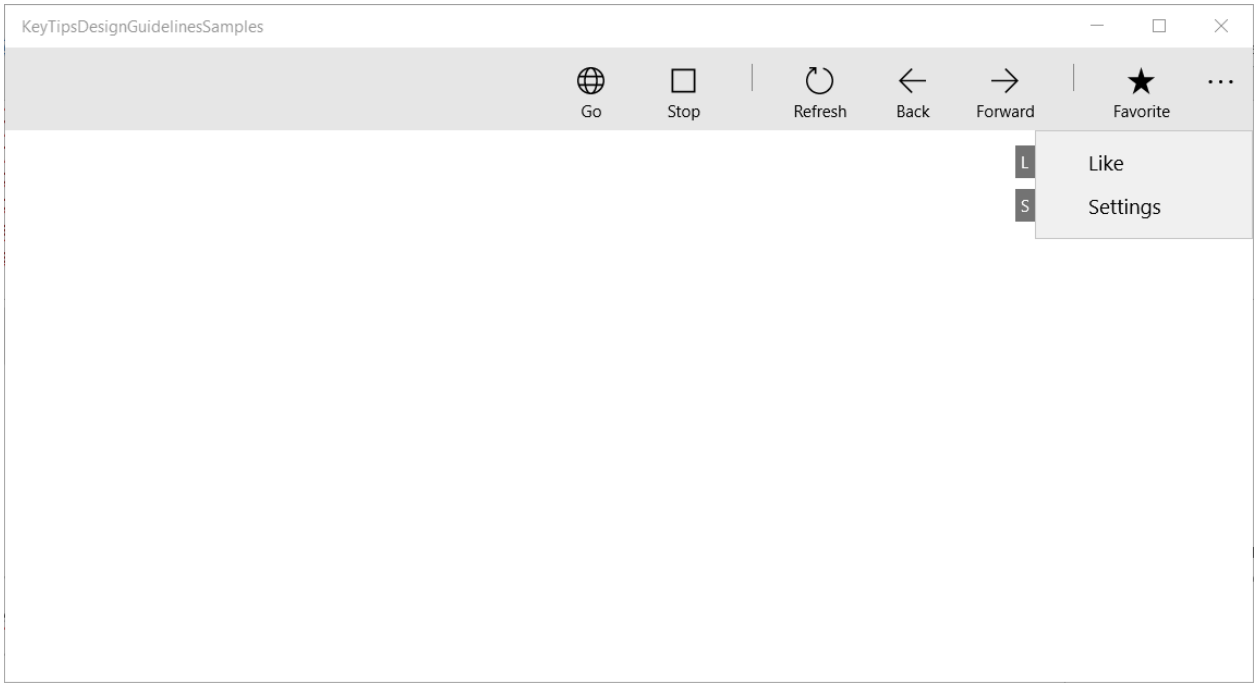
Here, we show how to define an access key scope.

XAML

```
<CommandBar x:Name="MainCommandBar" AccessKey="M" >
    <AppBarButton AccessKey="G" Icon="Globe" Label="Go"/>
    <AppBarButton AccessKey="S" Icon="Stop" Label="Stop"/>
    <AppBarSeparator/>
    <AppBarButton AccessKey="R" Icon="Refresh" Label="Refresh"
IsAccessKeyScope="True">
        <AppBarButton.Flyout>
            <MenuFlyout>
                <MenuFlyoutItem AccessKey="A" Icon="Globe" Text="Refresh A"
/>
                <MenuFlyoutItem AccessKey="B" Icon="Globe" Text="Refresh B"
/>
                <MenuFlyoutItem AccessKey="C" Icon="Globe" Text="Refresh C"
/>
                <MenuFlyoutItem AccessKey="D" Icon="Globe" Text="Refresh D"
/>
            </MenuFlyout>
        </AppBarButton.Flyout>
    </AppBarButton>
    <AppBarButton AccessKey="B" Icon="Back" Label="Back"/>
    <AppBarButton AccessKey="F" Icon="Forward" Label="Forward"/>
    <AppBarSeparator/>
    <AppBarToggleButton AccessKey="T" Icon="Favorite" Label="Favorite"/>
    <CommandBar.SecondaryCommands>
        <AppBarToggleButton Icon="Like" AccessKey="L" Label="Like"/>
        <AppBarButton Icon="Setting" AccessKey="S" Label="Settings" />
    </CommandBar.SecondaryCommands>
</CommandBar>
```



CommandBar primary scope and supported access keys



CommandBar secondary scope and supported access keys

Windows 10 Creators Update and older

Prior to Windows 10 Fall Creators Update, some controls, such as the CommandBar, didn't support built-in access key scopes.

The following example shows how to support CommandBar SecondaryCommands with access keys, which are available once a parent command is invoked (similar to the Ribbon in Word).

XAML

```
<local:CommandBarHack x:Name="MainCommandBar" AccessKey="M" >
    <AppBarButton AccessKey="G" Icon="Globe" Label="Go"/>
    <AppBarButton AccessKey="S" Icon="Stop" Label="Stop"/>
    <AppBarSeparator/>
    <AppBarButton AccessKey="R" Icon="Refresh" Label="Refresh"
IsAccessKeyScope="True">
        <AppBarButton.Flyout>
            <MenuFlyout>
                <MenuFlyoutItem AccessKey="A" Icon="Globe" Text="Refresh A"
/>
                <MenuFlyoutItem AccessKey="B" Icon="Globe" Text="Refresh B"
/>
                <MenuFlyoutItem AccessKey="C" Icon="Globe" Text="Refresh C"
/>
                <MenuFlyoutItem AccessKey="D" Icon="Globe" Text="Refresh D"
/>
            </MenuFlyout>
```

```
        </AppBarButton.Flyout>
    </AppBarButton>
    <AppBarButton AccessKey="B" Icon="Back" Label="Back"/>
    <AppBarButton AccessKey="F" Icon="Forward" Label="Forward"/>
    <AppBarSeparator/>
    <AppBarToggleButton AccessKey="T" Icon="Favorite" Label="Favorite"/>
    <CommandBar.SecondaryCommands>
        <AppBarToggleButton Icon="Like" AccessKey="L" Label="Like"/>
        <AppBarButton Icon="Setting" AccessKey="S" Label="Settings" />
    </CommandBar.SecondaryCommands>
</local:CommandBarHack>
```

C#

```
public class CommandBarHack : CommandBar
{
    CommandBarOverflowPresenter secondaryItemsControl;
    Popup overflowPopup;

    public CommandBarHack()
    {
        this.ExitDisplayModeOnAccessKeyInvoked = false;
        AccessKeyInvoked += OnAccessKeyInvoked;
    }

    protected override void OnApplyTemplate()
    {
        base.OnApplyTemplate();

        Button moreButton = GetTemplateChild("MoreButton") as Button;
        moreButton.SetValue(Control.IsTemplateKeyTipTargetProperty, true);
        moreButton.IsAccessKeyScope = true;

        // SecondaryItemsControl changes
        secondaryItemsControl = GetTemplateChild("SecondaryItemsControl") as
CommandBarOverflowPresenter;
        secondaryItemsControl.AccessKeyScopeOwner = moreButton;

        overflowPopup = GetTemplateChild("OverflowPopup") as Popup;
    }

    private void OnAccessKeyInvoked(UIElement sender,
AccessKeyInvokedEventArgs args)
    {
        if (overflowPopup != null)
        {
            overflowPopup.Opened += SecondaryMenuOpened;
        }
    }

    private void SecondaryMenuOpened(object sender, object e)
    {
        //This is not necessary given we are automatically pushing the
```

```
scope.
        var item = secondaryItemsControl.Items.First();
        if (item != null && item is Control)
        {
            (item as Control).Focus(FocusState.Keyboard);
        }
        overflowPopup.Opened -= SecondaryMenuOpened;
    }
}
```

Avoid access key collisions

Access key collisions occur when two or more elements in the same scope have duplicate access keys, or start with the same alphanumeric characters.

The system resolves duplicate access keys by processing the access key of the first element added to the visual tree, ignoring all others.

When multiple access keys start with the same character (for example, "A", "A1", and "AB"), the system processes the single character access key and ignores all others.

Avoid collisions by using unique access keys or by scoping commands.

Choose access keys

Consider the following when choosing access keys:

- Use a single character to minimize keystrokes and support accelerator keys by default (Alt+AccessKey)
- Avoid using more than two characters
- Avoid access keys collisions
- Avoid characters that are difficult to differentiate from other characters, such as the letter "I" and the number "1" or the letter "O" and the number "0"
- Use well-known precedents from other popular apps such as Word ("F" for "File", "H" for "Home", and so on)
- Use the first character of the command name, or a character with a close association to the command that helps with recall
  - If the first letter is already assigned, use a letter that is as close as possible to the first letter of the command name ("N" for Insert)
  - Use a distinctive consonant from the command name ("W" for View)
  - Use a vowel from the command name.

Localize access keys

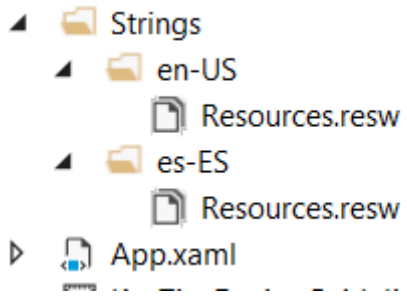
If your app is going to be localized in multiple languages, you should also **consider localizing the access keys**. For example, for "H" for "Home" in en-US and "I" for "Incio" in es-ES.

Use the x:Uid extension in markup to apply localized resources as shown here:

XAML

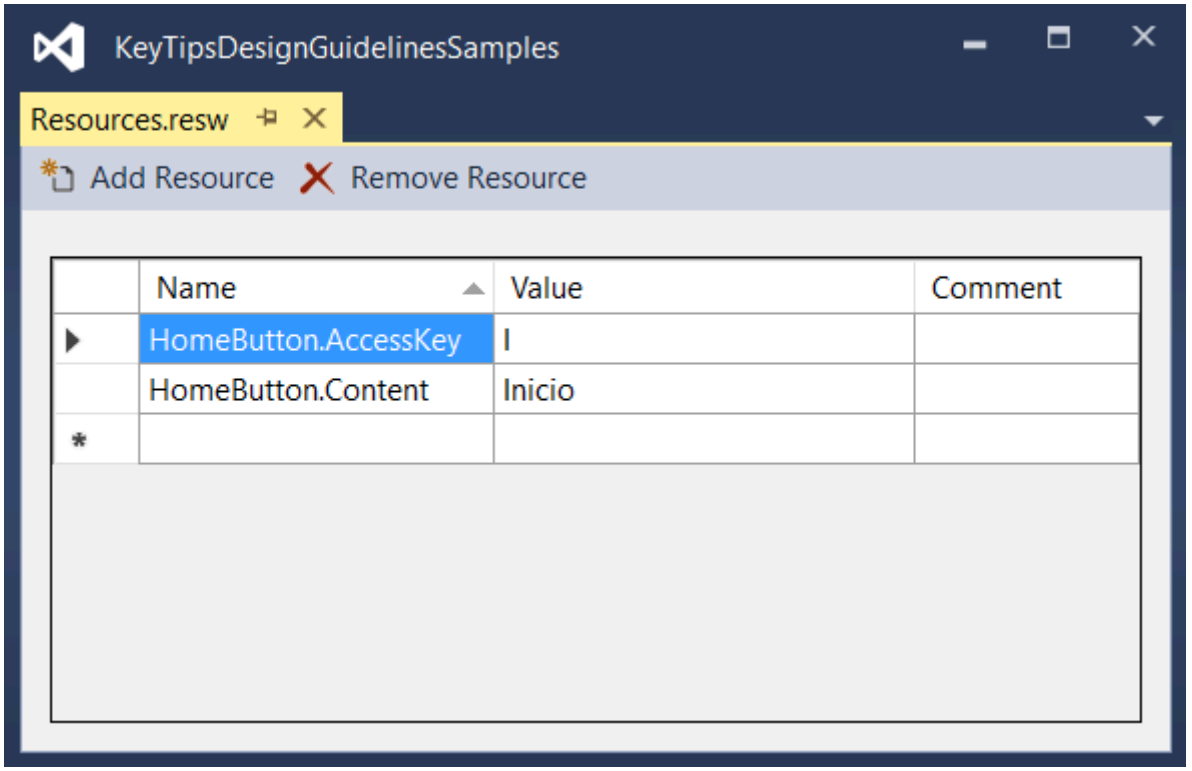
```
<Button Content="Home" AccessKey="H" x:Uid="HomeButton" />
```

Resources for each language are added to corresponding String folders in the project:



English and Spanish resource string folders

Localized access keys are specified in your projects resources.resw file:



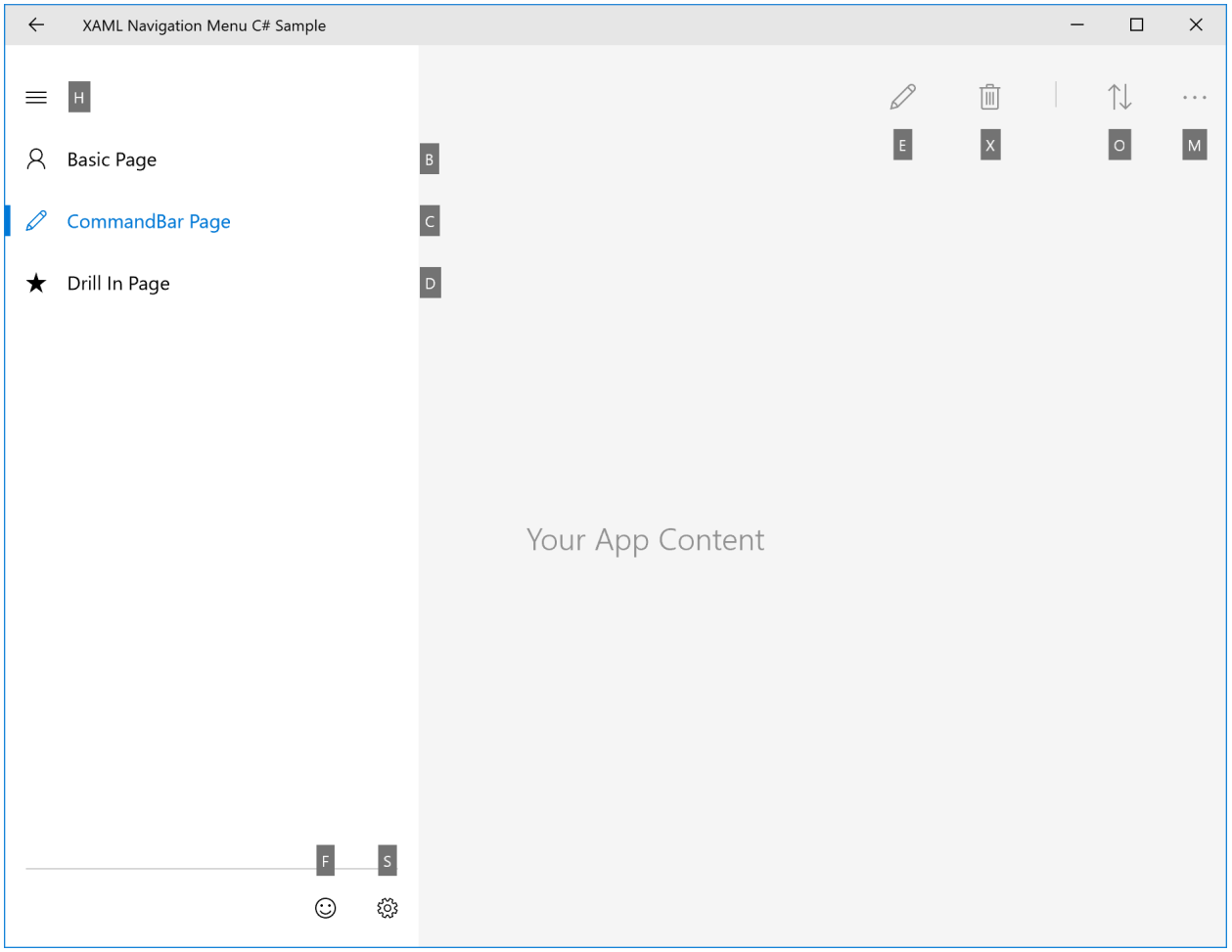
Specify the AccessKey property specified in the resources.resw file

For more info, see Translating UI resources

Keytip positioning

Keytips are displayed as floating badges relative to their corresponding UI element, taking into account the presence of other UI elements, other keytips, and the screen edge.

Typically, the default keytip location is sufficient and provides built-in support for adaptive UI.



Example of automatic keytip placement

However, should you need more control over keytip positioning, we recommend the following:

1. **Obvious association principle**: The user can associate the control with the keytip easily.

   a. The keytip should be **close** to the element who have the access key (the owner).
   b. The keytip should **avoid covering enabled elements** that have access keys.
   c. If a keytip can't be placed close to its owner, it should overlap the owner.

2. **Discoverability**: The user can discover the control with the keytip quickly.

    a. The keytip never **overlaps** other keytips.

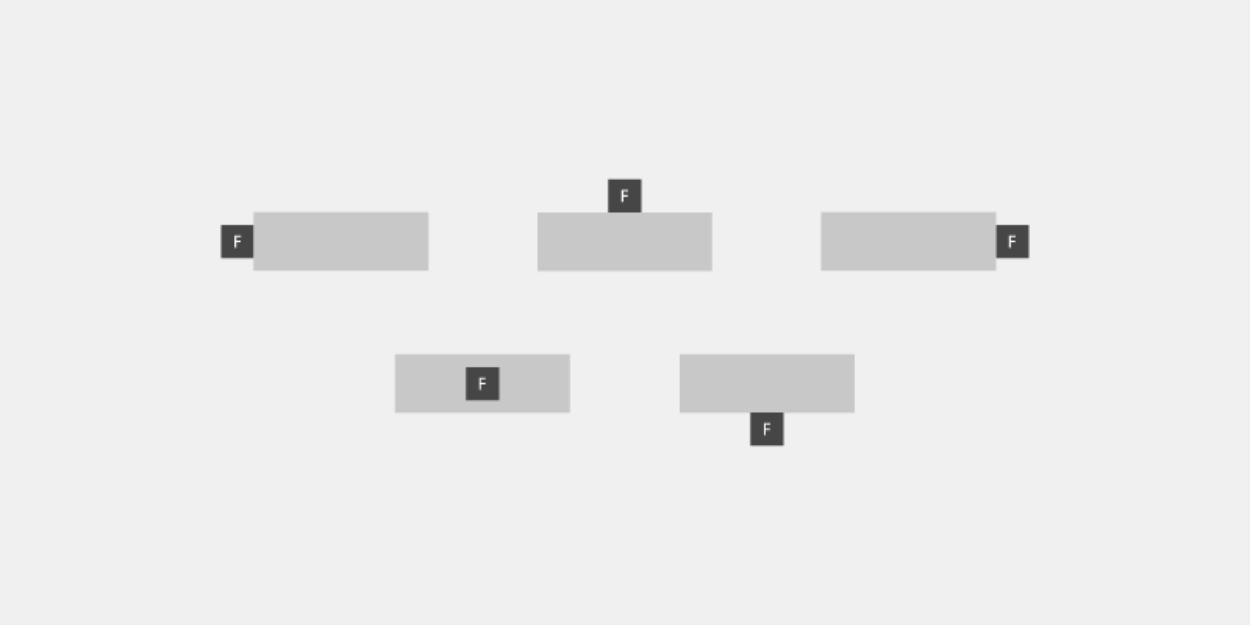
3. **Easy scanning:** The user can skim the keytips easily.

    a. keytips should be **aligned** with each other and with the UI Element. b. keytips should be **grouped** as much as possible.

Relative position

Use the **KeyTipPlacementMode** property to customize the placement of the keytip on a per element or per group basis.

The placement modes are: Top, Bottom, Right, Left, Hidden, Center, and Auto.



keytip placement modes

The center line of the control is used to calculate the vertical and horizontal alignment of the keytip.

They following example shows how to set the keytip placement of a group of controls using the KeyTipPlacementMode property of a StackPanel container.

XAML

```
<StackPanel Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
KeyTipPlacementMode="Top">
  <Button Content="File" AccessKey="F" />
  <Button Content="Home" AccessKey="H" />
```

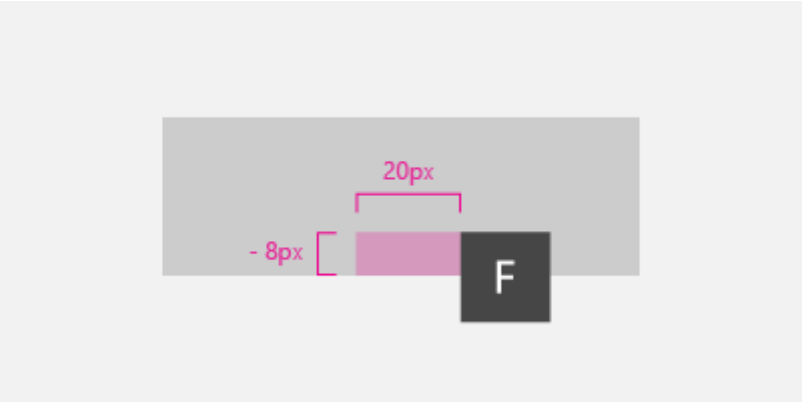
```
  <Button Content="Insert" AccessKey="N" />
</StackPanel>
```

Offsets

Use the KeyTipHorizontalOffset and KeyTipVerticalOffset properties of an element for even more granular control of the keytip location.

> **Note**
>
> Offsets cannot be set when KeyTipPlacementMode is set to Auto.

The KeyTipHorizontalOffset property indicates how far to move the keytip left or right.



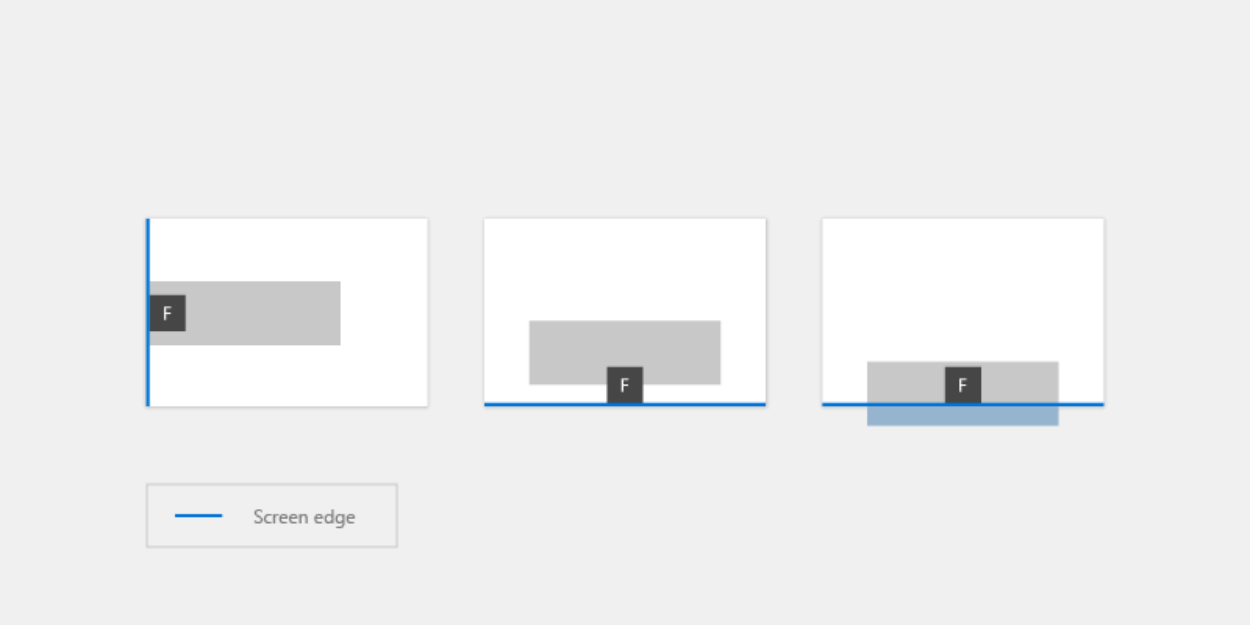
Set vertical and horizontal keytip offsets for a button

XAML

```
<Button
  Content="File"
  AccessKey="F"
  KeyTipPlacementMode="Bottom"
  KeyTipHorizontalOffset="20"
  KeyTipVerticalOffset="-8" />
```

Screen edge alignment {#screen-edge-alignment .ListParagraph}

The location of a keytip is automatically adjusted based on the screen edge to ensure the keytip is fully visible. When this occurs, the distance between the control and keytip alignment point might differ from the values specified for the horizontal and vertical offsets .

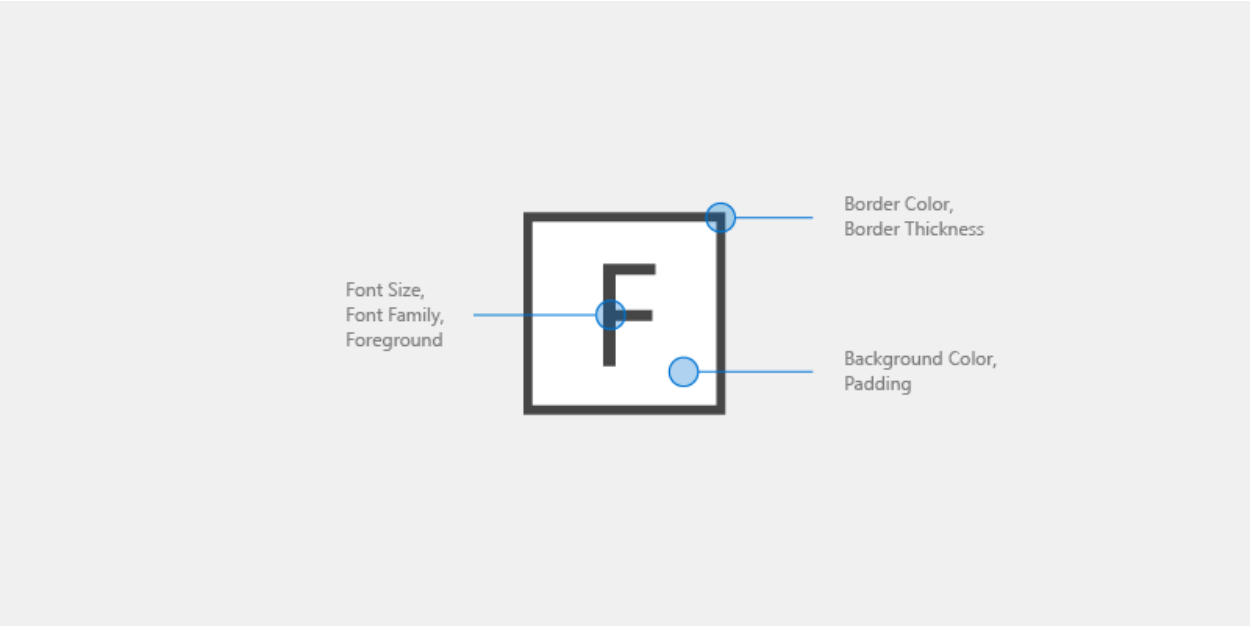


keytips are automatically positioned based on the screen edge

Keytip style

We recommend using the built-in keytip support for platform themes, including high contrast.

If you need to specify your own keytip styles, use application resources such as KeyTipFontSize (font size), KeyTipFontFamily (font family), KeyTipBackground (background), KeyTipForeground (foreground), KeyTipPadding (padding), KeyTipBorderBrush(Border color), and KeyTipBorderThemeThickness (border thickness).



keytip customization options

This example demonstrates how to change these application resources:

XAML

```
<Application.Resources>
 <SolidColorBrush Color="DarkGray" x:Key="MyBackgroundColor" />
 <SolidColorBrush Color="White" x:Key="MyForegroundColor" />
 <SolidColorBrush Color="Black" x:Key="MyBorderColor" />
 <StaticResource x:Key="KeyTipBackground" ResourceKey="MyBackgroundColor" />
 <StaticResource x:Key="KeyTipForeground" ResourceKey="MyForegroundColor" />
 <StaticResource x:Key="KeyTipBorderBrush" ResourceKey="MyBorderColor"/>
 <FontFamily x:Key="KeyTipFontFamily">Consolas</FontFamily>
 <x:Double x:Key="KeyTipContentThemeFontSize">18</x:Double>
 <Thickness x:Key="KeyTipBorderThemeThickness">2</Thickness>
 <Thickness x:Key="KeyTipThemePadding">4,4,4,4</Thickness>
</Application.Resources>
```

Access keys and Narrator

The XAML framework exposes Automation Properties that enable UI Automation clients to discover information about elements in the user interface.

If you specify the AccessKey property on a UIElement or TextElement control, you can use the AutomationProperties.AccessKey property to get this value. Accessibility clients, such as Narrator, read the value of this property each time an element gets focus.

Related articles

- Keyboard interactions
- Keyboard accelerators

Samples

- WinUI Gallery

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Keyboard accelerators

Article • 05/17/2022

Accelerator keys (or keyboard accelerators) are keyboard shortcuts that improve the usability and accessibility of your Windows applications by providing an intuitive way for users to invoke common actions or commands without navigating the app UI.

> **Note**
>
> A keyboard is indispensable for users with certain disabilities (see Keyboard accessibility), and is also an important tool for users who prefer it as a more efficient way to interact with an app.

See the Access keys topic for details on navigating the UI of a Windows application with keyboard shortcuts.

To create your own custom keyboard shortcuts, see the Keyboard events topic.

Overview

Accelerators are composed of two types of keys: modifiers and non-modifiers. Modifier keys include Shift, Menu, Control, and the Windows key, which are exposed through VirtualKeyModifiers. Non-modifiers include any VirtualKey, such as Delete, F3, Spacebar, Arrow, Esc, and all alphanumeric and punctuation keys.

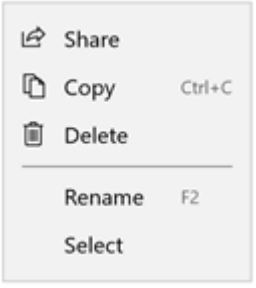
> **Note**
>
> Accelerators typically include the function keys F1 through F12 or some combination of a standard key paired with one or more modifier keys (CTRL, Shift).

For example, if a user presses Ctrl+Shift+M, the framework checks the modifiers (Ctrl and Shift) and fires the accelerator, if it exists.

Many XAML controls have built-in keyboard accelerators. For example, ListView supports Ctrl+A for selecting all the items in the list, and RichEditBox supports Ctrl+Tab for inserting a Tab in the text box. These built-in keyboard accelerators are referred to as **control accelerators** and are executed only if the focus is on the element or one of its children. Accelerators defined by you using the keyboard accelerator APIs discussed here are referred to as **app accelerators**.

Keyboard accelerators are not available for every action but are often associated with commands exposed in menus (and should be specified with the menu item content). Accelerators can also be associated with actions that do not have equivalent menu items. However, because users rely on an application's menus to discover and learn the available command set, you should try to make discovery of accelerators as easy as possible (using labels or established patterns can help with this).

An accelerator auto-repeats (for example, when the user presses Ctrl+Shift and then holds down M, the accelerator is invoked repeatedly until M is released). This behavior cannot be modified.



Keyboard accelerators described in a menu item label

When to use keyboard accelerators

We recommend that you specify keyboard accelerators wherever appropriate in your UI, and support accelerators in all custom controls.

- Keyboard accelerators make your app more accessible for users with motor disabilities, including those users who can press only one key at a time or have difficulty using a mouse.

  A well-designed keyboard UI is an important aspect of software accessibility. It enables users with vision impairments or who have certain motor disabilities to navigate an app and interact with its features. Such users might not be able to operate a mouse and instead rely on various assistive technologies such as

keyboard enhancement tools, on-screen keyboards, screen enlargers, screen readers, and voice input utilities. For these users, comprehensive command coverage is crucial.

- Keyboard accelerators make your app more usable for power users who prefer to interact through the keyboard.

  Experienced users often have a strong preference for using the keyboard because keyboard-based commands can be entered more quickly and don't require them to remove their hands from the keyboard. For these users, efficiency and consistency are crucial; comprehensiveness is important only for the most frequently used commands.

Specify a keyboard accelerator

Use the KeyboardAccelerator APIs to create keyboard accelerators in Windows apps. With these APIs, you don't have to handle multiple KeyDown events to detect the key combination pressed, and you can localize accelerators in the app resources.

We recommend that you set keyboard accelerators for the most common actions in your app and document them using the menu item label or tooltip. In this example, we declare keyboard accelerators only for the Rename and Copy commands.

XAML

```
<CommandBar Margin="0,200" AccessKey="M">
  <AppBarButton
    Icon="Share"
    Label="Share"
    Click="OnShare"
    AccessKey="S" />
  <AppBarButton
    Icon="Copy"
    Label="Copy"
    ToolTipService.ToolTip="Copy (Ctrl+C)"
    Click="OnCopy"
    AccessKey="C">
    <AppBarButton.KeyboardAccelerators>
      <KeyboardAccelerator
        Modifiers="Control"
        Key="C" />
    </AppBarButton.KeyboardAccelerators>
  </AppBarButton>

  <AppBarButton
    Icon="Delete"
    Label="Delete"
    Click="OnDelete"
```

```
      AccessKey="D" />
    <AppBarSeparator/>
    <AppBarButton
      Icon="Rename"
      Label="Rename"
      ToolTipService.ToolTip="Rename (F2)"
      Click="OnRename"
      AccessKey="R">
      <AppBarButton.KeyboardAccelerators>
        <KeyboardAccelerator
          Modifiers="None" Key="F2" />
      </AppBarButton.KeyboardAccelerators>
    </AppBarButton>

    <AppBarButton
      Icon="SelectAll"
      Label="Select"
      Click="OnSelect"
      AccessKey="A" />

    <CommandBar.SecondaryCommands>
      <AppBarButton
        Icon="OpenWith"
        Label="Sources"
        AccessKey="S">
        <AppBarButton.Flyout>
          <MenuFlyout>
            <ToggleMenuFlyoutItem Text="OneDrive" />
            <ToggleMenuFlyoutItem Text="Contacts" />
            <ToggleMenuFlyoutItem Text="Photos"/>
            <ToggleMenuFlyoutItem Text="Videos"/>
          </MenuFlyout>
        </AppBarButton.Flyout>
      </AppBarButton>
      <AppBarToggleButton
        Icon="Save"
        Label="Auto Save"
        IsChecked="True"
        AccessKey="A"/>
    </CommandBar.SecondaryCommands>

  </CommandBar>
```

Keyboard accelerator described in a tooltip

The UIElement object has a KeyboardAccelerator collection, KeyboardAccelerators, where you specify your custom KeyboardAccelerator objects and define the keystrokes

for the keyboard accelerator:

- **Key** - the VirtualKey used for the keyboard accelerator.
- **Modifiers** – the VirtualKeyModifiers used for the keyboard accelerator. If Modifiers is not set, the default value is None.

> ⓘ **Note**
>
> Single key (A, Delete, F2, Spacebar, Esc, Multimedia Key) accelerators and multi-key accelerators (Ctrl+Shift+M) are supported. However, Gamepad virtual keys are not supported.

Scoped accelerators

Some accelerators work only in specific scopes while others work app-wide.

For example, Microsoft Outlook includes the following accelerators:

- Ctrl+B, Ctrl+I and ESC work only on the scope of the send email form
- Ctrl+1 and Ctrl+2 work app-wide

Context menus

Context menu actions affect only specific areas or elements, such as the selected characters in a text editor or a song in a playlist. For this reason, we recommend setting the scope of keyboard accelerators for context menu items to the parent of the context menu.

Use the ScopeOwner property to specify the scope of the keyboard accelerator. This code demonstrates how to implement a context menu on a ListView with scoped keyboard accelerators:

XAML

```
<ListView x:Name="MyList">
  <ListView.ContextFlyout>
    <MenuFlyout>
      <MenuFlyoutItem Text="Share" Icon="Share"/>
      <MenuFlyoutItem Text="Copy" Icon="Copy">
        <MenuFlyoutItem.KeyboardAccelerators>
          <KeyboardAccelerator
            Modifiers="Control"
            Key="C"
```

```
            ScopeOwner="{x:Bind MyList }" />
        </MenuFlyoutItem.KeyboardAccelerators>
      </MenuFlyoutItem>

      <MenuFlyoutItem Text="Delete" Icon="Delete" />
      <MenuFlyoutSeparator />

      <MenuFlyoutItem Text="Rename">
        <MenuFlyoutItem.KeyboardAccelerators>
          <KeyboardAccelerator
            Modifiers="None"
            Key="F2"
            ScopeOwner="{x:Bind MyList}" />
        </MenuFlyoutItem.KeyboardAccelerators>
      </MenuFlyoutItem>

      <MenuFlyoutItem Text="Select" />
    </MenuFlyout>

  </ListView.ContextFlyout>

  <ListViewItem>Track 1</ListViewItem>
  <ListViewItem>Alternative Track 1</ListViewItem>

</ListView>
```

The ScopeOwner attribute of the MenuFlyoutItem.KeyboardAccelerators element marks the accelerator as scoped instead of global (the default is null, or global). For more detail, see the **Resolving accelerators** section later in this topic.

Invoke a keyboard accelerator

The KeyboardAccelerator object uses the UI Automation (UIA) control pattern to take action when an accelerator is invoked.

The UIA [control patterns] expose common control functionality. For example, the Button control implements the Invoke control pattern to support the Click event (typically a control is invoked by clicking, double-clicking, or pressing Enter, a predefined keyboard shortcut, or some other combination of keystrokes). When a keyboard accelerator is used to invoke a control, the XAML framework looks up whether the control implements the Invoke control pattern and, if so, activates it (it is not necessary to listen for the KeyboardAcceleratorInvoked event).

In the following example, Control+S triggers the Click event because the button implements the Invoke pattern.

XAML

```
<Button Content="Save" Click="OnSave">
  <Button.KeyboardAccelerators>
    <KeyboardAccelerator Key="S" Modifiers="Control" />
  </Button.KeyboardAccelerators>
</Button>
```

If an element implements multiple control patterns, only one can be activated through an accelerator. The control patterns are prioritized as follows:

1. Invoke (Button)
2. Toggle (Checkbox)
3. Selection (ListView)
4. Expand/Collapse (ComboBox)

If no match is identified, the accelerator is invalid and a debug message is provided ("No automation patterns for this component found. Implement all desired behavior in the Invoked event. Setting Handled to true in your event handler suppresses this message.")

Custom keyboard accelerator behavior

The Invoked event of the KeyboardAccelerator object is fired when the accelerator is executed. The KeyboardAcceleratorInvokedEventArgs event object includes the following properties:

- **Handled** (Boolean): Setting this to true prevents the event triggering the control pattern and stops accelerator event bubbling. The default is false.
- **Element** (DependencyObject): The object associated with the accelerator.
- **KeyboardAccelerator**: The keyboard accelerator used to raise the Invoked event.

Here we demonstrate how to define a collection of keyboard accelerators for items in a ListView, and how to handle the Invoked event for each accelerator.

XAML

```
<ListView x:Name="MyListView">
  <ListView.KeyboardAccelerators>
    <KeyboardAccelerator Key="A" Modifiers="Control,Shift"
Invoked="SelectAllInvoked" />
    <KeyboardAccelerator Key="F5" Invoked="RefreshInvoked"  />
  </ListView.KeyboardAccelerators>
</ListView>
```

C#

```
void SelectAllInvoked(KeyboardAccelerator sender,
KeyboardAcceleratorInvokedEventArgs args)
{
  MyListView.SelectAll();
  args.Handled = true;
}

void RefreshInvoked(KeyboardAccelerator sender,
KeyboardAcceleratorInvokedEventArgs args)
{
  MyListView.SelectionMode = ListViewSelectionMode.None;
  MyListView.SelectionMode = ListViewSelectionMode.Multiple;
  args.Handled = true;
}
```

Override default keyboard behavior

Some controls, when they have focus, support built-in keyboard accelerators that override any app-defined accelerator. For example, when a TextBox has focus, the Control+C accelerator only copies the currently selected text (app-defined accelerators are ignored and no other functionality is executed).

While we don't recommend overriding default control behaviors due to user familiarity and expectations, you can override a control's built-in keyboard accelerator. The following example shows how to override the Control+C keyboard accelerator for a TextBox through the PreviewKeyDown event handler:

C#

```
 private void TextBlock_PreviewKeyDown(object sender, KeyRoutedEventArgs e)
 {
    var ctrlState =
CoreWindow.GetForCurrentThread().GetKeyState(Windows.System.VirtualKey.Contr
ol);
    var isCtrlDown = ctrlState == CoreVirtualKeyStates.Down || ctrlState
        ==  (CoreVirtualKeyStates.Down | CoreVirtualKeyStates.Locked);
    if (isCtrlDown && e.Key == Windows.System.VirtualKey.C)
    {
        // Your custom keyboard accelerator behavior.

        e.Handled = true;
    }
 }
```

Disable a keyboard accelerator

If a control is disabled, the associated accelerator is also disabled. In the following example, because the IsEnabled property of the ListView is set to false, the associated Control+A accelerator can't be invoked.

XAML

```
<ListView >
  <ListView.KeyboardAccelerators>
    <KeyboardAccelerator Key="A"
      Modifiers="Control"
      Invoked="CustomListViewSelecAllInvoked" />
  </ListView.KeyboardAccelerators>

  <TextBox>
    <TextBox.KeyboardAccelerators>
      <KeyboardAccelerator
        Key="A"
        Modifiers="Control"
        Invoked="CustomTextSelecAllInvoked"
        IsEnabled="False" />
    </TextBox.KeyboardAccelerators>
  </TextBox>

<ListView>
```

Parent and child controls can share the same accelerator. In this case, the parent control can be invoked even if the child has focus and its accelerator is disabled.

Screen readers and keyboard accelerators

Screen readers such as Narrator can announce the keyboard accelerator key combination to users. By default, this is each modifier (in the VirtualModifiers enum order) followed by the key (and separated by "+" signs). You can customize this through the AcceleratorKey AutomationProperties attached property. If more than one accelerator is specified, only the first is announced.

In this example, the AutomationProperty.AcceleratorKey returns the string "Control+Shift+A":

XAML

```
<ListView x:Name="MyListView">
  <ListView.KeyboardAccelerators>

    <KeyboardAccelerator
      Key="A"
      Modifiers="Control,Shift"
      Invoked="CustomSelectAllInvoked" />
```

```
    <KeyboardAccelerator
      Key="F5"
      Modifiers="None"
      Invoked="RefreshInvoked" />

  </ListView.KeyboardAccelerators>

</ListView>
```

> **Note**
>
> Setting AutomationProperties.AcceleratorKey doesn't enable keyboard functionality, it only indicates to the UIA framework which keys are used.

Common Keyboard Accelerators

We recommend that you make keyboard accelerators consistent across Windows applications.

Users have to memorize keyboard accelerators and expect the same (or similar) results, but this might not always be possible due to differences in functionality across apps.

Expand table

Editing	Common Keyboard Accelerator
Begin editing mode	Ctrl + E
Select all items in a focused control or window	Ctrl + A
Search and replace	Ctrl + H
Undo	Ctrl + Z
Redo	Ctrl + Y
Delete selection and copy it to the clipboard	Ctrl + X
Copy selection to the clipboard	Ctrl + C, Ctrl + Insert
Paste the contents of the clipboard	Ctrl + V, Shift + Insert
Paste the contents of the clipboard (with options)	Ctrl + Alt + V
Rename an item	F2

Editing	Common Keyboard Accelerator
Add a new item	Ctrl + N
Add a new secondary item	Ctrl + Shift + N
Delete selected item (with undo)	Del, Ctrl+D
Delete selected item (without undo)	Shift + Del
Bold	Ctrl + B
Underline	Ctrl + U
Italic	Ctrl + I
Navigation	
Find content in a focused control or Window	Ctrl + F
Go to the next search result	F3
Go to the next UI pane	F6
Go to the previous UI pane	Shift + F6
Other Actions	
Add favorites	Ctrl + D
Refresh	F5 or Ctrl + R
Zoom In	Ctrl + +
Zoom out	Ctrl + -
Zoom to default view	Ctrl + 0
Save	Ctrl + S
Close	Ctrl + W
Print	Ctrl + P

Notice that some of the combinations are not valid for localized versions of Windows. For example, in the Spanish version of Windows, Ctrl+N is used for bold instead of Ctrl+B. We recommend providing localized keyboard accelerators if the app is localized.

Usability affordances for keyboard accelerators

Tooltips

As keyboard accelerators are not typically described directly in the UI of your Windows application, you can improve discoverability through tooltips, which display automatically when the user moves focus to, presses and holds, or hovers the mouse pointer over a control. The tooltip can identify whether a control has an associated keyboard accelerator and, if so, what the accelerator key combination is.

Windows 10, Version 1803 (April 2018 Update) and newer

By default, when keyboard accelerators are declared, all controls (except MenuFlyoutItem and ToggleMenuFlyoutItem) present the corresponding key combinations in a tooltip.

> ⓘ **Note**
>
> If a control has more than one accelerator defined, only the first is presented.

Accelerator key combo in tooltip

For Button, AppBarButton, and AppBarToggleButton objects, the keyboard accelerator is appended to the control's default tooltip. For MenuFlyoutItem and ToggleMenuFlyoutItem objects, the keyboard accelerator is displayed with the flyout text.

> ⓘ **Note**
>
> Specifying a tooltip (see Button1 in the following example) overrides this behavior.

XAML

```
<StackPanel x:Name="Container" Grid.Row="0" Background="AliceBlue">
    <Button Content="Button1" Margin="20"
            Click="OnSave"
            KeyboardAcceleratorPlacementMode="Auto"
            ToolTipService.ToolTip="Tooltip">
        <Button.KeyboardAccelerators>
            <KeyboardAccelerator  Key="A" Modifiers="Windows"/>
        </Button.KeyboardAccelerators>
```

```
    </Button>
    <Button Content="Button2"  Margin="20"
            Click="OnSave"
            KeyboardAcceleratorPlacementMode="Auto">
        <Button.KeyboardAccelerators>
            <KeyboardAccelerator  Key="B" Modifiers="Windows"/>
        </Button.KeyboardAccelerators>
    </Button>
    <Button Content="Button3"  Margin="20"
            Click="OnSave"
            KeyboardAcceleratorPlacementMode="Auto">
        <Button.KeyboardAccelerators>
            <KeyboardAccelerator  Key="C" Modifiers="Windows"/>
        </Button.KeyboardAccelerators>
    </Button>
</StackPanel>
```

Accelerator key combo appended to Button's default tooltip

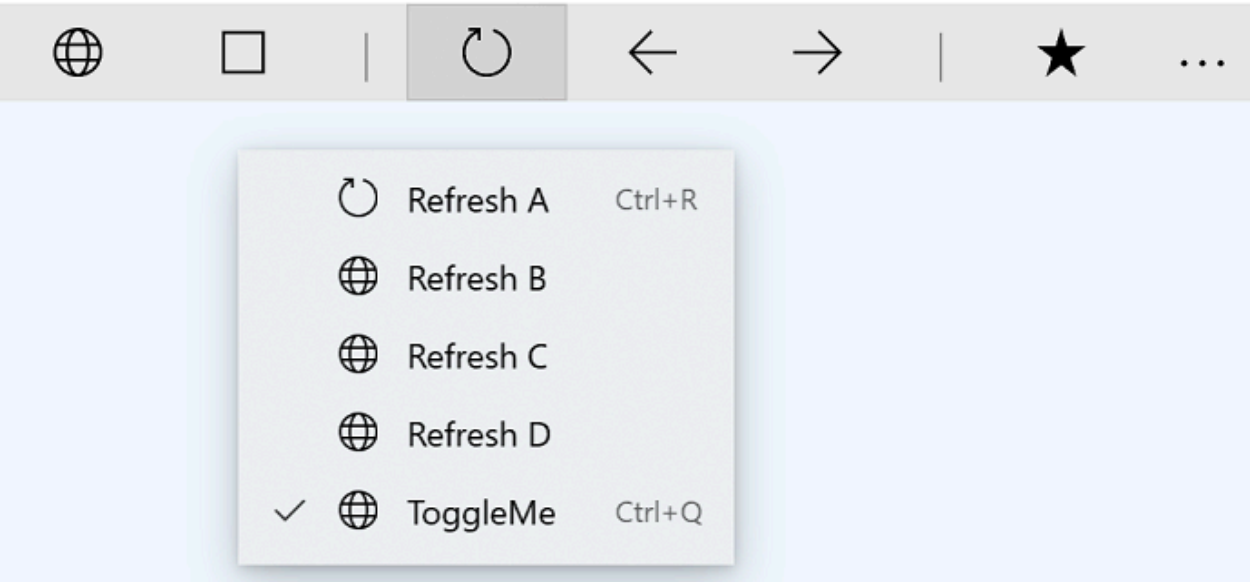
XAML

```
<AppBarButton Icon="Save" Label="Save">
    <AppBarButton.KeyboardAccelerators>
        <KeyboardAccelerator Key="S" Modifiers="Control"/>
    </AppBarButton.KeyboardAccelerators>
</AppBarButton>
```

Accelerator key combo appended to AppBarButton's default tooltip

XAML

```
<AppBarButton AccessKey="R" Icon="Refresh" Label="Refresh"
IsAccessKeyScope="True">
    <AppBarButton.Flyout>
        <MenuFlyout>
            <MenuFlyoutItem AccessKey="A" Icon="Refresh" Text="Refresh A">
                <MenuFlyoutItem.KeyboardAccelerators>
                    <KeyboardAccelerator Key="R" Modifiers="Control"/>
                </MenuFlyoutItem.KeyboardAccelerators>
            </MenuFlyoutItem>
            <MenuFlyoutItem AccessKey="B" Icon="Globe" Text="Refresh B" />
            <MenuFlyoutItem AccessKey="C" Icon="Globe" Text="Refresh C" />
            <MenuFlyoutItem AccessKey="D" Icon="Globe" Text="Refresh D" />
            <ToggleMenuFlyoutItem AccessKey="E" Icon="Globe"
Text="ToggleMe">
                <MenuFlyoutItem.KeyboardAccelerators>
                    <KeyboardAccelerator Key="Q" Modifiers="Control"/>
                </MenuFlyoutItem.KeyboardAccelerators>
            </ToggleMenuFlyoutItem>
        </MenuFlyout>
    </AppBarButton.Flyout>
</AppBarButton>
```



Accelerator key combo appended to MenuFlyoutItem's text

Control the presentation behavior by using the KeyboardAcceleratorPlacementMode property, which accepts two values: Auto or Hidden.

XAML

```
<Button Content="Save" Click="OnSave"
KeyboardAcceleratorPlacementMode="Auto">
    <Button.KeyboardAccelerators>
        <KeyboardAccelerator Key="S" Modifiers="Control" />
```

```
    </Button.KeyboardAccelerators>
</Button>
```

In some cases, you might need to present a tooltip relative to another element (typically a container object).

Here, we show how to use the KeyboardAcceleratorPlacementTarget property to display the keyboard accelerator key combination for a Save button with the Grid container instead of the button.

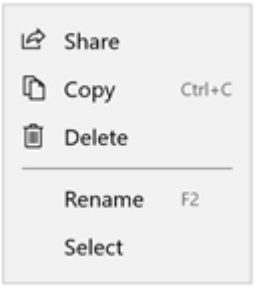
XAML

```
<Grid x:Name="Container" Padding="30">
  <Button Content="Save"
    Click="OnSave"
    KeyboardAcceleratorPlacementMode="Auto"
    KeyboardAcceleratorPlacementTarget="{x:Bind Container}">
    <Button.KeyboardAccelerators>
      <KeyboardAccelerator  Key="S" Modifiers="Control" />
    </Button.KeyboardAccelerators>
  </Button>
</Grid>
```

Labels

In some cases, we recommend using a control's label to identify whether the control has an associated keyboard accelerator and, if so, what the accelerator key combination is.

Some platform controls do this by default, specifically the MenuFlyoutItem and ToggleMenuFlyoutItem objects, while the AppBarButton and the AppBarToggleButton do it when they appear in the overflow menu of the CommandBar.



Keyboard accelerators described in a menu item label

You can override the default accelerator text for the label through the KeyboardAcceleratorTextOverride property of the MenuFlyoutItem, ToggleMenuFlyoutItem, AppBarButton, and AppBarToggleButton controls (use a single space for no text).

> **Note**
>
> The override text is not be presented if the system cannot detect an attached keyboard (you can check this yourself through the KeyboardPresent property).

Advanced Concepts

Here, we review some low-level aspects of keyboard accelerators.

Input event priority

Input events occur in a specific sequence that you can intercept and handle based on the requirements of your app.

The KeyDown/KeyUp bubbling event

In XAML, a keystroke is processed as if there is just one input bubbling pipeline. This input pipeline is used by the KeyDown/KeyUp events and character input. For example, if an element has focus and the user presses a key down, a KeyDown event is raised on the element, followed by the parent of the element, and so on up the tree, until the args.Handled property is true.

The KeyDown event is also used by some controls to implement the built-in control accelerators. When a control has a keyboard accelerator, it handles the KeyDown event, which means that there won't be KeyDown event bubbling. For example, the RichEditBox supports copy with Ctrl+C. When Ctrl is pressed, the KeyDown event is fired and bubbles, but when the user presses C at the same time, the KeyDown event is marked Handled and is not raised (unless the handledEventsToo parameter of UIElement.AddHandler is set to true).

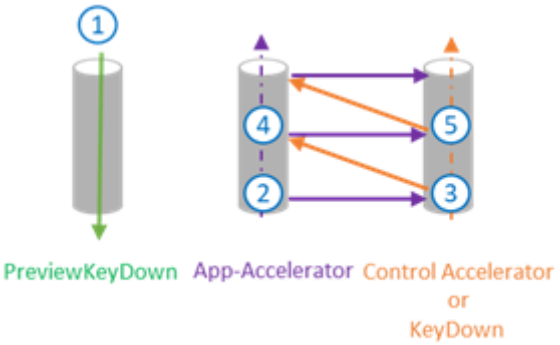
The CharacterReceived event

As the CharacterReceived event is fired after the KeyDown event for text controls such as TextBox, you can cancel character input in the KeyDown event handler.

The PreviewKeyDown and PreviewKeyUp events

The preview input events are fired before any other events. If you don't handle these events, the accelerator for the element that has the focus is fired, followed by the

KeyDown event. Both events bubble until handled.



Key event sequence

Order of events:

Preview KeyDown events
...
App accelerator
OnKeyDown method
KeyDown event
App accelerators on the parent
OnKeyDown method on the parent
KeyDown event on the parent
(Bubbles to the root)
...
CharacterReceived event
PreviewKeyUp events
KeyUpEvents

When the accelerator event is handled, the KeyDown event is also marked as handled. The KeyUp event remains unhandled.

Resolving accelerators

A keyboard accelerator event bubbles from the element that has focus up to the root. If the event isn't handled, the XAML framework looks for other unscoped app accelerators outside of the bubbling path.

When two keyboard accelerators are defined with the same key combination, the first keyboard accelerator found on the visual tree is invoked.

Scoped keyboard accelerators are invoked only when focus is inside a specific scope. For example, in a Grid that contains dozens of controls, a keyboard accelerator can be invoked for a control only when focus is within the Grid (the scope owner).

Scoping accelerators programmatically

The UIElement.TryInvokeKeyboardAccelerator method invokes any matching accelerators in the subtree of the element.

The UIElement.OnProcessKeyboardAccelerators method is executed before the keyboard accelerator. This method passes a ProcessKeyboardAcceleratorArgs object that contains the key, the modifier, and a Boolean indicating whether the keyboard accelerator is handled. If marked as handled, the keyboard accelerator bubbles (so the outside keyboard accelerator is never invoked).

> **Note**
>
> OnProcessKeyboardAccelerators always fires, whether handled or not (similar to the OnKeyDown event). You must check whether the event was marked as handled.

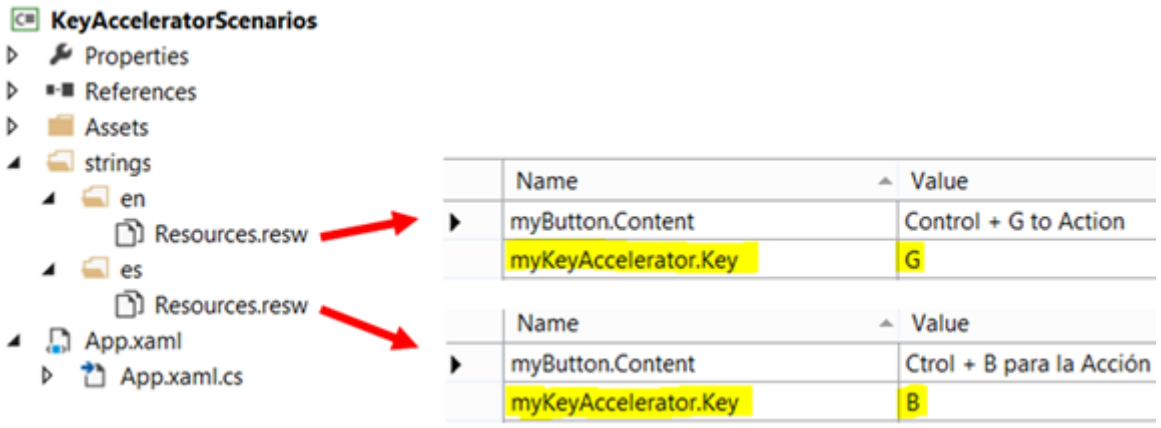
In this example, we use OnProcessKeyboardAccelerators and TryInvokeKeyboardAccelerator to scope keyboard accelerators to the Page object:

C#

```
protected override void OnProcessKeyboardAccelerators(
  ProcessKeyboardAcceleratorArgs args)
{
  if(args.Handled != true)
  {
    this.TryInvokeKeyboardAccelerator(args);
    args.Handled = true;
  }
}
```

Localize the accelerators

We recommend localizing all keyboard accelerators. You can do this with the standard resources (.resw) file and the x:Uid attribute in your XAML declarations. In this example, the Windows Runtime automatically loads the resources.



Keyboard accelerator localization with the resources file

XAML

```
<Button x:Uid="myButton" Click="OnSave">
  <Button.KeyboardAccelerators>
    <KeyboardAccelerator x:Uid="myKeyAccelerator" Modifiers="Control"/>
  </Button.KeyboardAccelerators>
</Button>
```

> **Note**
>
> Keyboard accelerators are implemented as virtual-keys. Localized accelerators must be chosen from the predefined collection of Virtual-Key codes (otherwise, a XAML parser error will occur).

Setup an accelerator programmatically

Here is an example of programmatically defining an accelerator:

C#

```
void AddAccelerator(
  VirtualKeyModifiers keyModifiers,
  VirtualKey key,
  TypedEventHandler<KeyboardAccelerator,
KeyboardAcceleratorInvokedEventArgs> handler )
  {
    var accelerator =
      new KeyboardAccelerator()
      {
        Modifiers = keyModifiers, Key = key
      };
    accelerator.Invoked += handler;
```

```
    this.KeyboardAccelerators.Add(accelerator);
  }
```

> **Note**
>
> KeyboardAccelerator is not shareable, the same KeyboardAccelerator can't be added to multiple elements.

Override keyboard accelerator behavior

You can handle the KeyboardAccelerator.Invoked event to override the default KeyboardAccelerator behavior.

This example shows how to override the "Select all" command (Ctrl+A keyboard accelerator) in a custom ListView control. We also set the Handled property to true to stop the event bubbling further.

C#

```
public class MyListView : ListView
{
  …
  protected override void
OnKeyboardAcceleratorInvoked(KeyboardAcceleratorInvokedEventArgs args)
  {
    if(args.Accelerator.Key == VirtualKey.A
      && args.Accelerator.Modifiers == KeyboardModifiers.Control)
    {
      CustomSelectAll(TypeOfSelection.OnlyNumbers);
      args.Handled = true;
    }
  }
  …
}
```

Related articles

- Keyboard interactions
- Access keys
- VirtualKey Enum

Samples

- WinUI Gallery

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Keyboard events

Article • 10/04/2021

Keyboard events and focus

The following keyboard events can occur for both hardware and touch keyboards.

Expand table

Event	Description
KeyDown	Occurs when a key is pressed.
KeyUp	Occurs when a key is released.

> **Important**
>
> Some XAML controls handle input events internally. In these cases, it might appear that an input event doesn't occur because your event listener doesn't invoke the associated handler. Typically, this subset of keys is processed by the class handler to provide built in support of basic keyboard accessibility. For example, the Button class overrides the OnKeyDown events for both the Space key and the Enter key (as well as OnPointerPressed) and routes them to the Click event of the control. When a key press is handled by the control class, the KeyDown and KeyUp events are not raised.
> This provides a built-in keyboard equivalent for invoking the button, similar to tapping it with a finger or clicking it with a mouse. Keys other than Space or Enter still fire KeyDown and KeyUp events. For more info about how class-based handling of events works (specifically, the "Input event handlers in controls" section), see Events and routed events overview.

Controls in your UI generate keyboard events only when they have input focus. An individual control gains focus when the user clicks or taps directly on that control in the layout, or uses the Tab key to step into a tab sequence within the content area.

You can also call a control's Focus method to force focus. This is necessary when you implement shortcut keys, because keyboard focus is not set by default when your UI loads. For more info, see the **Shortcut keys example** later in this topic.

For a control to receive input focus, it must be enabled, visible, and have **IsTabStop** and **HitTestVisible** property values of **true**. This is the default state for most controls. When a control has input focus, it can raise and respond to keyboard input events as described later in this topic. You can also respond to a control that is receiving or losing focus by handling the **GotFocus** and **LostFocus** events.

By default, the tab sequence of controls is the order in which they appear in the Extensible Application Markup Language (XAML). However, you can modify this order by using the **TabIndex** property. For more info, see Implementing keyboard accessibility.

Keyboard event handlers

An input event handler implements a delegate that provides the following information:

- The sender of the event. The sender reports the object where the event handler is attached.
- Event data. For keyboard events, that data will be an instance of **KeyRoutedEventArgs**. The delegate for handlers is **KeyEventHandler**. The most relevant properties of **KeyRoutedEventArgs** for most handler scenarios are **Key** and possibly **KeyStatus**.
- **OriginalSource**. Because the keyboard events are routed events, the event data provides **OriginalSource**. If you deliberately allow events to bubble up through an object tree, **OriginalSource** is sometimes the object of concern rather than sender. However, that depends on your design. For more information about how you might use **OriginalSource** rather than sender, see the "Keyboard Routed Events" section of this topic, or Events and routed events overview.

Attaching a keyboard event handler

You can attach keyboard event-handler functions for any object that includes the event as a member. This includes any **UIElement** derived class. The following XAML example shows how to attach handlers for the **KeyUp** event for a **Grid**.

XAML

```
<Grid KeyUp="Grid_KeyUp">
  ...
</Grid>
```

You can also attach an event handler in code. For more info, see Events and routed events overview.

Defining a keyboard event handler

The following example shows the incomplete event handler definition for the KeyUp event handler that was attached in the preceding example.

C#

```
void Grid_KeyUp(object sender, KeyRoutedEventArgs e)
{
    //handling code here
}
```

VB

```
Private Sub Grid_KeyUp(ByVal sender As Object, ByVal e As
KeyRoutedEventArgs)
    ' handling code here
End Sub
```

c++

```
void MyProject::MainPage::Grid_KeyUp(
  Platform::Object^ sender,
  Windows::UI::Xaml::Input::KeyRoutedEventArgs^ e)
  {
      //handling code here
  }
```

Using KeyRoutedEventArgs

All keyboard events use KeyRoutedEventArgs for event data, and **KeyRoutedEventArgs** contains the following properties:

- Key
- KeyStatus
- Handled
- OriginalSource (inherited from RoutedEventArgs)

Virtual keys

The KeyDown event is raised if a key is pressed. Likewise, KeyUp is raised if a key is released. Usually, you listen to the events to process a specific key value. To determine which key is pressed or released, check the Key value in the event data. **Key** returns a VirtualKey value. The **VirtualKey** enumeration includes all the supported keys.

Modifier keys

Modifier keys are keys such as Ctrl or Shift that users typically press in combination with other keys. Your app can use these combinations as custom keyboard shortcuts to invoke app commands.

> **Note**
>
> For built-in keyboard shortcuts, see Access keys and Keyboard accelerators.

You can detect shortcut key combinations in the **KeyDown** and **KeyUp** event handlers. When a keyboard event occurs for a non-modifier key, you can then check whether a modifier key is in the pressed state.

Alternatively, the **GetKeyState()** function of the **CoreWindow** (obtained through **CoreWindow.GetForCurrentThread()**) can also be used to check modifier state when a non-modifier key is pressed.

The following examples implement this second method while also including stub code for the first implementation.

> **Note**
>
> The Alt key is represented by the **VirtualKey.Menu** value.

Shortcut keys example

The following example demonstrates how to implement a set of custom shortcut keys. In this example, users can control media playback using Play, Pause, and Stop buttons or Ctrl+P, Ctrl+A, and Ctrl+S keyboard shortcuts. The button XAML shows the shortcuts by using tooltips and **AutomationProperties** properties in the button labels. This self-documentation is important to increase the usability and accessibility of your app. For more info, see Keyboard accessibility.

Note also that the page sets input focus to itself when it is loaded. Without this step, no control has initial input focus, and the app does not raise input events until the user sets the input focus manually (for example, by tabbing to or clicking a control).

XAML

```
<Grid KeyDown="Grid_KeyDown">
```

```
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
  </Grid.RowDefinitions>

  <MediaElement x:Name="DemoMovie" Source="xbox.wmv"
    Width="500" Height="500" Margin="20" HorizontalAlignment="Center" />

  <StackPanel Grid.Row="1" Margin="10"
    Orientation="Horizontal" HorizontalAlignment="Center">

    <Button x:Name="PlayButton" Click="MediaButton_Click"
      ToolTipService.ToolTip="Shortcut key: Ctrl+P"
      AutomationProperties.AcceleratorKey="Control P">
      <TextBlock>Play</TextBlock>
    </Button>

    <Button x:Name="PauseButton" Click="MediaButton_Click"
      ToolTipService.ToolTip="Shortcut key: Ctrl+A"
      AutomationProperties.AcceleratorKey="Control A">
      <TextBlock>Pause</TextBlock>
    </Button>

    <Button x:Name="StopButton" Click="MediaButton_Click"
      ToolTipService.ToolTip="Shortcut key: Ctrl+S"
      AutomationProperties.AcceleratorKey="Control S">
      <TextBlock>Stop</TextBlock>
    </Button>

  </StackPanel>

</Grid>
```

c++

```
//showing implementations but not header definitions
void MainPage::OnNavigatedTo(NavigationEventArgs^ e)
{
    (void) e;    // Unused parameter
    this->Loaded+=ref new
RoutedEventHandler(this,&amp;MainPage::ProgrammaticFocus);
}
void MainPage::ProgrammaticFocus(Object^ sender, RoutedEventArgs^ e)
{
    this->Focus(Windows::UI::Xaml::FocusState::Programmatic);
}

void KeyboardSupport::MainPage::MediaButton_Click(Platform::Object^ sender,
Windows::UI::Xaml::RoutedEventArgs^ e)
{
    FrameworkElement^ fe = safe_cast<FrameworkElement^>(sender);
    if (fe->Name == "PlayButton") {DemoMovie->Play();}
    if (fe->Name == "PauseButton") {DemoMovie->Pause();}
```

```
    if (fe->Name == "StopButton") {DemoMovie->Stop();}
}


bool KeyboardSupport::MainPage::IsCtrlKeyPressed()
{
    auto ctrlState = CoreWindow::GetForCurrentThread()-
>GetKeyState(VirtualKey::Control);
    return (ctrlState & CoreVirtualKeyStates::Down) ==
CoreVirtualKeyStates::Down;
}

void KeyboardSupport::MainPage::Grid_KeyDown(Platform::Object^ sender,
Windows::UI::Xaml::Input::KeyRoutedEventArgs^ e)
{
    if (e->Key == VirtualKey::Control) isCtrlKeyPressed = true;
}


void KeyboardSupport::MainPage::Grid_KeyUp(Platform::Object^ sender,
Windows::UI::Xaml::Input::KeyRoutedEventArgs^ e)
{
    if (IsCtrlKeyPressed())
    {
        if (e->Key==VirtualKey::P) { DemoMovie->Play(); }
        if (e->Key==VirtualKey::A) { DemoMovie->Pause(); }
        if (e->Key==VirtualKey::S) { DemoMovie->Stop(); }
    }
}
```

C#

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    // Set the input focus to ensure that keyboard events are raised.
    this.Loaded += delegate { this.Focus(FocusState.Programmatic); };
}

private void MediaButton_Click(object sender, RoutedEventArgs e)
{
    switch ((sender as Button).Name)
    {
        case "PlayButton": DemoMovie.Play(); break;
        case "PauseButton": DemoMovie.Pause(); break;
        case "StopButton": DemoMovie.Stop(); break;
    }
}

private static bool IsCtrlKeyPressed()
{
    var ctrlState =
CoreWindow.GetForCurrentThread().GetKeyState(VirtualKey.Control);
    return (ctrlState & CoreVirtualKeyStates.Down) ==
```

```
CoreVirtualKeyStates.Down;
}

private void Grid_KeyDown(object sender, KeyRoutedEventArgs e)
{
    if (IsCtrlKeyPressed())
    {
        switch (e.Key)
        {
            case VirtualKey.P: DemoMovie.Play(); break;
            case VirtualKey.A: DemoMovie.Pause(); break;
            case VirtualKey.S: DemoMovie.Stop(); break;
        }
    }
}
```

VisualBasic

```
Private isCtrlKeyPressed As Boolean
Protected Overrides Sub OnNavigatedTo(e As Navigation.NavigationEventArgs)

End Sub

Private Function IsCtrlKeyPressed As Boolean
    Dim ctrlState As CoreVirtualKeyStates =
CoreWindow.GetForCurrentThread().GetKeyState(VirtualKey.Control);
    Return (ctrlState & CoreVirtualKeyStates.Down) ==
CoreVirtualKeyStates.Down;
End Function

Private Sub Grid_KeyDown(sender As Object, e As KeyRoutedEventArgs)
    If IsCtrlKeyPressed() Then
        Select Case e.Key
            Case Windows.System.VirtualKey.P
                DemoMovie.Play()
            Case Windows.System.VirtualKey.A
                DemoMovie.Pause()
            Case Windows.System.VirtualKey.S
                DemoMovie.Stop()
        End Select
    End If
End Sub

Private Sub MediaButton_Click(sender As Object, e As RoutedEventArgs)
    Dim fe As FrameworkElement = CType(sender, FrameworkElement)
    Select Case fe.Name
        Case "PlayButton"
            DemoMovie.Play()
        Case "PauseButton"
            DemoMovie.Pause()
        Case "StopButton"
            DemoMovie.Stop()
```

```
    End Select
End Sub
```

> **Note**
>
> Setting AutomationProperties.AcceleratorKey or AutomationProperties.AccessKey in XAML provides string information, which documents the shortcut key for invoking that particular action. The information is captured by Microsoft UI Automation clients such as Narrator, and is typically provided directly to the user.
>
> Setting **AutomationProperties.AcceleratorKey** or **AutomationProperties.AccessKey** does not have any action on its own. You will still need to attach handlers for KeyDown or KeyUp events in order to actually implement the keyboard shortcut behavior in your app. Also, the underline text decoration for an access key is not provided automatically. You must explicitly underline the text for the specific key in your mnemonic as inline Underline formatting if you wish to show underlined text in the UI.

Keyboard routed events

Certain events are routed events, including KeyDown and KeyUp. Routed events use the bubbling routing strategy. The bubbling routing strategy means that an event originates from a child object and is then routed up to successive parent objects in the object tree. This presents another opportunity to handle the same event and interact with the same event data.

Consider the following XAML example, which handles KeyUp events for a Canvas and two Button objects. In this case, if you release a key while focus is held by either **Button** object, it raises the **KeyUp** event. The event is then bubbled up to the parent **Canvas**.

XAML

```
<StackPanel KeyUp="StackPanel_KeyUp">
  <Button Name="ButtonA" Content="Button A"/>
  <Button Name="ButtonB" Content="Button B"/>
  <TextBlock Name="statusTextBlock"/>
</StackPanel>
```

The following example shows how to implement the KeyUp event handler for the corresponding XAML content in the preceding example.

C#

```
void StackPanel_KeyUp(object sender, KeyRoutedEventArgs e)
{
    statusTextBlock.Text = String.Format(
        "The key {0} was pressed while focus was on {1}",
        e.Key.ToString(), (e.OriginalSource as FrameworkElement).Name);
}
```

Notice the use of the OriginalSource property in the preceding handler. Here, **OriginalSource** reports the object that raised the event. The object could not be the StackPanel because the **StackPanel** is not a control and cannot have focus. Only one of the two buttons within the **StackPanel** could possibly have raised the event, but which one? You use **OriginalSource** to distinguish the actual event source object, if you are handling the event on a parent object.

The Handled property in event data

Depending on your event handling strategy, you might want only one event handler to react to a bubbling event. For instance, if you have a specific KeyUp handler attached to one of the Button controls, it would have the first opportunity to handle that event. In this case, you might not want the parent panel to also handle the event. For this scenario, you can use the Handled property in the event data.

The purpose of the Handled property in a routed event data class is to report that another handler you registered earlier on the event route has already acted. This influences the behavior of the routed event system. When you set **Handled** to **true** in an event handler, that event stops routing and is not sent to successive parent elements.

AddHandler and already-handled keyboard events

You can use a special technique for attaching handlers that can act on events that you already marked as handled. This technique uses the AddHandler method to register a handler, rather than using XAML attributes or language-specific syntax for adding handlers, such as += in C#.

A general limitation of this technique is that the **AddHandler** API takes a parameter of type RoutedEvent identifying the routed event in question. Not all routed events provide a **RoutedEvent** identifier, and this consideration thus affects which routed events can still be handled in the Handled case. The KeyDown and KeyUp events have

routed event identifiers (KeyDownEvent and KeyUpEvent) on UIElement. However, other events such as TextBox.TextChanged do not have routed event identifiers and thus cannot be used with the **AddHandler** technique.

Overriding keyboard events and behavior

You can override key events for specific controls (such as GridView) to provide consistent focus navigation for various input devices, including keyboard and gamepad.

In the following example, we subclass the control and override the KeyDown behavior to move focus to the GridView content when any arrow key is pressed.

C#

```
public class CustomGridView : GridView
{
  protected override void OnKeyDown(KeyRoutedEventArgs e)
  {
    // Override arrow key behaviors.
    if (e.Key != Windows.System.VirtualKey.Left && e.Key !=
      Windows.System.VirtualKey.Right && e.Key !=
        Windows.System.VirtualKey.Down && e.Key !=
          Windows.System.VirtualKey.Up)
            base.OnKeyDown(e);
    else
      FocusManager.TryMoveFocus(FocusNavigationDirection.Down);
  }
}
```

> ⓘ Note
>
> If using a GridView for layout only, consider using other controls such as ItemsControl with ItemsWrapGrid.

Commanding

A small number of UI elements provide built-in support for commanding. Commanding uses input-related routed events in its underlying implementation. It enables processing of related UI input, such as a certain pointer action or a specific accelerator key, by invoking a single command handler.

If commanding is available for a UI element, consider using its commanding APIs instead of any discrete input events. For more info, see ButtonBase.Command.

You can also implement **ICommand** to encapsulate command functionality that you invoke from ordinary event handlers. This enables you to use commanding even when there is no **Command** property available.

Text input and controls

Certain controls react to keyboard events with their own handling. For instance, **TextBox** is a control that is designed to capture and then visually represent text that was entered by using the keyboard. It uses **KeyUp** and **KeyDown** in its own logic to capture keystrokes, then also raises its own **TextChanged** event if the text actually changed.

You can still generally add handlers for **KeyUp** and **KeyDown** to a **TextBox**, or any related control that is intended to process text input. However, as part of its intended design, a control might not respond to all key values that are directed to it through key events. Behavior is specific to each control.

As an example, **ButtonBase** (the base class for **Button**) processes **KeyUp** so that it can check for the Spacebar or Enter key. **ButtonBase** considers **KeyUp** equivalent to a mouse left button down for purposes of raising a **Click** event. This processing of the event is accomplished when **ButtonBase** overrides the virtual method **OnKeyUp**. In its implementation, it sets **Handled** to **true**. The result is that any parent of a button that is listening for a key event, in the case of a Spacebar, would not receive the already-handled event for its own handlers.

Another example is **TextBox**. Some keys, such as the arrow keys, are not considered text by **TextBox** and are instead considered specific to the control UI behavior. The **TextBox** marks these event cases as handled.

Custom controls can implement their own similar override behavior for key events by overriding **OnKeyDown** / **OnKeyUp**. If your custom control processes specific accelerator keys, or has control or focus behavior that is similar to the scenario described for **TextBox**, you should place this logic in your own **OnKeyDown** / **OnKeyUp** overrides.

The touch keyboard

Text input controls provide automatic support for the touch keyboard. When the user sets the input focus to a text control by using touch input, the touch keyboard appears automatically. When the input focus is not on a text control, the touch keyboard is hidden.

When the touch keyboard appears, it automatically repositions your UI to ensure that the focused element remains visible. This can cause other important areas of your UI to move off screen. However, you can disable the default behavior and make your own UI adjustments when the touch keyboard appears. For more info, see the Touch keyboard sample.

If you create a custom control that requires text input, but does not derive from a standard text input control, you can add touch keyboard support by implementing the correct UI Automation control patterns. For more info, see the Touch keyboard sample.

Key presses on the touch keyboard raise **KeyDown** and **KeyUp** events just like key presses on hardware keyboards. However, the touch keyboard will not raise input events for Ctrl+A, Ctrl+Z, Ctrl+X, Ctrl+C, and Ctrl+V, which are reserved for text manipulation in the input control.

You can make it much faster and easier for users to enter data in your app by setting the input scope of the text control to match the kind of data you expect the user to enter. The input scope provides a hint at the type of text input expected by the control so the system can provide a specialized touch keyboard layout for the input type. For example, if a text box is used only to enter a 4-digit PIN, set the **InputScope** property to **Number**. This tells the system to show the numeric keypad layout, which makes it easier for the user to enter the PIN. For more detail, see Use input scope to change the touch keyboard.

Related articles

Developers

- Keyboard interactions
- Identify input devices
- Respond to the presence of the touch keyboard

Designers

- Keyboard design guidelines

Samples

- Touch keyboard sample
- Basic input sample

- Low latency input sample
- Focus visuals sample

Archive Samples

- Input sample
- Input: Device capabilities sample
- Input: Touch keyboard sample
- Responding to the appearance of the on-screen keyboard sample
- XAML text editing sample

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Focus navigation for keyboard, gamepad, remote control, and accessibility tools

Article • 06/24/2021

Use focus navigation to provide comprehensive and consistent interaction experiences in your Windows apps and custom controls for keyboard power users, those with disabilities and other accessibility requirements, as well as the 10-foot experience of television screens and the Xbox One.

Overview

Focus navigation refers to the underlying mechanism that enables users to navigate and interact with the UI of a Windows application using a keyboard, gamepad, or remote control.

> **Note**
>
> Input devices are typically classified as pointing devices, such as touch, touchpad, pen, and mouse, and non-pointing devices, such as keyboard, gamepad, and remote control.

This topic describes how to optimize a Windows application and build custom interaction experiences for users that rely on non-pointing input types.

Even though we focus on keyboard input for custom controls in Windows apps on PCs, a well-designed keyboard experience is also important for software keyboards such as the touch keyboard and the On-Screen Keyboard (OSK), supporting accessibility tools such as Windows Narrator, and supporting the 10-foot experience.

See Handle pointer input for guidance on building custom experiences in Windows applications for pointing devices.

For more general information on building apps and experiences for keyboard, see Keyboard Interaction.

General guidance

Only those UI elements that require user interaction should support focus navigation, elements that don't require an action, such as static images, do not need keyboard focus. Screen readers and similar accessibility tools still announce these static elements, even when they are not included in focus navigation.

It is important to remember that unlike navigating with a pointer device such as a mouse or touch, focus navigation is linear. When implementing focus navigation, consider how a user will interact with your application and what the logical navigation should be. In most cases, we recommend custom focus navigation behavior follows the preferred reading pattern of the user's culture.

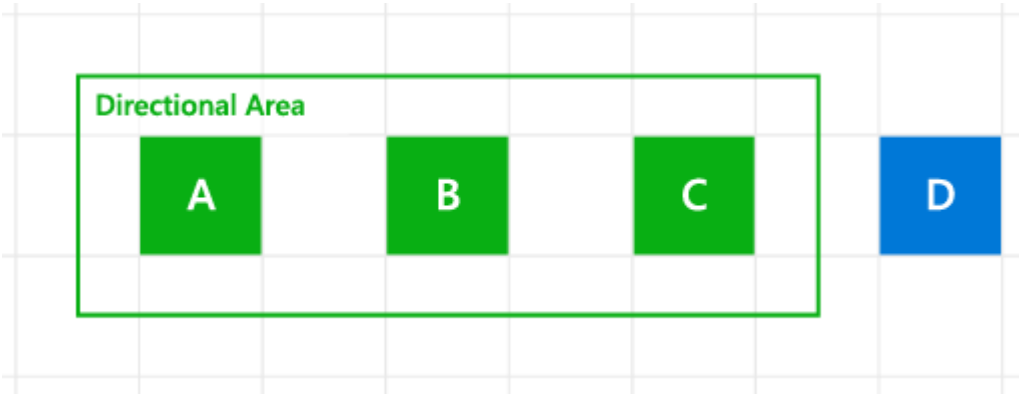
Some other focus navigation considerations include:

- Are controls grouped logically?
- Are there groups of controls with greater importance?
  - If yes, do those groups contain sub-groups?
- Does the layout require custom directional navigation (arrow keys) and tab order?

The Engineering Software for Accessibility eBook has an excellent chapter on *Designing the Logical Hierarchy*.

2D directional navigation for keyboard

The 2D inner navigation region of a control, or control group, is referred to as its "directional area". When focus shifts to this object, the keyboard arrow keys (left, right, up, and down) can be used to navigate between child elements within the directional area.



2D Inner navigation region, or directional area, of a control group

You can use the XYFocusKeyboardNavigation property (which has possible values of Auto, Enabled, or Disabled) to manage 2D inner navigation with the keyboard arrow keys.

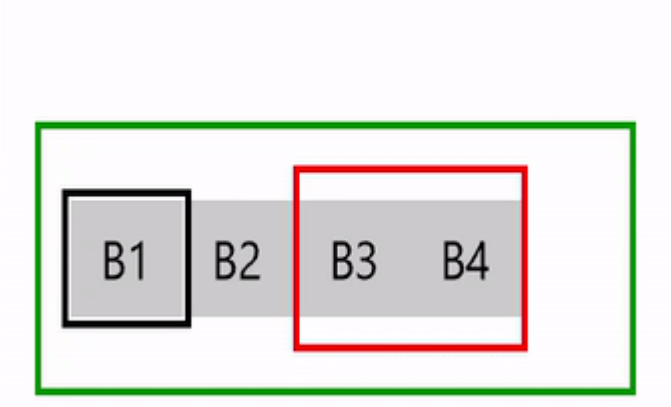
> ⓘ **Note**
>
> Tab order is not affected by this property. To avoid a confusing navigation experience, we recommend that child elements of a directional area *not* be explicitly specified in the tab navigation order of your application. See the **UIElement.TabFocusNavigation** and **TabIndex** properties for more detail on tabbing behavior for an element.

Auto (default behavior)

When set to Auto, directional navigation behavior is determined by the element's ancestry, or inheritance hierarchy. If all ancestors are in default mode (set to **Auto**), directional navigation with the keyboard is *not* supported.

Disabled

Set **XYFocusKeyboardNavigation** to **Disabled** to block directional navigation to the control and its child elements.



XYFocusKeyboardNavigation disabled behavior

In this example, the primary StackPanel (ContainerPrimary) has **XYFocusKeyboardNavigation** set to **Enabled**. All child elements inherit this setting, and can be navigated to with the arrow keys. However, the B3 and B4 elements are in a secondary StackPanel (ContainerSecondary) with **XYFocusKeyboardNavigation** set to **Disabled**, which overrides the primary container and disables arrow key navigation to itself and between its child elements.

XAML

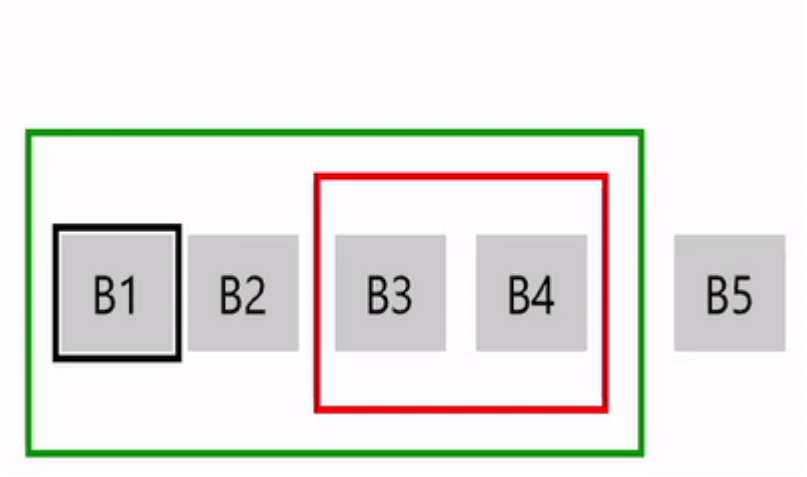
```
<Grid
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
    TabFocusNavigation="Cycle">
    <Grid.RowDefinitions>
        <RowDefinition Height="40"/>
        <RowDefinition Height="75"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <TextBlock Name="KeyPressed"
               Grid.Row="0"
               FontWeight="ExtraBold"
               HorizontalTextAlignment="Center"
               TextWrapping="Wrap"
               Padding="10" />
    <StackPanel Name="ContainerPrimary"
               XYFocusKeyboardNavigation="Enabled"
               KeyDown="ContainerPrimary_KeyDown"
               Orientation="Horizontal"
               BorderBrush="Green"
               BorderThickness="2"
               Grid.Row="1"
               Padding="10"
               MaxWidth="200">
        <Button Name="B1"
               Content="B1"
               GettingFocus="Btn_GettingFocus" />
        <Button Name="B2"
               Content="B2"
               GettingFocus="Btn_GettingFocus" />
        <StackPanel Name="ContainerSecondary"
```

```
                    XYFocusKeyboardNavigation="Disabled"
                    Orientation="Horizontal"
                    BorderBrush="Red"
                    BorderThickness="2">
            <Button Name="B3"
                    Content="B3"
                    GettingFocus="Btn_GettingFocus" />
            <Button Name="B4"
                    Content="B4"
                    GettingFocus="Btn_GettingFocus" />
        </StackPanel>
    </StackPanel>
</Grid>
```

Enabled

Set **XYFocusKeyboardNavigation** to **Enabled** to support 2D directional navigation to a control and each of its UIElement child objects.

When set, navigation with the arrow keys is restricted to elements within the directional area. Tab navigation is not affected, as all controls remain accessible through their tab order hierarchy.



XYFocusKeyboardNavigation enabled behavior

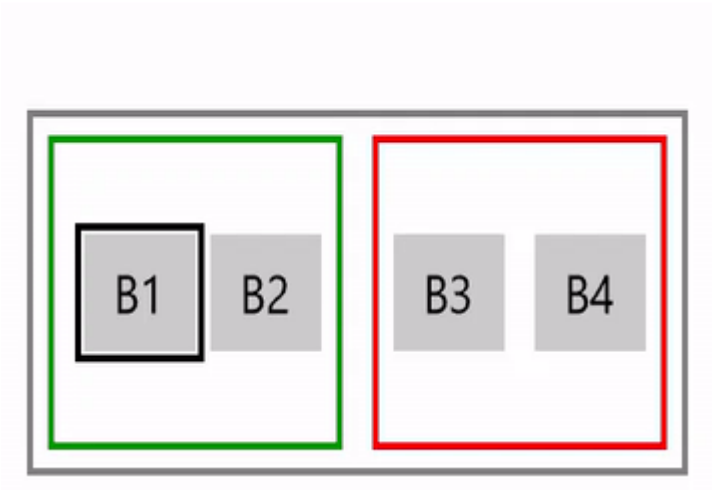
In this example, the primary StackPanel (ContainerPrimary) has **XYFocusKeyboardNavigation** set to **Enabled**. All child elements inherit this setting, and can be navigated to with the arrow keys. The B3 and B4 elements are in a secondary StackPanel (ContainerSecondary) where **XYFocusKeyboardNavigation** is not set, which then inherits the primary container setting. The B5 element is not within a declared directional area, and does not support arrow key navigation but does support standard tab navigation behavior.

XAML

```
<Grid
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
    TabFocusNavigation="Cycle">
    <Grid.RowDefinitions>
        <RowDefinition Height="40"/>
        <RowDefinition Height="100"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <TextBlock Name="KeyPressed"
               Grid.Row="0"
               FontWeight="ExtraBold"
               HorizontalTextAlignment="Center"
               TextWrapping="Wrap"
               Padding="10" />
    <StackPanel Grid.Row="1"
                Orientation="Horizontal"
                HorizontalAlignment="Center">
        <StackPanel Name="ContainerPrimary"
                    XYFocusKeyboardNavigation="Enabled"
                    KeyDown="ContainerPrimary_KeyDown"
                    Orientation="Horizontal"
                    BorderBrush="Green"
                    BorderThickness="2"
                    Padding="5" Margin="5">
            <Button Name="B1"
                    Content="B1"
                    GettingFocus="Btn_GettingFocus" Margin="5" />
            <Button Name="B2"
                    Content="B2"
                    GettingFocus="Btn_GettingFocus" />
            <StackPanel Name="ContainerSecondary"
                        Orientation="Horizontal"
                        BorderBrush="Red"
                        BorderThickness="2"
                        Margin="5">
                <Button Name="B3"
                        Content="B3"
                        GettingFocus="Btn_GettingFocus"
                        Margin="5" />
                <Button Name="B4"
                        Content="B4"
                        GettingFocus="Btn_GettingFocus"
                        Margin="5" />
            </StackPanel>
        </StackPanel>
        <Button Name="B5"
                Content="B5"
                GettingFocus="Btn_GettingFocus"
                Margin="5" />
    </StackPanel>
</Grid>
```

You can have multiple levels of nested directional areas. If all parent elements have XYFocusKeyboardNavigation set to Enabled, inner navigation region boundaries are ignored.

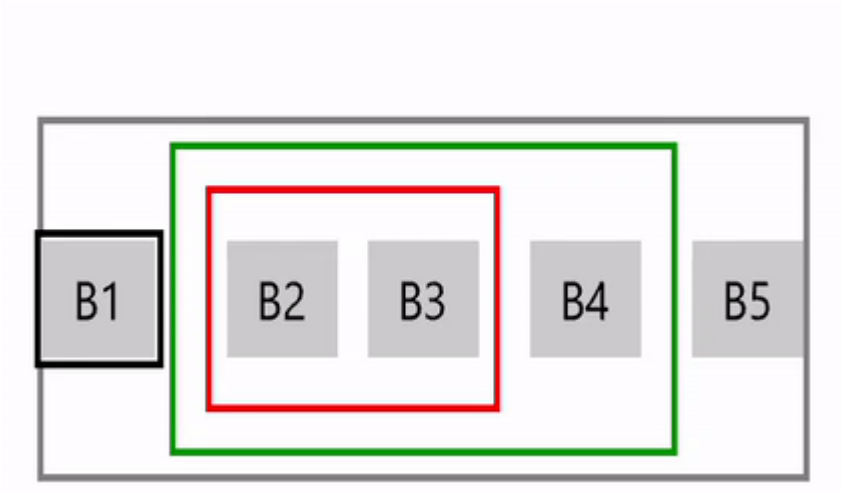
Here's an example of two nested directional areas within an element that does not explicitly support 2D directional navigation. In this case, directional navigation is not supported between the two nested areas.



XYFocusKeyboardNavigation enabled and nested behavior

Here's a more complex example of three nested directional areas where:

- When B1 has focus, only B5 can be navigated to (and vice versa) because there is a directional area boundary where XYFocusKeyboardNavigation set to Disabled, making B2, B3, and B4 unreachable with the arrow keys
- When B2 has focus, only B3 can be navigated to (and vice versa) because the directional area boundary prevents arrow key navigation to B1, B4, and B5
- When B4 has focus, the Tab key must be used to navigate between controls



XYFocusKeyboardNavigation enabled and complex nested behavior

Tab navigation

While the arrow keys can be used for 2D directional navigation witin a control, or control group, the Tab key can be used to navigate between all controls in a Windows application.

All interactive controls support Tab key navigation by default (IsEnabled and IsTabStop property are **true**), with the logical tab order derived from the control layout in your application. However, the default order does not necessarily correspond to the visual order. The actual display position might depend on the parent layout container and certain properties that you can set on the child elements to influence the layout.

Avoid a custom tab order that makes the focus jump around in your application. For example, a list of controls in a form should have a tab order that flows from top to bottom and left to right (depending on locale).

In this section we describe how this tab order can be fully customized to suit your app.

Set the tab navigation behavior

The TabFocusNavigation property of UIElement specifies the tab navigation behavior for its entire object tree (or directional area).

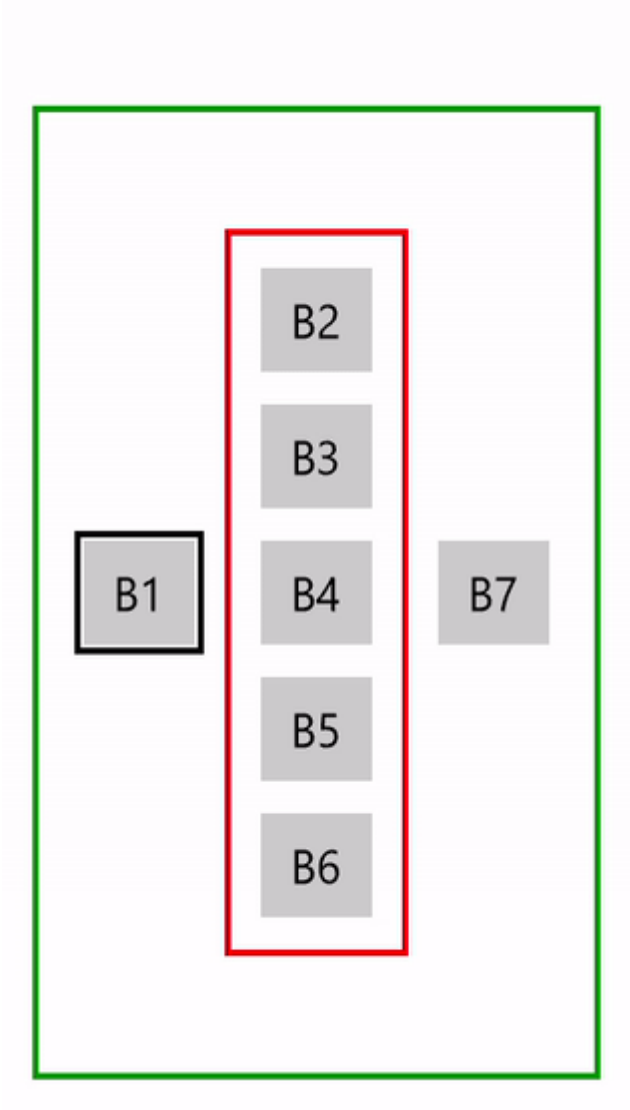
> ⓘ **Note**
>
> Use this property instead of the **Control.TabNavigation** property for objects that do not use a **ControlTemplate** to define their appearance.

As we mentioned in the previous section, to avoid a confusing navigation experience, we recommend that child elements of a directional area *not* be explicitly specified in the tab navigation order of your application. See the UIElement.TabFocusNavigation and the TabIndex properties for more detail on tabbing behavior for an element.

> For versions older than Windows 10 Creators Update (build 10.0.15063), tab settings were limited to ControlTemplate objects. For more info, see Control.TabNavigation.

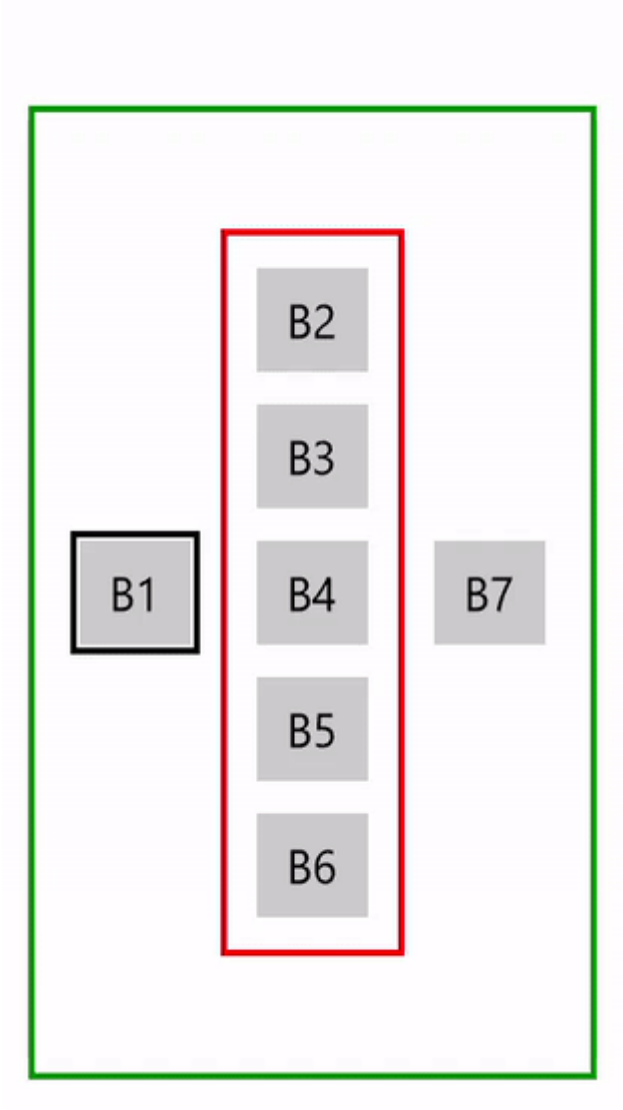
TabFocusNavigation has a value of type KeyboardNavigationMode with the following possible values (note that these examples are not custom control groups and do not require inner navigation with the arrow keys):

- **Local** (default) Tab indexes are recognized on the local subtree inside the container. For this example, the tab order is B1, B2, B3, B4, B5, B6, B7, B1.



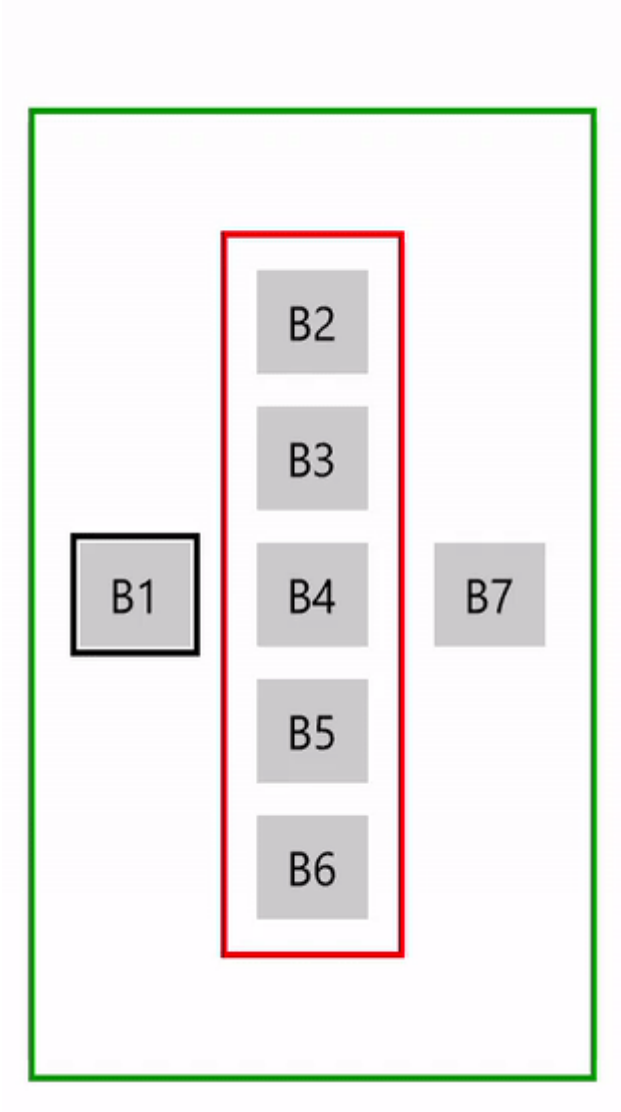
"Local" tab navigation behavior

- **Once** The container and all child elements receive focus once. For this example, the tab order is B1, B2, B7, B1 (inner navigation with arrow key is also demonstrated).



"Once" tab navigation behavior

- **Cycle**
  Focus cycles back to the initial focusable element inside a container. For this example, the tab order is B1, B2, B3, B4, B5, B6, B2...



"Cycle" tab navigation behavior

Here's the code for the preceding examples (with TabFocusNavigation ="Cycle").

XAML

```
<Grid
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
    TabFocusNavigation="Cycle">
    <Grid.RowDefinitions>
        <RowDefinition Height="40"/>
        <RowDefinition Height="300"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <TextBlock Name="KeyPressed"
               Grid.Row="0"
               FontWeight="ExtraBold"
               HorizontalTextAlignment="Center"
               TextWrapping="Wrap"
               Padding="10" />
    <StackPanel Name="ContainerPrimary"
                KeyDown="Container_KeyDown"
                Orientation="Horizontal"
```

```
                HorizontalAlignment="Center"
                BorderBrush="Green"
                BorderThickness="2"
                Grid.Row="1"
                Padding="10"
                MaxWidth="200">
        <Button Name="B1"
                Content="B1"
                GettingFocus="Btn_GettingFocus"
                Margin="5"/>
        <StackPanel Name="ContainerSecondary"
                    KeyDown="Container_KeyDown"
                    XYFocusKeyboardNavigation="Enabled"
                    TabFocusNavigation ="Cycle"
                    Orientation="Vertical"
                    VerticalAlignment="Center"
                    BorderBrush="Red"
                    BorderThickness="2"
                    Padding="5" Margin="5">
            <Button Name="B2"
                    Content="B2"
                    GettingFocus="Btn_GettingFocus"
                    Margin="5"/>
            <Button Name="B3"
                    Content="B3"
                    GettingFocus="Btn_GettingFocus"
                    Margin="5"/>
            <Button Name="B4"
                    Content="B4"
                    GettingFocus="Btn_GettingFocus"
                    Margin="5"/>
            <Button Name="B5"
                    Content="B5"
                    GettingFocus="Btn_GettingFocus"
                    Margin="5"/>
            <Button Name="B6"
                    Content="B6"
                    GettingFocus="Btn_GettingFocus"
                    Margin="5"/>
        </StackPanel>
        <Button Name="B7"
                Content="B7"
                GettingFocus="Btn_GettingFocus"
                Margin="5"/>
    </StackPanel>
</Grid>
```

TabIndex

Use TabIndex to specify the order in which elements receive focus when the user navigates through controls using the Tab key. A control with a lower tab index receives focus before a control with a higher index.

When a control has no TabIndex specified, it is assigned a higher index value than the current highest index value (and the lowest priority) of all interactive controls in the visual tree, based on scope.

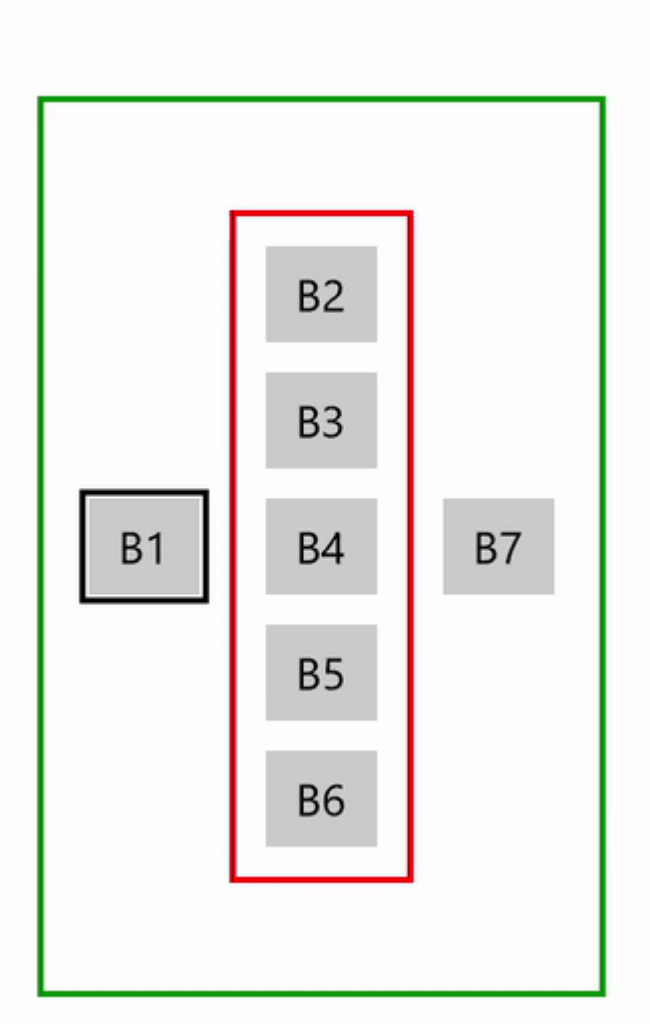
All child elements of a control are considered a scope, and if one of these elements also has child elements, they are considered another scope. Any ambiguity is resolved by choosing the first element on the visual tree of the scope.

To exclude a control from the tab order, set the IsTabStop property to **false**.

Override the default tab order by setting the TabIndex property.

> ⓘ **Note**
>
> **TabIndex** works the same way with both **UIElement.TabFocusNavigation** and **Control.TabNavigation**.

Here, we show how focus navigation can be affected by the TabIndex property on specific elements.



"Local" tab navigation with TabIndex behavior

In the preceding example, there are two scopes:

- B1, directional area (B2 - B6), and B7
- directional area (B2 - B6)

When B3 (in the directional area) gets focus, the scope changes and tab navigation transfers to the directional area where the best candidate for subsequent focus is identified. In this case, B2 followed by B4, B5, and B6. Scope then changes again, and focus moves to B1.

Here's the code for this example.

XAML

```
<Grid
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
    TabFocusNavigation="Cycle">
    <Grid.RowDefinitions>
        <RowDefinition Height="40"/>
        <RowDefinition Height="300"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <TextBlock Name="KeyPressed"
               Grid.Row="0"
               FontWeight="ExtraBold"
               HorizontalTextAlignment="Center"
               TextWrapping="Wrap"
               Padding="10" />
    <StackPanel Name="ContainerPrimary"
                KeyDown="Container_KeyDown"
                Orientation="Horizontal"
                HorizontalAlignment="Center"
                BorderBrush="Green"
                BorderThickness="2"
                Grid.Row="1"
                Padding="10"
                MaxWidth="200">
        <Button Name="B1"
                Content="B1"
                TabIndex="1"
                ToolTipService.ToolTip="TabIndex = 1"
                GettingFocus="Btn_GettingFocus"
                Margin="5"/>
        <StackPanel Name="ContainerSecondary"
                    KeyDown="Container_KeyDown"
                    TabFocusNavigation ="Local"
                    Orientation="Vertical"
                    VerticalAlignment="Center"
                    BorderBrush="Red"
                    BorderThickness="2"
                    Padding="5" Margin="5">
            <Button Name="B2"
```

```
                    Content="B2"
                    GettingFocus="Btn_GettingFocus"
                    Margin="5"/>
            <Button Name="B3"
                    Content="B3"
                    TabIndex="3"
                    ToolTipService.ToolTip="TabIndex = 3"
                    GettingFocus="Btn_GettingFocus"
                    Margin="5"/>
            <Button Name="B4"
                    Content="B4"
                    GettingFocus="Btn_GettingFocus"
                    Margin="5"/>
            <Button Name="B5"
                    Content="B5"
                    GettingFocus="Btn_GettingFocus"
                    Margin="5"/>
            <Button Name="B6"
                    Content="B6"
                    GettingFocus="Btn_GettingFocus"
                    Margin="5"/>
        </StackPanel>
        <Button Name="B7"
                Content="B7"
                TabIndex="2"
                ToolTipService.ToolTip="TabIndex = 2"
                GettingFocus="Btn_GettingFocus"
                Margin="5"/>
    </StackPanel>
</Grid>
```

2D directional navigation for keyboard, gamepad, and remote control

Non-pointer input types such as keyboard, gamepad, remote control, and accessibility tools like Windows Narrator, share a common, underlying mechanism for navigating and interacting with the UI of your Windows application.

In this section, we cover how to specify a preferred navigation strategy and fine tune focus navigation within your application through a set of navigation strategy properties that support all focus-based, non-pointer input types.

For more general information on building apps and experiences for Xbox/TV, see Keyboard Interaction, Designing for Xbox and TV, and Gamepad and remote control interactions.

Navigation Strategies

> Navigation strategies are applicable to keyboard, gamepad, remote control, and various accessibility tools.

The following navigation strategy properties let you influence which control receives focus based on the arrow key, directional pad (D-pad) button, or similar pressed.

- XYFocusUpNavigationStrategy
- XYFocusDownNavigationStrategy
- XYFocusLeftNavigationStrategy
- XYFocusRightNavigationStrategy

These properties have possible values of Auto (default), NavigationDirectionDistance, Projection, or RectilinearDistance .

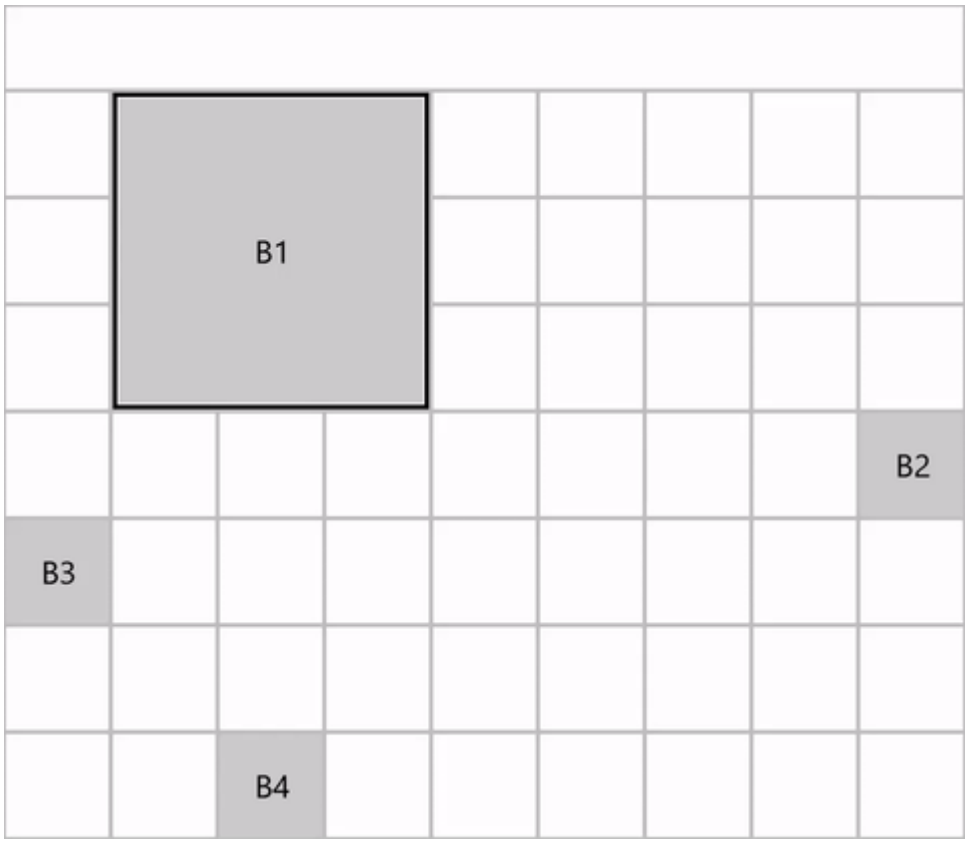
If set to **Auto**, the behavior of the element is based on the ancestors of the element. If all elements are set to **Auto**, **Projection** is used.

> ⓘ **Note**
>
> Other factors, such as the previously focused element or proximity to the axis of the navigation direction, can influence the result.

Projection

The Projection strategy moves focus to the first element encountered when the edge of the currently focused element is *projected* in the direction of navigation.

In this example, each focus navigation direction is set to Projection. Notice how focus moves down from B1 to B4, bypassing B3. This is because, B3 is not in the projection zone. Also notice how a focus candidate is not identified when moving left from B1. This is because the position of B2 relative to B1 eliminates B3 as a candidate. If B3 was in the same row as B2, it would be a viable candidate for left navigation. B2 is a viable candidate due to its unobstructed proximity to the axis of navigation direction.

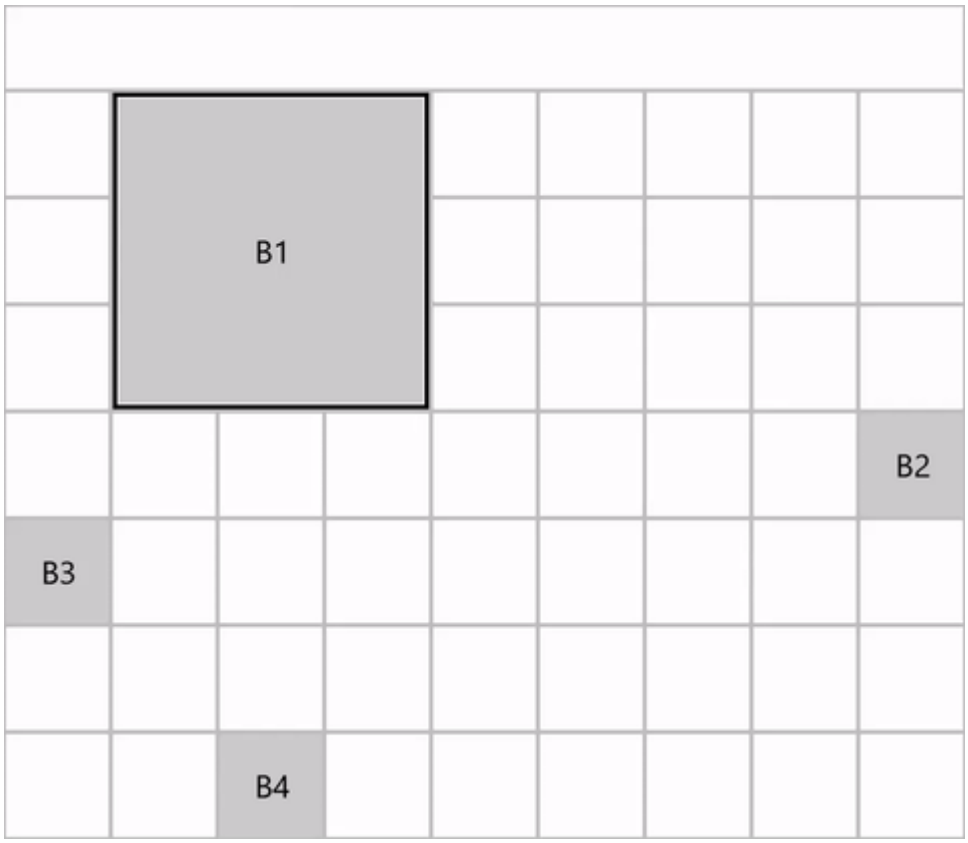


Projection navigation strategy

NavigationDirectionDistance

The NavigationDirectionDistance strategy moves focus to the element closest to the axis of the navigation direction.

The edge of the bounding rect corresponding to the navigation direction is *extended* and *projected* to identify candidate targets. The first element encountered is identified as the target. In the case of multiple candidates, the closest element is identified as the target. If there are still multiple candidates, the topmost/leftmost element is identified as the candidate.

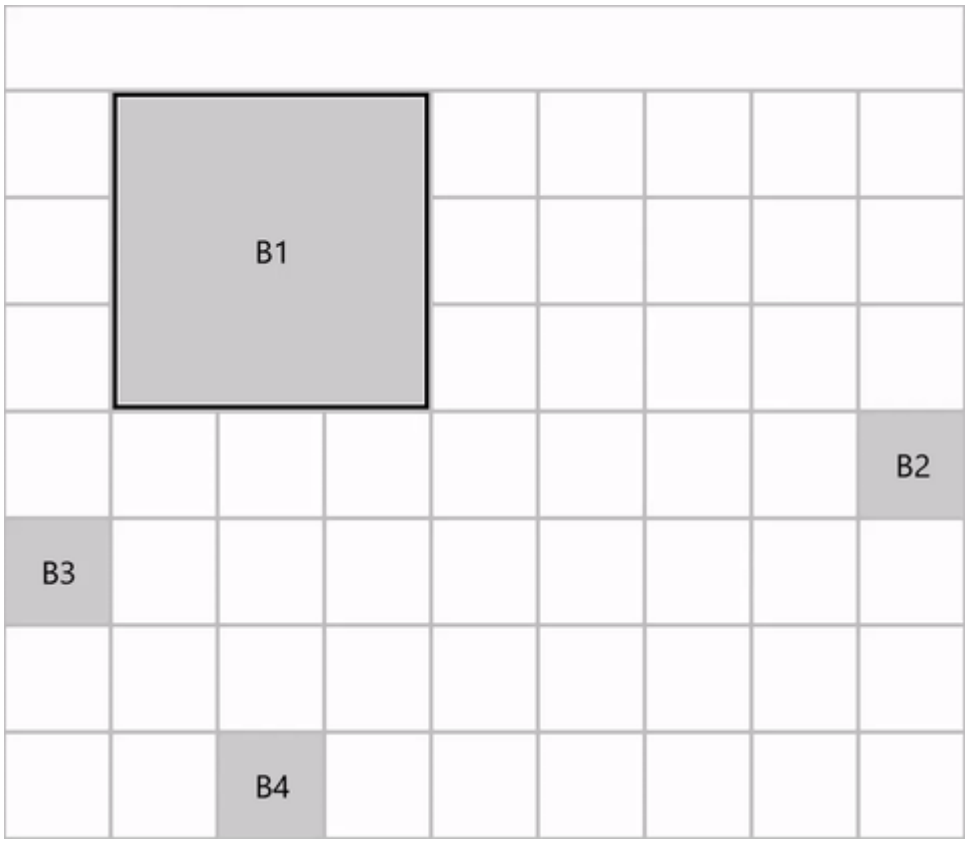


NavigationDirectionDistance navigation strategy

RectilinearDistance

The RectilinearDistance strategy moves focus to the closest element based on 2D rectilinear distance (Taxicab geometry).

The sum of the primary distance and the secondary distance to each potential candidate is used to identify the best candidtate. In a tie, the first element to the left is selected if the requested direction is up or down, and the first element to the top is selected if the requested direction is left or right.



RectilinearDistance navigation strategy

This image shows how, when B1 has focus and down is the requested direction, B3 is the RectilinearDistance focus candidate. This is based on the following calcualations for this example:

- Distance (B1, B3, Down) is 10 + 0 = 10
- Distance (B1, B2, Down) is 0 + 40 = 30
- Distance (B1, D, Down) is 30 + 0 = 30

Related articles

- Programmatic focus navigation
- Keyboard interactions
- Keyboard accessibility

Programmatic focus navigation

Article • 06/24/2021

To move focus programmatically in your Windows application, you can use either the FocusManager.TryMoveFocus method or the FindNextElement method.

TryMoveFocus attempts to change focus from the element with focus to the next focusable element in the specified direction, while FindNextElement retrieves the element (as a DependencyObject) that will receive focus based on the specified navigation direction (directional navigation only, cannot be used to emulate tab navigation).

> **Note**
>
> We recommend using the **FindNextElement** method instead of **FindNextFocusableElement** because FindNextFocusableElement retrieves a UIElement, which returns null if the next focusable element is not a UIElement (such as a Hyperlink object).

Find a focus candidate within a scope

You can customize the focus navigation behavior for both TryMoveFocus and FindNextElement, including searching for the next focus candidate within a specific UI tree or excluding specific elements from consideration.

This example uses a TicTacToe game to demonstrate the TryMoveFocus and FindNextElement methods.

XAML

```
<StackPanel Orientation="Horizontal"
                VerticalAlignment="Center"
```

```
                HorizontalAlignment="Center" >
    <Button Content="Start Game" />
    <Button Content="Undo Movement" />
    <Grid x:Name="TicTacToeGrid" KeyDown="OnKeyDown">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="50" />
            <ColumnDefinition Width="50" />
            <ColumnDefinition Width="50" />
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="50" />
            <RowDefinition Height="50" />
            <RowDefinition Height="50" />
        </Grid.RowDefinitions>
        <myControls:TicTacToeCell
            Grid.Column="0" Grid.Row="0"
            x:Name="Cell00" />
        <myControls:TicTacToeCell
            Grid.Column="1" Grid.Row="0"
            x:Name="Cell10"/>
        <myControls:TicTacToeCell
            Grid.Column="2" Grid.Row="0"
            x:Name="Cell20"/>
        <myControls:TicTacToeCell
            Grid.Column="0" Grid.Row="1"
            x:Name="Cell01"/>
        <myControls:TicTacToeCell
            Grid.Column="1" Grid.Row="1"
            x:Name="Cell11"/>
        <myControls:TicTacToeCell
            Grid.Column="2" Grid.Row="1"
            x:Name="Cell21"/>
        <myControls:TicTacToeCell
            Grid.Column="0" Grid.Row="2"
            x:Name="Cell02"/>
        <myControls:TicTacToeCell
            Grid.Column="1" Grid.Row="2"
            x:Name="Cell22"/>
        <myControls:TicTacToeCell
            Grid.Column="2" Grid.Row="2"
            x:Name="Cell32"/>
    </Grid>
</StackPanel>
```

C#

```
private void OnKeyDown(object sender, KeyRoutedEventArgs e)
{
    DependencyObject candidate = null;

    var options = new FindNextElementOptions ()
    {
        SearchRoot = TicTacToeGrid,
```

```
        XYFocusNavigationStrategyOverride =
XYFocusNavigationStrategyOverride.Projection
    };

    switch (e.Key)
    {
        case Windows.System.VirtualKey.Up:
            candidate =
                FocusManager.FindNextElement(
                    FocusNavigationDirection.Up, options);
            break;
        case Windows.System.VirtualKey.Down:
            candidate =
                FocusManager.FindNextElement(
                    FocusNavigationDirection.Down, options);
            break;
        case Windows.System.VirtualKey.Left:
            candidate = FocusManager.FindNextElement(
                FocusNavigationDirection.Left, options);
            break;
        case Windows.System.VirtualKey.Right:
            candidate =
                FocusManager.FindNextElement(
                    FocusNavigationDirection.Right, options);
            break;
    }
    // Also consider whether candidate is a Hyperlink, WebView, or
TextBlock.
    if (candidate != null && candidate is Control)
    {
        (candidate as Control).Focus(FocusState.Keyboard);
    }
}
```

Use FindNextElementOptions to further customize how focus candidates are identified. This object provides the following properties:

- SearchRoot - Scope the search for focus navigation candidates to the children of this DependencyObject. Null indicates to start the search from the root of the visual tree.

> **ⓘ Important**
>
> If one or more transforms are applied to the descendants of **SearchRoot** that place them outside of the directional area, these elements are still considered candidates.

- ExclusionRect - Focus navigation candidates are identified using a "fictitious" bounding rectangle where all overlapping objects are excluded from navigation

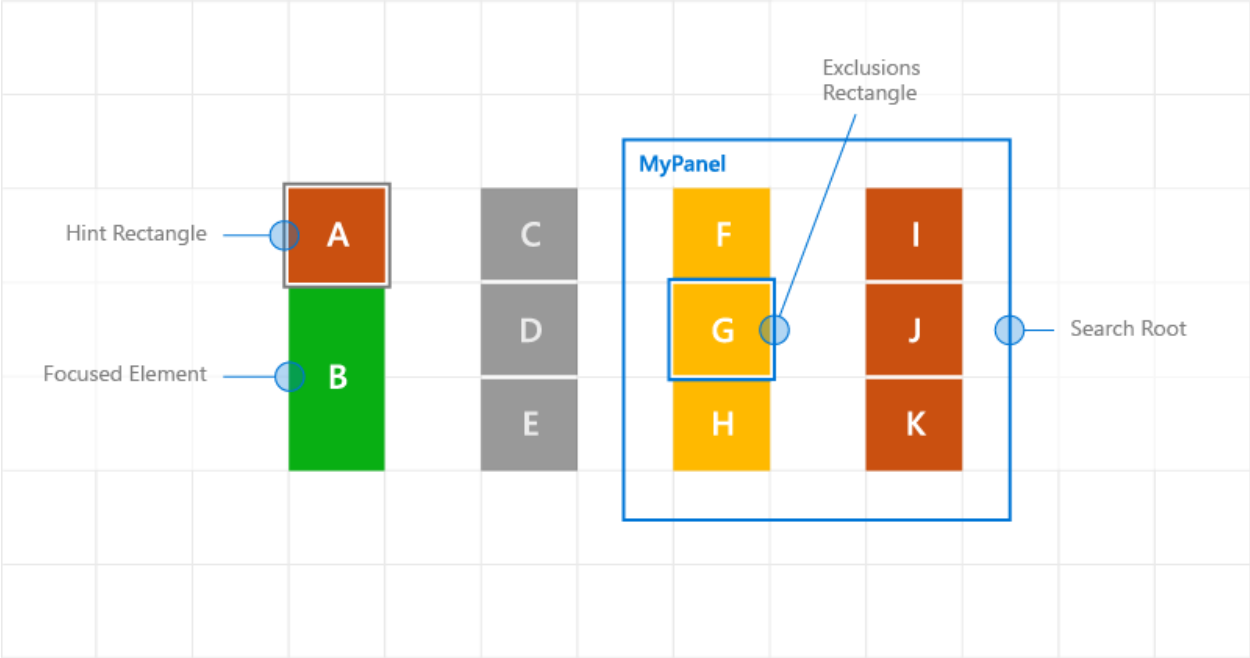
focus. This rectangle is used only for calculations and is never added to the visual tree.

- HintRect - Focus navigation candidates are identified using a "fictitious" bounding rectangle that identifies the elements most likely to receive focus. This rectangle is used only for calculations and is never added to the visual tree.
- XYFocusNavigationStrategyOverride - The focus navigation strategy used to identify the best candidate element to receive focus.

The following image illustrates some of these concepts.

When element B has focus, FindNextElement identifies I as the focus candidate when navigating to the right. The reasons for this are:

- Because of the HintRect on A, the starting reference is A, not B
- C is not a candidate because MyPanel has been specified as the SearchRoot
- F is not a candidate because the ExclusionRect overlaps it



Custom focus navigation behavior using navigation hints

Navigation focus events

NoFocusCandidateFound event

The UIElement.NoFocusCandidateFound event is fired when the tab or arrow keys are pressed and there is no focus candidate in the specified direction. This event is not fired for TryMoveFocus.

Because this is a routed event, it bubbles from the focused element up through successive parent objects to the root of the object tree. This lets you handle the event wherever appropriate.

Here, we show how a Grid opens a SplitView when the user attempts to move focus to the left of the left-most focusable control (see Designing for Xbox and TV).

XAML

```
<Grid NoFocusCandidateFound="OnNoFocusCandidateFound">
...
</Grid>
```

C#

```
private void OnNoFocusCandidateFound (
    UIElement sender, NoFocusCandidateFoundEventArgs args)
{
    if(args.NavigationDirection == FocusNavigationDirection.Left)
    {
        if(args.InputDevice == FocusInputDeviceKind.Keyboard ||
        args.InputDevice == FocusInputDeviceKind.GameController )
            {
                OpenSplitPaneView();
            }
        args.Handled = true;
    }
}
```

GotFocus and LostFocus events

The UIElement.GotFocus and UIElement.LostFocus events are fired when an element gets focus or loses focus, respectively. This event is not fired for TryMoveFocus.

Because these are routed events, they bubble from the focused element up through successive parent objects to the root of the object tree. This lets you handle the event wherever appropriate.

GettingFocus and LosingFocus events

The UIElement.GettingFocus and UIElement.LosingFocus events fire before the respective UIElement.GotFocus and UIElement.LostFocus events.

Because these are routed events, they bubble from the focused element up through successive parent objects to the root of the object tree. As this happens before a focus

change takes place, you can redirect or cancel the focus change.

GettingFocus and LosingFocus are synchronous events so focus won't be moved while these events are bubbling. However, GotFocus and LostFocus are asynchronous events, which means there is no guarantee that focus won't move again before the handler is executed.

If focus moves through a call to Control.Focus, GettingFocus is raised during the call, while GotFocus is raised after the call.

The focus navigation target can be changed during the GettingFocus and LosingFocus events (before focus moves) through the GettingFocusEventArgs.NewFocusedElement property. Even if the target is changed, the event still bubbles and the target can be changed again.

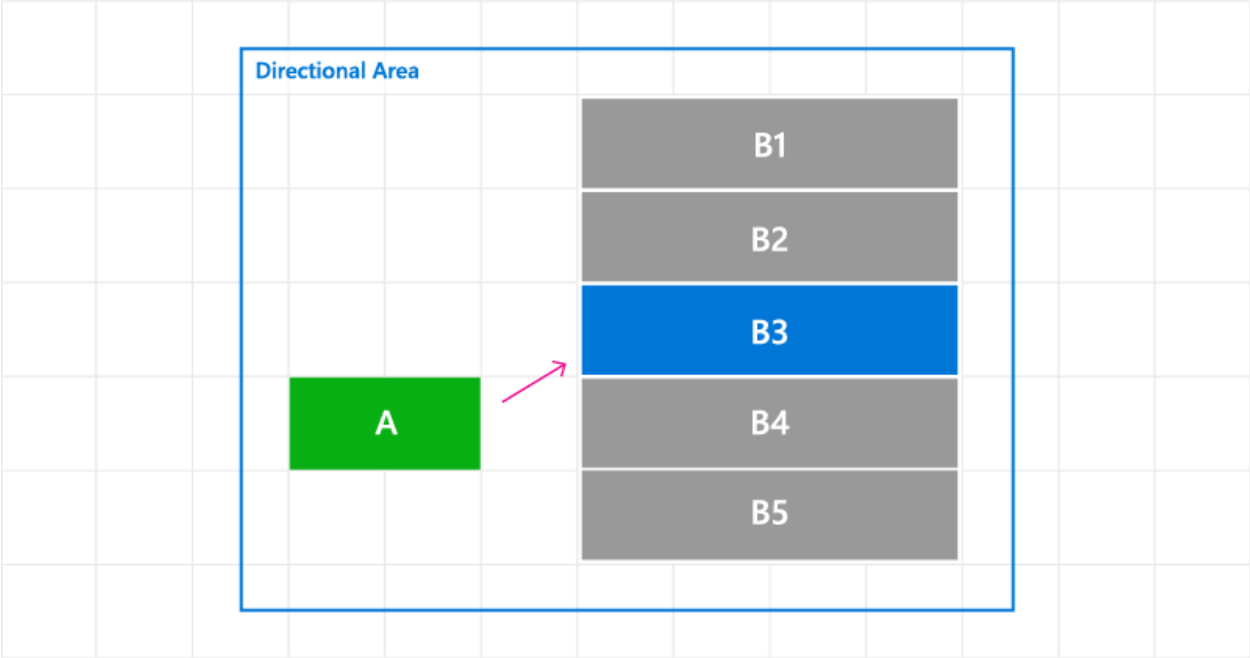
To avoid reentrancy issues, an exception is thrown if you try to move focus (using TryMoveFocus or Control.Focus) while these events are bubbling.

These events are fired regardless of the reason for the focus moving (including tab navigation, directional navigation, and programmatic navigation).

Here is the order of execution for the focus events:

1. LosingFocus If focus is reset back to the losing focus element or TryCancel is successful, no further events are fired.
2. GettingFocus If focus is reset back to the losing focus element or TryCancel is successful, no further events are fired.
3. LostFocus
4. GotFocus

The following image shows how, when moving to the right from A, the XYFocus chooses B4 as a candidate. B4 then fires the GettingFocus event where the ListView has the opportunity to reassign focus to B3.



Changing focus navigation target on GettingFocus event

Here, we show how to handle the GettingFocus event and redirect focus.

XAML

```
<StackPanel Orientation="Horizontal">
    <Button Content="A" />
    <ListView x:Name="MyListView" SelectedIndex="2"
GettingFocus="OnGettingFocus">
        <ListViewItem>LV1</ListViewItem>
        <ListViewItem>LV2</ListViewItem>
        <ListViewItem>LV3</ListViewItem>
        <ListViewItem>LV4</ListViewItem>
        <ListViewItem>LV5</ListViewItem>
    </ListView>
</StackPanel>
```

C#

```
private void OnGettingFocus(UIElement sender, GettingFocusEventArgs args)
{
    //Redirect the focus only when the focus comes from outside of the
ListView.
    // move the focus to the selected item
    if (MyListView.SelectedIndex != -1 &&
        IsNotAChildOf(MyListView, args.OldFocusedElement))
    {
        var selectedContainer =
            MyListView.ContainerFromItem(MyListView.SelectedItem);
        if (args.FocusState ==
            FocusState.Keyboard &&
            args.NewFocusedElement != selectedContainer)
        {
```

```
            args.TryRedirect(
                MyListView.ContainerFromItem(MyListView.SelectedItem));
            args.Handled = true;
        }
    }
}
```

Here, we show how to handle the LosingFocus event for a CommandBar and set focus when the menu is closed.

XAML

```
<CommandBar x:Name="MyCommandBar" LosingFocus="OnLosingFocus">
     <AppBarButton Icon="Back" Label="Back" />
     <AppBarButton Icon="Stop" Label="Stop" />
     <AppBarButton Icon="Play" Label="Play" />
     <AppBarButton Icon="Forward" Label="Forward" />

     <CommandBar.SecondaryCommands>
         <AppBarButton Icon="Like" Label="Like" />
         <AppBarButton Icon="Share" Label="Share" />
     </CommandBar.SecondaryCommands>
 </CommandBar>
```

C#

```
private void OnLosingFocus(UIElement sender, LosingFocusEventArgs args)
{
    if (MyCommandBar.IsOpen == true &&
        IsNotAChildOf(MyCommandBar, args.NewFocusedElement))
    {
        if (args.TryCancel())
        {
            args.Handled = true;
        }
    }
}
```

Find the first and last focusable element

The FocusManager.FindFirstFocusableElement and FocusManager.FindLastFocusableElement methods move focus to the first or last focusable element within the scope of an object (the element tree of a UIElement or the text tree of a TextElement). The scope is specified in the call (if the argument is null, the scope is the current window).

If no focus candidates can be identified in the scope, null is returned.

Here, we show how to specify that the buttons of a CommandBar have a wrap-around directional behavior (see Keyboard Interactions).

XAML

```
<CommandBar x:Name="MyCommandBar" LosingFocus="OnLosingFocus">
    <AppBarButton Icon="Back" Label="Back" />
    <AppBarButton Icon="Stop" Label="Stop" />
    <AppBarButton Icon="Play" Label="Play" />
    <AppBarButton Icon="Forward" Label="Forward" />

    <CommandBar.SecondaryCommands>
        <AppBarButton Icon="Like" Label="Like" />
        <AppBarButton Icon="ReShare" Label="Share" />
    </CommandBar.SecondaryCommands>
</CommandBar>
```

C#

```
private void OnLosingFocus(UIElement sender, LosingFocusEventArgs args)
{
    if (IsNotAChildOf(MyCommandBar, args.NewFocussedElement))
    {
        DependencyObject candidate = null;
        if (args.Direction == FocusNavigationDirection.Left)
        {
            candidate = FocusManager.FindLastFocusableElement(MyCommandBar);
        }
        else if (args.Direction == FocusNavigationDirection.Right)
        {
            candidate =
FocusManager.FindFirstFocusableElement(MyCommandBar);
        }
        if (candidate != null)
        {
            args.NewFocusedElement = candidate;
            args.Handled = true;
        }
    }
}
```

Related articles

- Focus navigation for keyboard, gamepad, remote control, and accessibility tools
- Keyboard interactions
- Keyboard accessibility

Respond to the presence of the touch keyboard

Article • 06/24/2021

Learn how to tailor the UI of your app when showing or hiding the touch keyboard.

Important APIs

- AutomationPeer
- InputPane

The touch keyboard in default layout mode

The touch keyboard enables text entry for devices that support touch. Windows app text input controls invoke the touch keyboard by default when a user taps on an editable input field. The touch keyboard typically remains visible while the user navigates between controls in a form, but this behavior can vary based on the other control types within the form.

To support corresponding touch keyboard behavior in a custom text input control that does not derive from a standard text input control, you must use the AutomationPeer class to expose your controls to Microsoft UI Automation and implement the correct UI Automation control patterns. See Keyboard accessibility and Custom automation peers.

Once this support has been added to your custom control, you can respond appropriately to the presence of the touch keyboard.

Prerequisites:

This topic builds on Keyboard interactions.

You should have a basic understanding of standard keyboard interactions, handling keyboard input and events, and UI Automation.

If you're new to developing Windows apps, have a look through these topics to get familiar with the technologies discussed here.

- Create your first app
- Learn about events with Events and routed events overview

User experience guidelines:

For helpful tips about designing a useful and engaging app optimized for keyboard input, see Keyboard interactions .

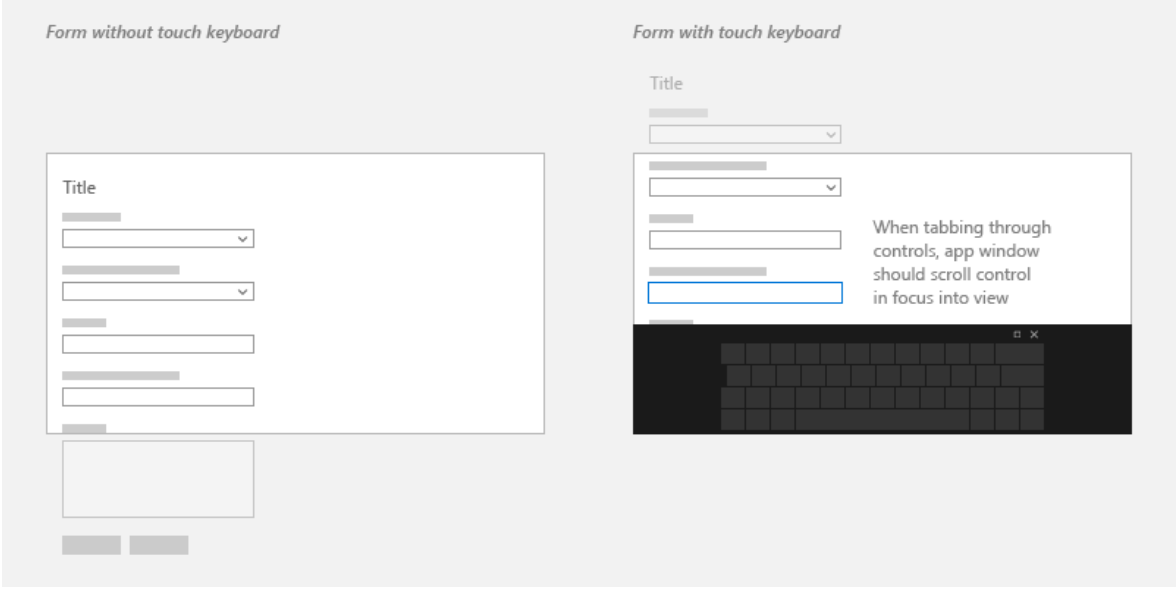
Touch keyboard and a custom UI

Here are a few basic recommendations for custom text input controls.

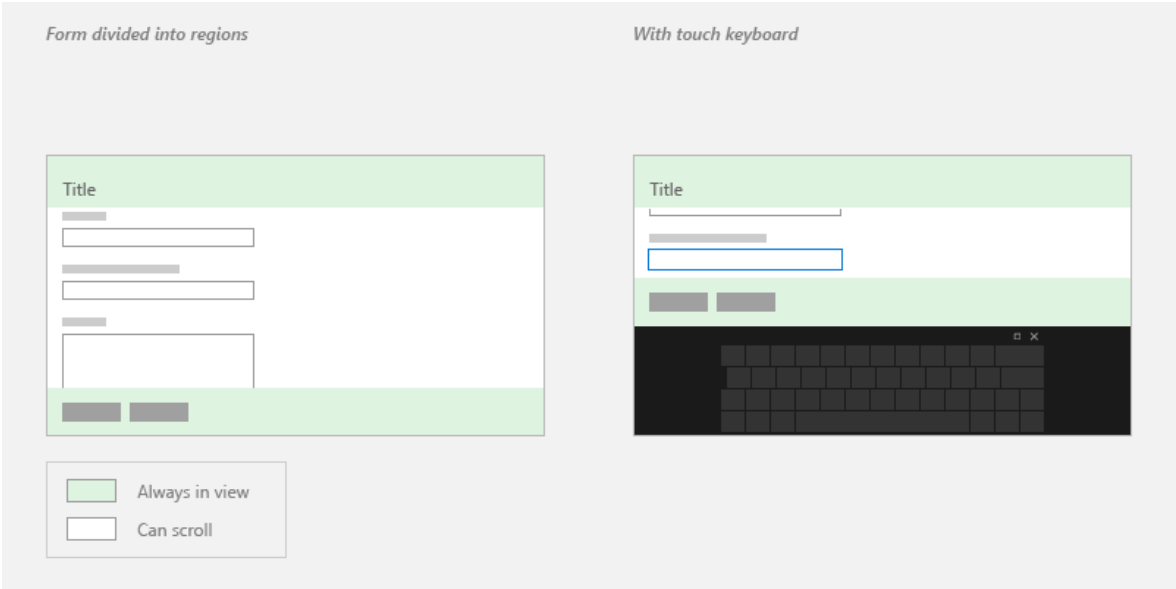
- Display the touch keyboard throughout the entire interaction with your form.
- Ensure that your custom controls have the appropriate UI Automation AutomationControlType for the keyboard to persist when focus moves from a text input field while in the context of text entry. For example, if you have a menu that's opened in the middle of a text-entry scenario, and you want the keyboard to persist, the menu must have the **AutomationControlType** of Menu.
- Don't manipulate UI Automation properties to control the touch keyboard. Other accessibility tools rely on the accuracy of UI Automation properties.
- Ensure that users can always see the input field that they're interacting with.

  Because the touch keyboard occludes a large portion of the screen, Windows ensures that the input field with focus scrolls into view as a user navigates through the controls on the form, including controls that are not currently in view.

  When customizing your UI, provide similar behavior on the appearance of the touch keyboard by handling the Showing and Hiding events exposed by the **InputPane** object.



In some cases, there are UI elements that should stay on the screen the entire time. Design the UI so that the form controls are contained in a panning region and the important UI elements are static. For example:



Handling the Showing and Hiding events

Here's an example of attaching event handlers for the Showing and Hiding events of the touch keyboard.

C#

```
using Windows.UI.ViewManagement;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;
using Windows.Foundation;
using Windows.UI.Xaml.Navigation;

namespace SDKTemplate
```

```
{
    /// <summary>
    /// Sample page to subscribe show/hide event of Touch Keyboard.
    /// </summary>
    public sealed partial class Scenario2_ShowHideEvents : Page
    {
        public Scenario2_ShowHideEvents()
        {
            this.InitializeComponent();
        }

        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            InputPane currentInputPane = InputPane.GetForCurrentView();
            // Subscribe to Showing/Hiding events
            currentInputPane.Showing += OnShowing;
            currentInputPane.Hiding += OnHiding;
        }

        protected override void OnNavigatedFrom(NavigationEventArgs e)
        {
            InputPane currentInputPane = InputPane.GetForCurrentView();
            // Unsubscribe from Showing/Hiding events
            currentInputPane.Showing -= OnShowing;
            currentInputPane.Hiding -= OnHiding;
        }

        void OnShowing(InputPane sender, InputPaneVisibilityEventArgs e)
        {
            LastInputPaneEventRun.Text = "Showing";
        }

        void OnHiding(InputPane sender, InputPaneVisibilityEventArgs e)
        {
            LastInputPaneEventRun.Text = "Hiding";
        }
    }
}
```

C++/WinRT

```
...
#include <winrt/Windows.UI.ViewManagement.h>
...
private:
    winrt::event_token m_showingEventToken;
    winrt::event_token m_hidingEventToken;
...
Scenario2_ShowHideEvents::Scenario2_ShowHideEvents()
{
    InitializeComponent();
}
```

```
void
Scenario2_ShowHideEvents::OnNavigatedTo(Windows::UI::Xaml::Navigation::Navig
ationEventArgs const& e)
{
    auto inputPane{
Windows::UI::ViewManagement::InputPane::GetForCurrentView() };
    // Subscribe to Showing/Hiding events
    m_showingEventToken = inputPane.Showing({ this,
&Scenario2_ShowHideEvents::OnShowing });
    m_hidingEventToken = inputPane.Hiding({ this,
&Scenario2_ShowHideEvents::OnHiding });
}

void
Scenario2_ShowHideEvents::OnNavigatedFrom(Windows::UI::Xaml::Navigation::Nav
igationEventArgs const& e)
{
    auto inputPane{
Windows::UI::ViewManagement::InputPane::GetForCurrentView() };
    // Unsubscribe from Showing/Hiding events
    inputPane.Showing(m_showingEventToken);
    inputPane.Hiding(m_hidingEventToken);
}

void
Scenario2_ShowHideEvents::OnShowing(Windows::UI::ViewManagement::InputPane
const& /*sender*/, Windows::UI::ViewManagement::InputPaneVisibilityEventArgs
const& /*args*/)
{
    LastInputPaneEventRun().Text(L"Showing");
}

void
Scenario2_ShowHideEvents::OnHiding(Windows::UI::ViewManagement::InputPane
const& /*sender*/, Windows::UI::ViewManagement::InputPaneVisibilityEventArgs
const& /*args*/)
{
    LastInputPaneEventRun().Text(L"Hiding");
}
```

C++

```
#include "pch.h"
#include "Scenario2_ShowHideEvents.xaml.h"

using namespace SDKTemplate;
using namespace Platform;
using namespace Windows::Foundation;
using namespace Windows::UI::ViewManagement;
using namespace Windows::UI::Xaml;
using namespace Windows::UI::Xaml::Controls;
using namespace Windows::UI::Xaml::Navigation;
```

```
Scenario2_ShowHideEvents::Scenario2_ShowHideEvents()
{
    InitializeComponent();
}

void Scenario2_ShowHideEvents::OnNavigatedTo(NavigationEventArgs^ e)
{
    auto inputPane = InputPane::GetForCurrentView();
    // Subscribe to Showing/Hiding events
    showingEventToken = inputPane->Showing +=
        ref new TypedEventHandler<InputPane^, InputPaneVisibilityEventArgs^>
(this, &Scenario2_ShowHideEvents::OnShowing);
    hidingEventToken = inputPane->Hiding +=
        ref new TypedEventHandler<InputPane^, InputPaneVisibilityEventArgs^>
(this, &Scenario2_ShowHideEvents::OnHiding);
}

void Scenario2_ShowHideEvents::OnNavigatedFrom(NavigationEventArgs^ e)
{
    auto inputPane =
Windows::UI::ViewManagement::InputPane::GetForCurrentView();
    // Unsubscribe from Showing/Hiding events
    inputPane->Showing -= showingEventToken;
    inputPane->Hiding -= hidingEventToken;
}

void Scenario2_ShowHideEvents::OnShowing(InputPane^ /*sender*/,
InputPaneVisibilityEventArgs^ /*args*/)
{
    LastInputPaneEventRun->Text = L"Showing";
}

void Scenario2_ShowHideEvents::OnHiding(InputPane^ /*sender*/,
InputPaneVisibilityEventArgs ^ /*args*/)
{
    LastInputPaneEventRun->Text = L"Hiding";
}
```

Related articles

- Keyboard interactions
- Keyboard accessibility
- Custom automation peers

Samples

- Touch keyboard sample

Archive samples

- Input: Touch keyboard sample
- Responding to the appearance of the on-screen keyboard sample
- XAML text editing sample
- XAML accessibility sample

Use input scope to change the touch keyboard

Article • 06/24/2021

To help users to enter data using the touch keyboard, or Soft Input Panel (SIP), you can set the input scope of the text control to match the kind of data the user is expected to enter.

Important APIs

- InputScope
- InputScopeNameValue

The touch keyboard can be used for text entry when your app runs on a device with a touch screen. The touch keyboard is invoked when the user taps on an editable input field, such as a **TextBox** or **RichEditBox**. You can make it much faster and easier for users to enter data in your app by setting the *input scope* of the text control to match the kind of data you expect the user to enter. The input scope provides a hint to the system about the type of text input expected by the control so the system can provide a specialized touch keyboard layout for the input type.

For example, if a text box is used only to enter a 4-digit PIN, set the **InputScope** property to **Number**. This tells the system to show the number keypad layout, which makes it easier for the user to enter the PIN.

> ⓘ **Important**
>
> - This info applies only to the SIP. It does not apply to hardware keyboards or the On-Screen Keyboard available in the Windows Ease of Access options.
> - The input scope does not cause any input validation to be performed, and does not prevent the user from providing any input through a hardware keyboard or other input device. You are still responsible for validating the input in your code as needed.

Changing the input scope of a text control

The input scopes that are available to your app are members of the InputScopeNameValue enumeration. You can set the **InputScope** property of a TextBox

or RichEditBox to one of these values.

> **ⓘ Important**
>
> The InputScope property on PasswordBox supports only the **Password** and **NumericPin** values. Any other value is ignored.

Here, you change the input scope of several text boxes to match the expected data for each text box.

To change the input scope in XAML

1. In the XAML file for your page, locate the tag for the text control that you want to change.
2. Add the InputScope attribute to the tag and specify the InputScopeNameValue value that matches the expected input.

   Here are some text boxes that might appear on a common customer-contact form. With the InputScope set, a touch keyboard with a suitable layout for the data shows for each text box.

   XAML

   ```
   <StackPanel Width="300">
       <TextBox Header="Name" InputScope="Default"/>
       <TextBox Header="Email Address" InputScope="EmailSmtpAddress"/>
       <TextBox Header="Telephone Number" InputScope="TelephoneNumber"/>
       <TextBox Header="Web site" InputScope="Url"/>
   </StackPanel>
   ```

To change the input scope in code

1. In the XAML file for your page, locate the tag for the text control that you want to change. If it's not set, set the x:Name attribute so you can reference the control in your code.

   C#

   ```
   <TextBox Header="Telephone Number" x:Name="phoneNumberTextBox"/>
   ```

2. Instantiate a new InputScope object.

   C#

```
InputScope scope = new InputScope();
```

3. Instantiate a new **InputScopeName** object.

```
C#

InputScopeName scopeName = new InputScopeName();
```

4. Set the **NameValue** property of the **InputScopeName** object to a value of the **InputScopeNameValue** enumeration.

```
C#

scopeName.NameValue = InputScopeNameValue.TelephoneNumber;
```

5. Add the **InputScopeName** object to the **Names** collection of the **InputScope** object.

```
C#

scope.Names.Add(scopeName);
```

6. Set the **InputScope** object as the value of the text control's **InputScope** property.

```
C#

phoneNumberTextBox.InputScope = scope;
```

Here's the code all together.

```
CSharp

InputScope scope = new InputScope();
InputScopeName scopeName = new InputScopeName();
scopeName.NameValue = InputScopeNameValue.TelephoneNumber;
scope.Names.Add(scopeName);
phoneNumberTextBox.InputScope = scope;
```

The same steps can be condensed into this shorthand code.

```
CSharp

phoneNumberTextBox.InputScope = new InputScope()
{
```

```
    Names = {new InputScopeName(InputScopeNameValue.TelephoneNumber)}
};
```

Text prediction, spell checking, and auto-correction

The TextBox and RichEditBox controls have several properties that influence the behavior of the SIP. To provide the best experience for your users, it's important to understand how these properties affect text input using touch.

- IsSpellCheckEnabled—When spell check is enabled for a text control, the control interacts with the system's spell-check engine to mark words that are not recognized. You can tap a word to see a list of suggested corrections. Spell checking is enabled by default.

  For the **Default** input scope, this property also enables automatic capitalization of the first word in a sentence and auto-correction of words as you type. These auto-correction features might be disabled in other input scopes. For more info, see the tables later in this topic.

- IsTextPredictionEnabled—When text prediction is enabled for a text control, the system shows a list of words that you might be beginning to type. You can select from the list so you don't have to type the whole word. Text prediction is enabled by default.

  Text prediction might be disabled if the input scope is other than **Default**, even if the IsTextPredictionEnabled property is **true**. For more info, see the tables later in this topic.

- PreventKeyboardDisplayOnProgrammaticFocus—When this property is **true**, it prevents the system from showing the SIP when focus is programmatically set on a text control. Instead, the keyboard is shown only when the user interacts with the control.

Touch keyboard index for Windows

These tables show the Windows Soft Input Panel (SIP) layouts for common input scope values. The effect of the input scope on the features enabled by the **IsSpellCheckEnabled** and **IsTextPredictionEnabled** properties is listed for each input scope. This is not a comprehensive list of available input scopes.

> **Tip**
>
> You can toggle most touch keyboards between an alphabetic layout and a numbers-and-symbols layout by pressing the **&123** key to change to the numbers-and-symbols layout, and press the **abcd** key to change to the alphabetic layout.

Default

`<TextBox InputScope="Default"/>`

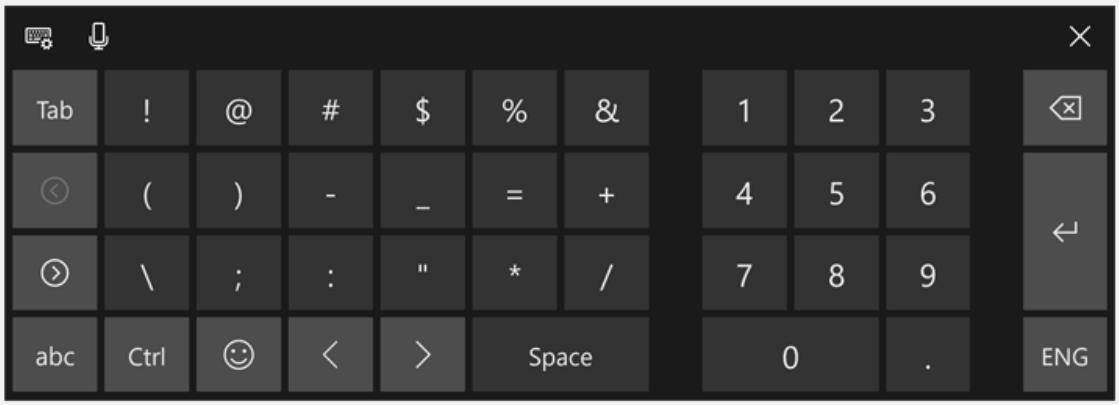
The default Windows touch keyboard.

- Spell check: enabled if **IsSpellCheckEnabled** = **true**, disabled if **IsSpellCheckEnabled** = **false**
- Auto-correction: enabled if **IsSpellCheckEnabled** = **true**, disabled if **IsSpellCheckEnabled** = **false**
- Automatic capitalization: enabled if **IsSpellCheckEnabled** = **true**, disabled if **IsSpellCheckEnabled** = **false**
- Text prediction: enabled if **IsTextPredictionEnabled** = **true**, disabled if **IsTextPredictionEnabled** = **false**

CurrencyAmountAndSymbol

`<TextBox InputScope="CurrencyAmountAndSymbol"/>`

The default numbers and symbols keyboard layout.



- Includes page left/right keys to show more symbols
- Spell check: on by default, can be disabled
- Auto-correction: on by default, can be disabled
- Automatic capitalization: always disabled
- Text prediction: on by default, can be disabled

Url

`<TextBox InputScope="Url"/>`

- Includes the **.com** and → (Go) keys. Press and hold the **.com** key to display additional options (**.org**, **.net**, and region-specific suffixes)
- Includes the **:**, **-**, and **/** keys
- Spell check: off by default, can be enabled
- Auto-correction: off by default, can be enabled

- Automatic capitalization: off by default, can be enabled
- Text prediction: off by default, can be enabled

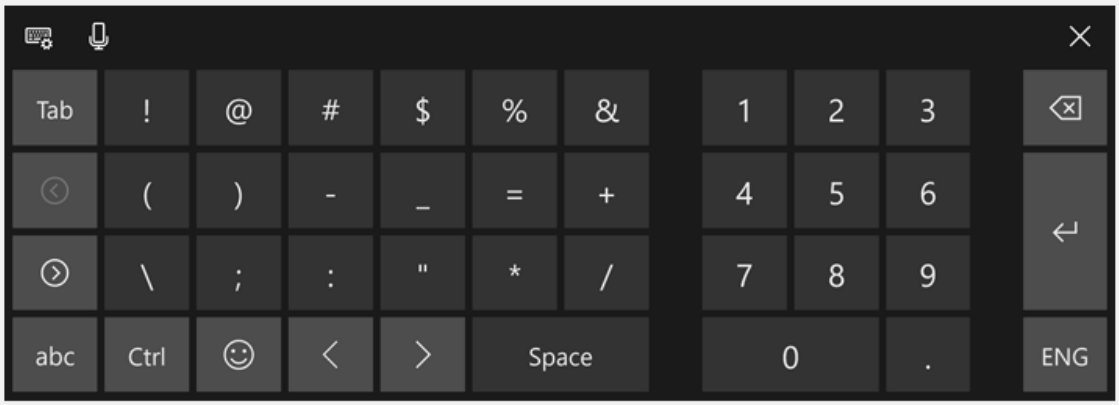
EmailSmtpAddress

```
<TextBox InputScope="EmailSmtpAddress"/>
```

- Includes the @ and **.com** keys. Press and hold the **.com** key to display additional options (**.org**, **.net**, and region-specific suffixes)
- Includes the _ and - keys
- Spell check: off by default, can be enabled
- Auto-correction: off by default, can be enabled
- Automatic capitalization: off by default, can be enabled
- Text prediction: off by default, can be enabled

Number

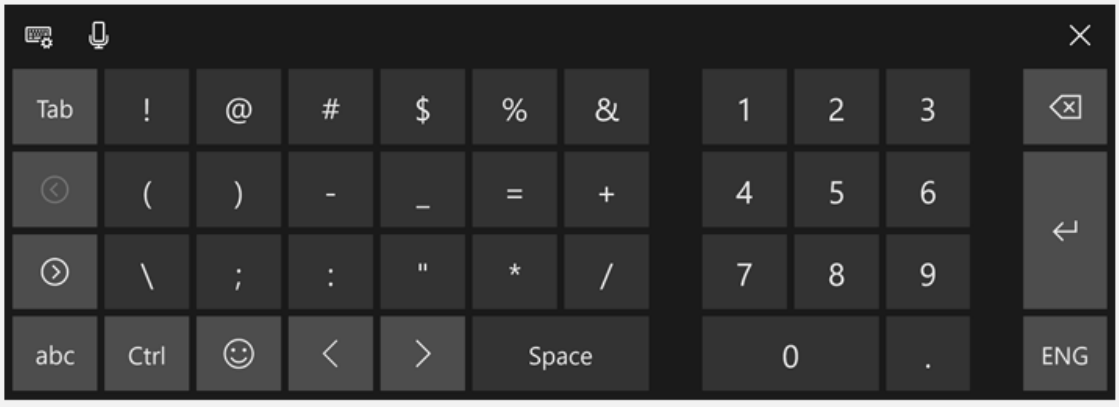
```
<TextBox InputScope="Number"/>
```



- Spell check: on by default, can be disabled
- Auto-correction: on by default, can be disabled
- Automatic capitalization: always disabled
- Text prediction: on by default, can be disabled

TelephoneNumber

`<TextBox InputScope="TelephoneNumber"/>`



- Spell check: on by default, can be disabled
- Auto-correction: on by default, can be disabled
- Automatic capitalization: always disabled
- Text prediction: on by default, can be disabled

Search

`<TextBox InputScope="Search"/>`

- Includes the **Search** key instead of the **Enter** key
- Spell check: on by default, can be disabled
- Auto-correction: on by default, can be disabled
- Auto-capitalization: always disabled
- Text prediction: on by default, can be disabled

SearchIncremental

`<TextBox InputScope="SearchIncremental"/>`

- Same layout as **Default**

- Spell check: off by default, can be enabled
- Auto-correction: always disabled
- Automatic capitalization: always disabled
- Text prediction: always disabled

Formula

```
<TextBox InputScope="Formula"/>
```

- Includes the = key
- Also includes the %, $, and + keys
- Spell check: on by default, can be disabled
- Auto-correction: on by default, can be disabled
- Automatic capitalization: always disabled
- Text prediction: on by default, can be disabled

Chat

```
<TextBox InputScope="Chat"/>
```

- Same layout as **Default**
- Spell check: on by default, can be disabled
- Auto-correction: on by default, can be disabled
- Automatic capitalization: on by default, can be disabled
- Text prediction: on by default, can be disabled

NameOrPhoneNumber

```
<TextBox InputScope="NameOrPhoneNumber"/>
```

- Same layout as **Default**
- Spell check: off by default, can be enabled
- Auto-correction: off by default, can be enabled
- Automatic capitalization: off by default, can be enabled (first letter of each word is capitalized)

- Text prediction: off by default, can be enabled

Custom text input

Article • 06/24/2021

The core text APIs in the **Windows.UI.Text.Core** namespace enable a Windows app to receive text input from any text service supported on Windows devices. The APIs are similar to the Text Services Framework APIs in that the app is not required to have detailed knowledge of the text services. This enables the app to receive text in any language and from any input type, like keyboard, speech, or pen.

> **Important APIs**: **Windows.UI.Text.Core**, **CoreTextEditContext**

Why use core text APIs?

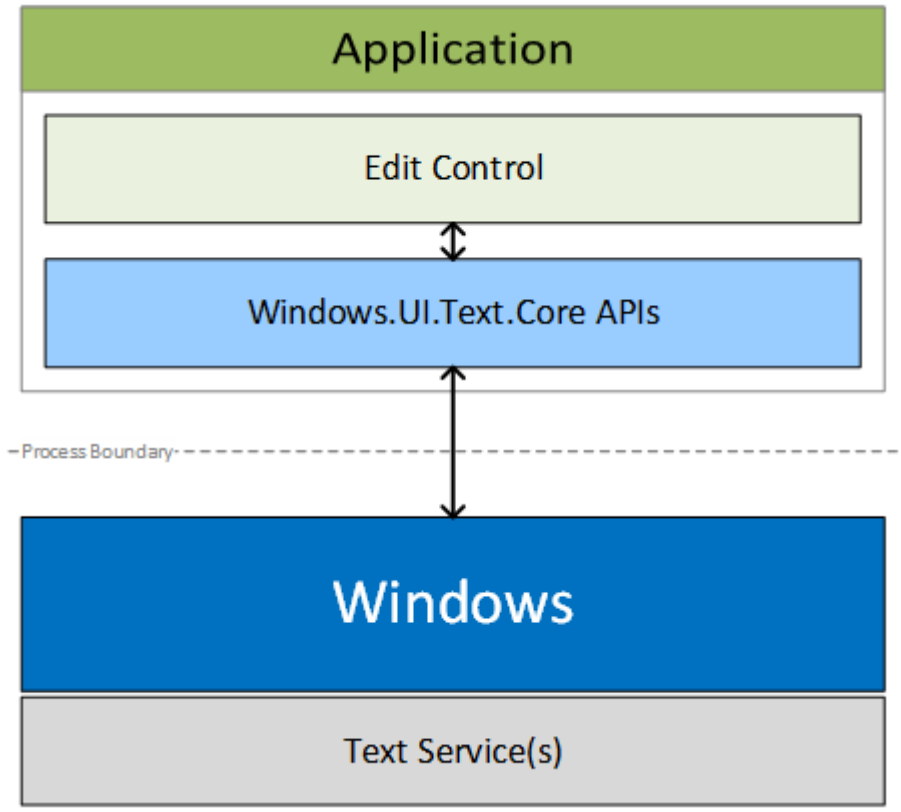
For many apps, the XAML or HTML text box controls are sufficient for text input and editing. However, if your app handles complex text scenarios, like a word processing app, you might need the flexibility of a custom text edit control. You could use the **CoreWindow** keyboard APIs to create your text edit control, but these don't provide a way to receive composition-based text input, which is required to support East Asian languages.

Instead, use the **Windows.UI.Text.Core** APIs when you need to create a custom text edit control. These APIs are designed to give you a lot of flexibility in processing text input, in any language, and let you provide the text experience best suited to your app. Text input and edit controls built with the core text APIs can receive text input from all existing text input methods on Windows devices, from Text Services Framework based Input Method Editors (IMEs) and handwriting on PCs to the WordFlow keyboard (which provides auto-correction, prediction, and dictation) on mobile devices.

Architecture

The following is a simple representation of the text input system.

- "Application" represents a Windows app hosting a custom edit control built using the core text APIs.
- The **Windows.UI.Text.Core** APIs facilitate the communication with text services through Windows. Communication between the text edit control and the text services is handled primarily through a **CoreTextEditContext** object that provides the methods and events to facilitate the communication.



Text ranges and selection

Edit controls provide space for text entry and users expect to edit text anywhere in this space. Here, we explain the text positioning system used by the core text APIs and how ranges and selections are represented in this system.

Application caret position

Text ranges used with the core text APIs are expressed in terms of caret positions. An "Application Caret Position (ACP)" is a zero-based number that indicates the count of characters from the start of the text stream immediately before the caret, as shown here.

Text Stream	H	e	l	l	o	_	W	o	r	l	d
ACP	0	1	2	3	4	5	6	7	8	9	10	11

Text ranges and selection

Text ranges and selections are represented by the CoreTextRange structure which contains two fields:

Field	Data type	Description

Field	Data type	Description
StartCaretPosition	**Number** [JavaScript] | **System.Int32** [.NET] | **int32** [C++]	The start position of a range is the ACP immediately before the first character.
EndCaretPosition	**Number** [JavaScript] | **System.Int32** [.NET] | **int32** [C++]	The end position of a range is the ACP immediately after the last character.

For example, in the text range shown previously, the range [0, 5] specifies the word "Hello". **StartCaretPosition** must always be less than or equal to the **EndCaretPosition**. The range [5, 0] is invalid.

Insertion point

The current caret position, frequently referred to as the insertion point, is represented by setting the **StartCaretPosition** to be equal to the **EndCaretPosition**.

Noncontiguous selection

Some edit controls support noncontiguous selections. For example, Microsoft Office apps support multiple arbitrary selections, and many source code editors support column selection. However, the core text APIs do not support non-contiguous selections. Edit controls must report only a single contiguous selection, most often the active sub-range of the noncontiguous selections.

For example, the following image shows a text stream with two non-contiguous selections: [0, 1] and [6, 11] for which the edit control must report only one (either [0, 1] or [6, 11]).

Working with text

The CoreTextEditContext class enables text flow between Windows and edit controls through the TextUpdating event, the TextRequested event, and the NotifyTextChanged method.

Your edit control receives text through TextUpdating events that are generated when users interact with text input methods like keyboards, speech, or IMEs.

When you change text in your edit control, for example, by pasting text into the control, you need to notify Windows by calling NotifyTextChanged.

If the text service requires the new text, then a TextRequested event is raised. You must provide the new text in the **TextRequested** event handler.

Accepting text updates

Your edit control should typically accept text update requests because they represent the text the user wants to enter. In the TextUpdating event handler, these actions are expected of your edit control:

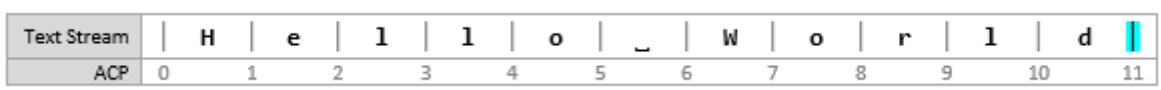
1. Insert the text specified in CoreTextTextUpdatingEventArgs.Text in the position specified in CoreTextTextUpdatingEventArgs.Range.
2. Place selection at the position specified in CoreTextTextUpdatingEventArgs.NewSelection.
3. Notify the system that the update succeeded by setting CoreTextTextUpdatingEventArgs.Result to CoreTextTextUpdatingResult.Succeeded.

For example, this is the state of an edit control before the user types "d". The insertion point is at [10, 10].

When the user types "d", a TextUpdating event is raised with the following CoreTextTextUpdatingEventArgs data:

- Range = [10, 10]
- Text = "d"
- NewSelection = [11, 11]

In your edit control, apply the specified changes and set Result to **Succeeded**. Here's the state of the control after the changes are applied.



Rejecting text updates

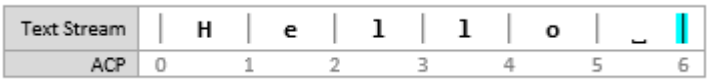
Sometimes, you cannot apply text updates because the requested range is in an area of the edit control that should not be changed. In this case, you should not apply any changes. Instead, notify the system that the update failed by setting CoreTextTextUpdatingEventArgs.Result to CoreTextTextUpdatingResult.Failed.

For example, consider an edit control that accepts only an e-mail address. Spaces should be rejected because e-mail addresses cannot contain spaces, so when TextUpdating events are raised for the space key, you should simply set Result to **Failed** in your edit control.

Notifying text changes

Sometimes, your edit control makes changes to text such as when text is pasted or auto-corrected. In these cases, you must notify the text services of these changes by calling the NotifyTextChanged method.

For example, this is the state of an edit control before the user pastes "World". The insertion point is at [6, 6].



The user performs the paste action and the edit control after the changes are applied:

When this happens, you should call NotifyTextChanged with these arguments:

- *modifiedRange* = [6, 6]
- *newLength* = 5
- *newSelection* = [11, 11]

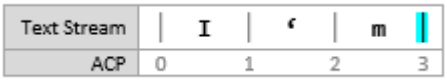
One or more TextRequested events will follow, which you handle to update the text that the text services are working with.

Overriding text updates

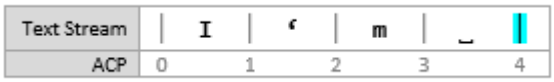
In your edit control, you might want to override a text update to provide auto-correction features.

For example, consider an edit control that provides a correction feature that formalizes contractions. This is the state of the edit control before the user types the space key to

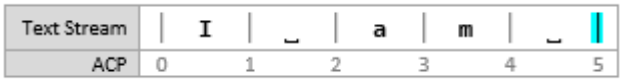
trigger the correction. The insertion point is at [3, 3].



The user presses the space key and a corresponding **TextUpdating** event is raised. The edit control accepts the text update. This is the state of the edit control for a brief moment before the correction is completed. The insertion point is at [4, 4].



Outside of the **TextUpdating** event handler, the edit control makes the following correction. This is the state of the edit control after the correction is complete. The insertion point is at [5, 5].



When this happens, you should call **NotifyTextChanged** with these arguments:

- *modifiedRange* = [1, 2]
- *newLength* = 2
- *newSelection* = [5, 5]

One or more **TextRequested** events will follow, which you handle to update the text that the text services are working with.

Providing requested text

It's important for text services to have the correct text to provide features like auto-correction or prediction, especially for text that already existed in the edit control, from loading a document, for example, or text that is inserted by the edit control as explained in previous sections. Therefore, whenever a **TextRequested** event is raised, you must provide the text currently in your edit control for the specified range.

There will be times the **Range** in **CoreTextTextRequest** specifies a range that your edit control cannot accommodate as-is. For example, the **Range** is larger than the size of the edit control at the time of the **TextRequested** event, or the end of the **Range** is out of bounds. In these cases, you should return whatever range makes sense, which is typically a subset of the requested range.

Related articles

Samples

- Custom Edit Control sample

Archive samples

- XAML text editing sample

Text scaling

Article • 05/13/2022

Example of text scaling in Windows 11 (100% to 225%)

Overview

Reading text on a computer screen (from mobile device to laptop to desktop monitor to the giant screen of a Surface Hub) can be challenging for many people. Conversely, some users find the font sizes used in apps and web sites to be larger than necessary.

To ensure text is as legible as possible for the broadest range of users, Windows provides the ability for users to change relative font size across both the OS and individual applications. Instead of using a magnifier app (which typically just enlarges everything within an area of the screen and introduces its own usability issues), changing display resolution, or relying on DPI scaling (which resizes everything based on display and typical viewing distance), a user can quickly access a setting to resize just text, ranging from 100% (the default size) up to 225%.

Support

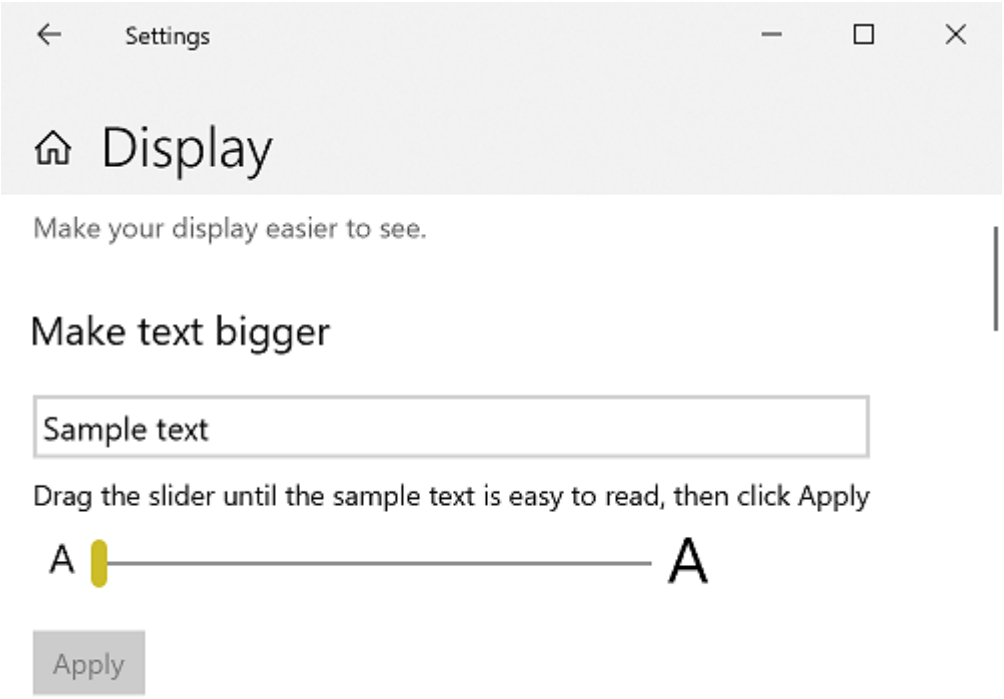
Universal Windows applications (both standard and PWA), support text scaling by default.

If your Windows application includes custom controls, custom text surfaces, hard-coded control heights, older frameworks, or 3rd party frameworks, you likely have to make some updates to ensure a consistent and useful experience for your users.

DirectWrite, GDI, and XAML SwapChainPanels do not natively support text scaling, while Win32 support is limited to menus, icons, and toolbars.

User experience

Users can adjust text scale with the Make text bigger slider on the Settings -> Ease of Access -> Vision/Display screen.



Text scale setting from Settings -> Ease of Access -> Vision/Display screen

UX guidance

As text is resized, controls and containers must also resize and reflow to accommodate the text and its new layout. As mentioned previously, depending on the app, framework, and platform, much of this work is done for you. The following UX guidance covers those cases where it's not.

Use the platform controls

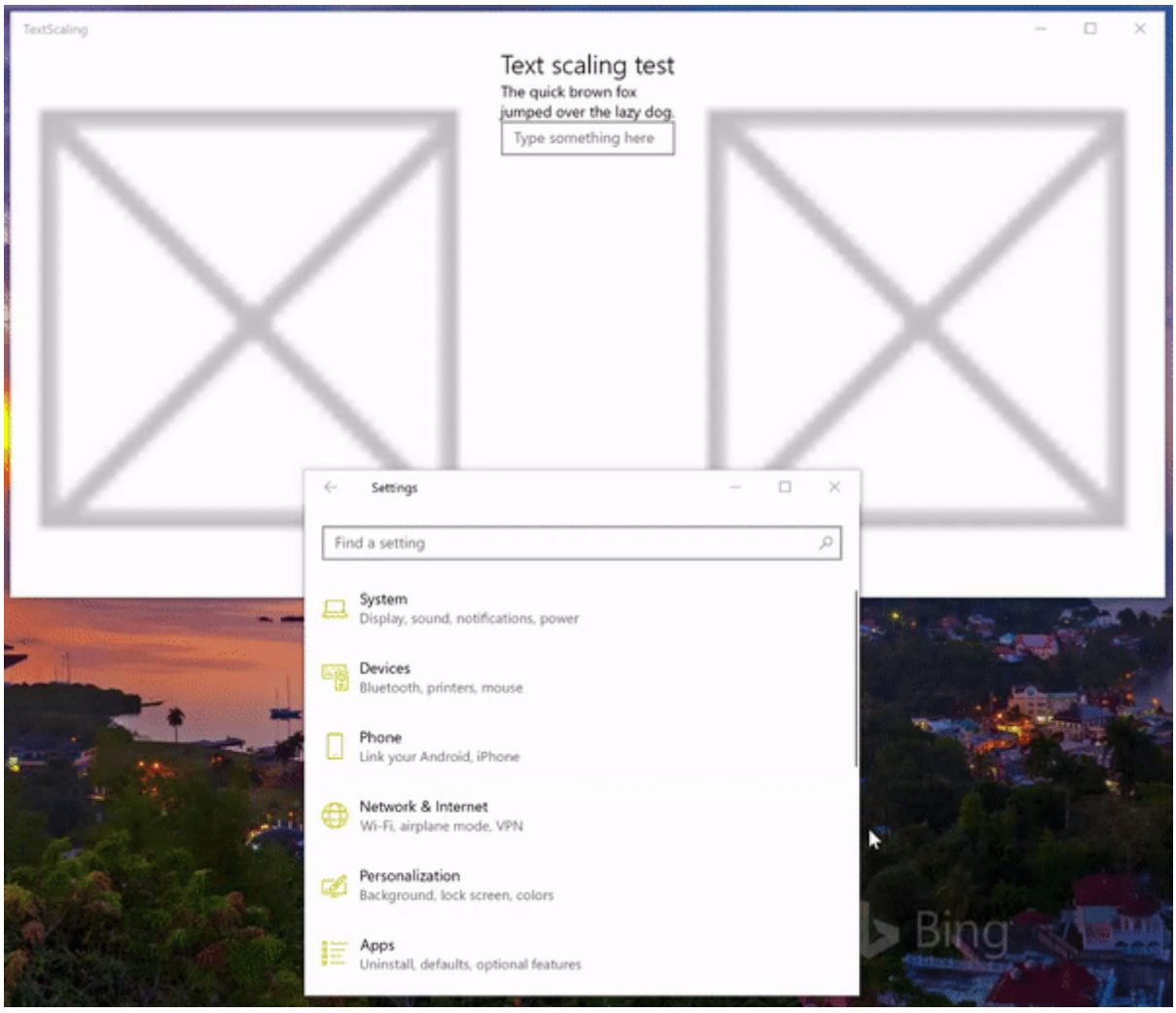
Did we say this already? It's worth repeating: When possible, always use the built-in controls provided with the various Windows app frameworks to get the most comprehensive user experience possible for the least amount of effort.

For example, all UWP text controls support the full text scaling experience without any customization or templating.

Here's a snippet from a basic UWP app that includes a couple of standard text controls:

XAML

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <TextBlock Grid.Row="0"
               Style="{ThemeResource TitleTextBlockStyle}"
               Text="Text scaling test"
               HorizontalTextAlignment="Center" />
    <Grid Grid.Row="1">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="Auto"/>
        </Grid.ColumnDefinitions>
        <Image Grid.Column="0"
               Source="Assets/StoreLogo.png"
               Width="450" Height="450"/>
        <StackPanel Grid.Column="1"
                    HorizontalAlignment="Center">
            <TextBlock TextWrapping="WrapWholeWords">
                The quick brown fox jumped over the lazy dog.
            </TextBlock>
            <TextBox PlaceholderText="Type something here" />
        </StackPanel>
        <Image Grid.Column="2"
               Source="Assets/StoreLogo.png"
               Width="450" Height="450"/>
    </Grid>
    <TextBlock Grid.Row="2"
               Style="{ThemeResource TitleTextBlockStyle}"
               Text="Text scaling test footer"
               HorizontalTextAlignment="Center" />
</Grid>
```



Animated text scaling

Use auto-sizing

Don't specify absolute sizes for your controls. Whenever possible, let the platform resize your controls automatically based on user and device settings.

In this snippet from the previous example, we use the `Auto` and `*` width values for a set of grid columns and let the platform adjust the app layout based on the size of the elements contained within the grid.

XAML

```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto"/>
    <ColumnDefinition Width="*"/>
    <ColumnDefinition Width="Auto"/>
</Grid.ColumnDefinitions>
```

Use text wrapping

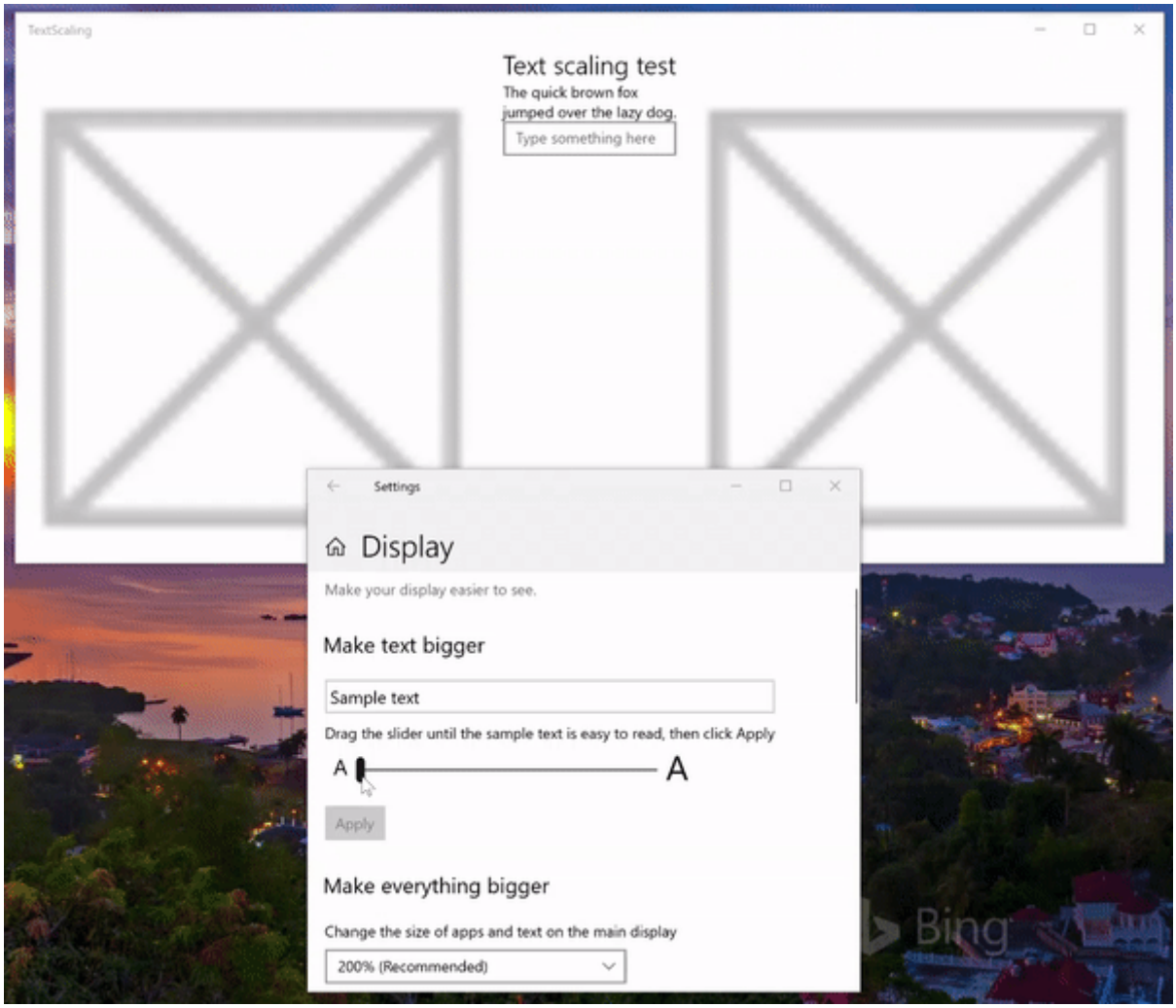
To ensure the layout of your app is as flexible and adaptable as possible, enable text wrapping in any control that contains text (many controls do not support text wrapping by default).

If you don't specify text wrapping, the platform uses other methods to adjust the layout, including clipping (see previous example).

Here, we use the `AcceptsReturn` and `TextWrapping` TextBox properties to ensure our layout is as flexible as possible.

XAML

```
<TextBox PlaceholderText="Type something here"
         AcceptsReturn="True" TextWrapping="Wrap" />
```



Animated text scaling with text wrapping

Specify text trimming behavior

If text wrapping is not the preferred behavior, most text controls let you either clip your text or specify ellipses for the text trimming behavior. Clipping is preferred to ellipses as

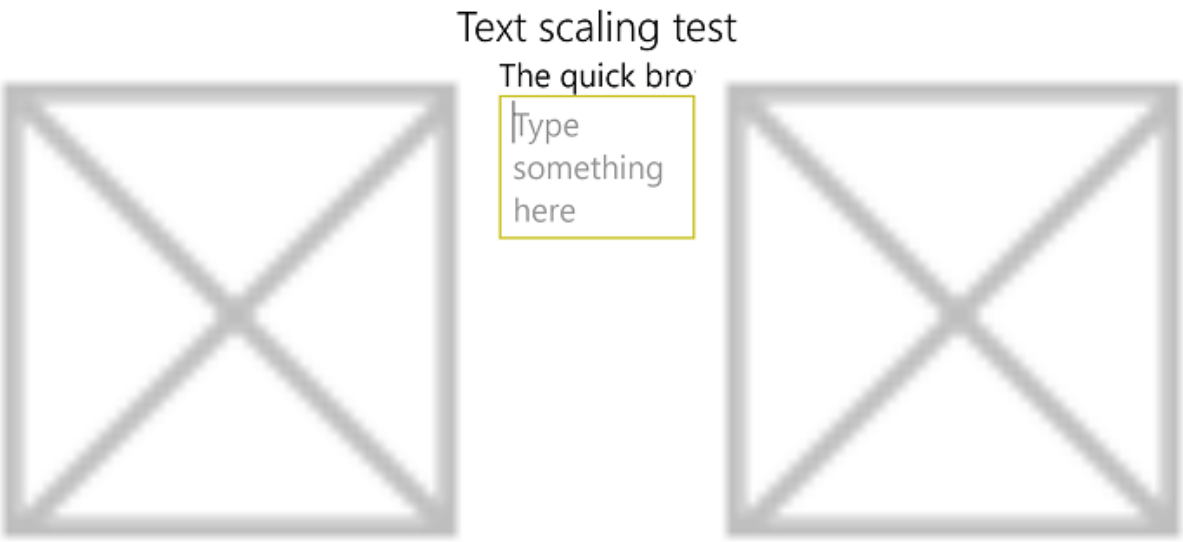
ellipses take up space themselves.

> **Note**
>
> If you need to clip your text, clip the end of the string, not the beginning.

In this example, we show how to clip text in a TextBlock using the TextTrimming property.

XAML

```
<TextBlock TextTrimming="Clip">
    The quick brown fox jumped over the lazy dog.
</TextBlock>
```



Text scaling with text clipping

Use a tooltip

If you clip text, use a tooltip to provide the full text to your users.

Here, we add a tooltip to a TextBlock that doesn't support text wrapping:

XAML

```
<TextBlock TextTrimming="Clip">
    <ToolTipService.ToolTip>
        <ToolTip Content="The quick brown fox jumped over the lazy dog."/>
    </ToolTipService.ToolTip>
    The quick brown fox jumped over the lazy dog.
</TextBlock>
```

Don’t scale font-based icons or symbols

When using font-based icons for emphasis or decoration, disable scaling on these characters.

Set the IsTextScaleFactorEnabled property to `false` for most XAML controls.

Support text scaling natively

Handle the TextScaleFactorChanged UISettings system event in your custom framework and controls. This event is raised each time the user sets the text scaling factor on their system.

Summary

This topic provides an overview of text scaling support in Windows and includes UX and developer guidance on how to customize the user experience.

Related articles

API reference

- IsTextScaleFactorEnabled
- TextScaleFactorChanged

Selecting text and images

Article • 06/24/2021

This article describes selecting and manipulating text, images, and controls and provides user experience guidelines that should be considered when using these mechanisms in your apps.

> **Important APIs**: Windows.UI.Xaml.Input, Windows.UI.Input

Dos and don'ts

- Use font glyphs when implementing your own gripper UI. The gripper is a combination of two Segoe UI fonts that are available system-wide. Using font resources simplifies rendering issues at different dpi and works well with the various UI scaling plateaus. When implementing your own grippers, they should share the following UI traits:
  - Circular shape
  - Visible against any background
  - Consistent size

- Provide a margin around the selectable content to accommodate the gripper UI. If your app enables text selection in a region that doesn't pan/scroll, allow a 1/2 gripper margin on the left and right sides of the text area and 1 gripper height on the top and bottom sides of the text area (as shown in the following images). This ensures that the entire gripper UI is exposed to the user and minimizes unintended interactions with other edge-based UI.

- Hide grippers UI during interaction. Eliminates occlusion by the grippers during the interaction. This is useful when a gripper isn't completely obscured by the finger or there are multiple text selection grippers. This eliminates visual artifacts when displaying child windows.

- Don't allow selection of UI elements such as controls, labels, images, proprietary content, and so on. Typically, Windows applications allow selection only within specific controls. Controls such as buttons, labels, and logos are not selectable. Assess whether selection is an issue for your app and, if so, identify the areas of the UI where selection should be prohibited.

Additional usage guidance

Text selection and manipulation is particularly susceptible to user experience challenges introduced by touch interactions. Mouse, pen/stylus, and keyboard input are highly granular: a mouse click or pen/stylus contact is typically mapped to a single pixel, and a key is pressed or not pressed. Touch input is not granular; it's difficult to map the entire surface of a fingertip to a specific x-y location on the screen to place a text caret accurately.

Considerations and recommendations

Use the built-in controls exposed through the language frameworks in Windows to build apps that provide the full platform user interaction experience, including selection and manipulation behaviors. You'll find the interaction functionality of the built-in controls sufficient for the majority of Windows apps.

When using standard Windows text controls, the selection behaviors and visuals described in this topic cannot be customized.

Text selection

If your app requires a custom UI that supports text selection, we recommend that you follow the Windows selection behaviors described here.

Editable and non-editable content

With touch, selection interactions are performed primarily through gestures such as a tap to set an insertion cursor or select a word, and a slide to modify a selection. As with other Windows touch interactions, timed interactions are limited to the press and hold gesture to display informational UI. For more information, see Guidelines for visual feedback.

Windows recognizes two possible states for selection interactions, editable and non-editable, and adjusts selection UI, feedback, and functionality accordingly.

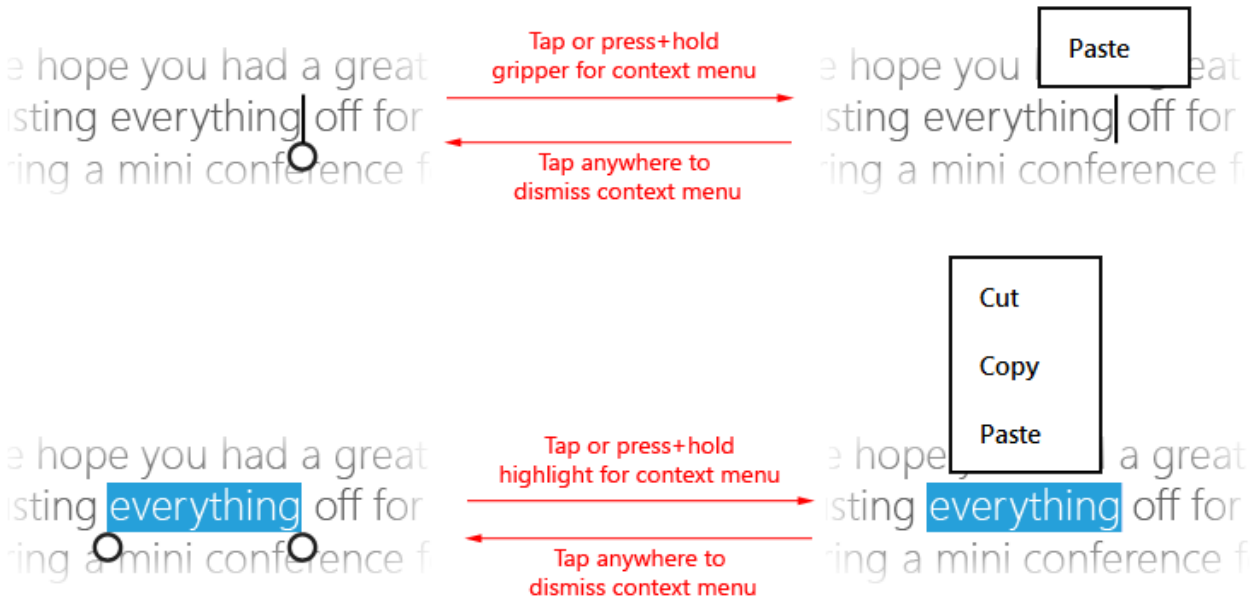
Editable content

Tapping within the left half of a word places the cursor to the immediate left of the word, while tapping within the right half places the cursor to the immediate right of the word.

The following image demonstrates how to place an initial insertion cursor with gripper by tapping near the beginning or ending of a word.

The following image demonstrates how to adjust a selection by dragging the gripper.

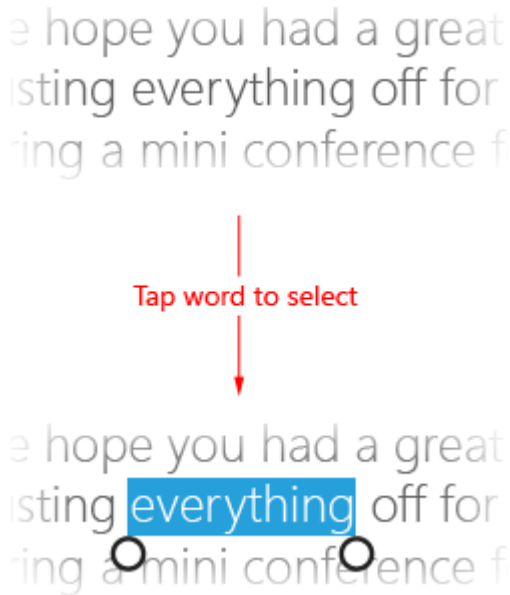
The following images demonstrate how to invoke the context menu by tapping within the selection or on a gripper (press and hold can also be used).



Note These interactions vary somewhat in the case of a misspelled word. Tapping a word that is marked as misspelled will both highlight the entire word and invoke the suggested spelling context menu.

Non-editable content

The following image demonstrates how to select a word by tapping within the word (no spaces are included in the initial selection).

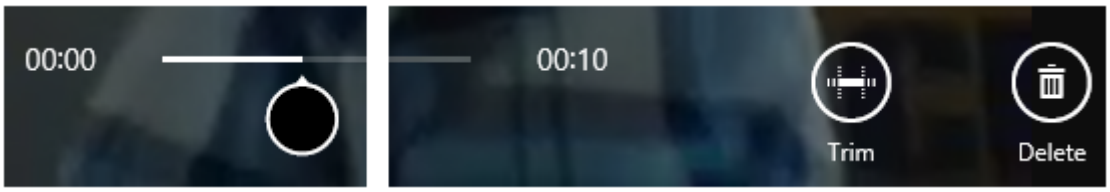


Follow the same procedures as for editable text to adjust the selection and display the context menu.

Object manipulation

Wherever possible, use the same (or similar) gripper resources as text selection when implementing custom object manipulation in a Windows app. This helps provide a consistent interaction experience across the platform.

For example, grippers can also be used in image processing apps that support resizing and cropping or media player apps that provide adjustable progress bars, as shown in the following images.



Media player with adjustable progress bar.

Image editor with cropping grippers.

Related articles

For developers

- Custom user interactions

Samples

- Basic input sample
- Low latency input sample

- User interaction mode sample
- Focus visuals sample

Archive samples

- Input: XAML user input events sample
- Input: Device capabilities sample
- Input: Touch hit testing sample
- XAML scrolling, panning, and zooming sample
- Input: Simplified ink sample
- Input: Manipulations and gestures sample
- DirectX touch input sample

Input Method Editors (IME)

Article • 06/24/2021

An Input Method Editor (IME) is a software component that enables a user to input text in a language that can't be represented easily on a standard QWERTY keyboard. This is typically due to the number of characters in the user's written language, such as the various East Asian languages.

Instead of each single character appearing on a single keyboard key, a user types combinations of keys that are interpreted by the IME. The IME generates either the character that matches the set of key strokes or a list of candidate characters to choose from. The selected character is then inserted into the edit control that the user is interacting with.

> **Note**
>
> IMEs can support both hardware keyboards and on-screen or touch keyboards.

Your app doesn't need to interact directly with the IME. The IME is built into the system, just as the touch keyboard is. If your app has text input, and you intend to support text input in languages that require an IME, you should test the end-to-end customer experience for text entry. This lets you fix any issues, such as adjusting your UI so it isn't occluded by the touch keyboard or IME candidate window.

Creating an IME

To enable a great input experience for all users, Microsoft produces IMEs that ship in-box for a variety of languages.

In addition to the in-box IMEs, you can build your own custom IMEs that users can install and use just like an in-box IME.

All IMEs run in the Windows system, which is hardened to stop malicious IMEs and to improve the security and user experience of all IMEs.

Custom IMEs can link to the default touch keyboard and use its layout so that end users can use their IME with the touch keyboard. However, you cannot provide your own independent touch keyboard and certain functions of in-box IMEs for touch keyboards are not available to custom IMEs.

Requirements for IMEs

A third-party IME must meet these requirements:

- Must be digitally signed
- Must be Text Services Framework (TSF) aware, with appropriate IME flags set correctly
- Must follow the guidelines described in Input Method Editor (IME) requirements and Design and code Windows apps

A third-party IME that doesn't meet these requirements is blocked from running.

> **Note**
>
> Legacy custom IMEs can run in desktop apps, but are blocked in Windows apps.

Also, Windows Defender removes malicious IMEs from the system. Because of this, it's important that you familiarize yourself with the IME coding requirements. For more info, see Input Method Editor (IME) requirements.

Design guidelines for IMEs

Read the Input Method Editor (IME) requirements for more details on best practices and design guidelines for IMEs. In general, all IME UIs need to:

- Follow the UX guidelines for Windows Runtime apps
- Avoid modal experiences and only show the IME window when needed
- include icons that are black and white only

Related topics

- Input Method Editor (IME) requirements
- ITfFnGetPreferredTouchKeyboardLayout
- ITfCompartmentEventSink
- ITfThreadMgrEx::GetActiveFlags
- ITfContextView::GetWnd
- TF_INPUTPROCESSORPROFILE
- SendInput

Custom Input Method Editor (IME) requirements

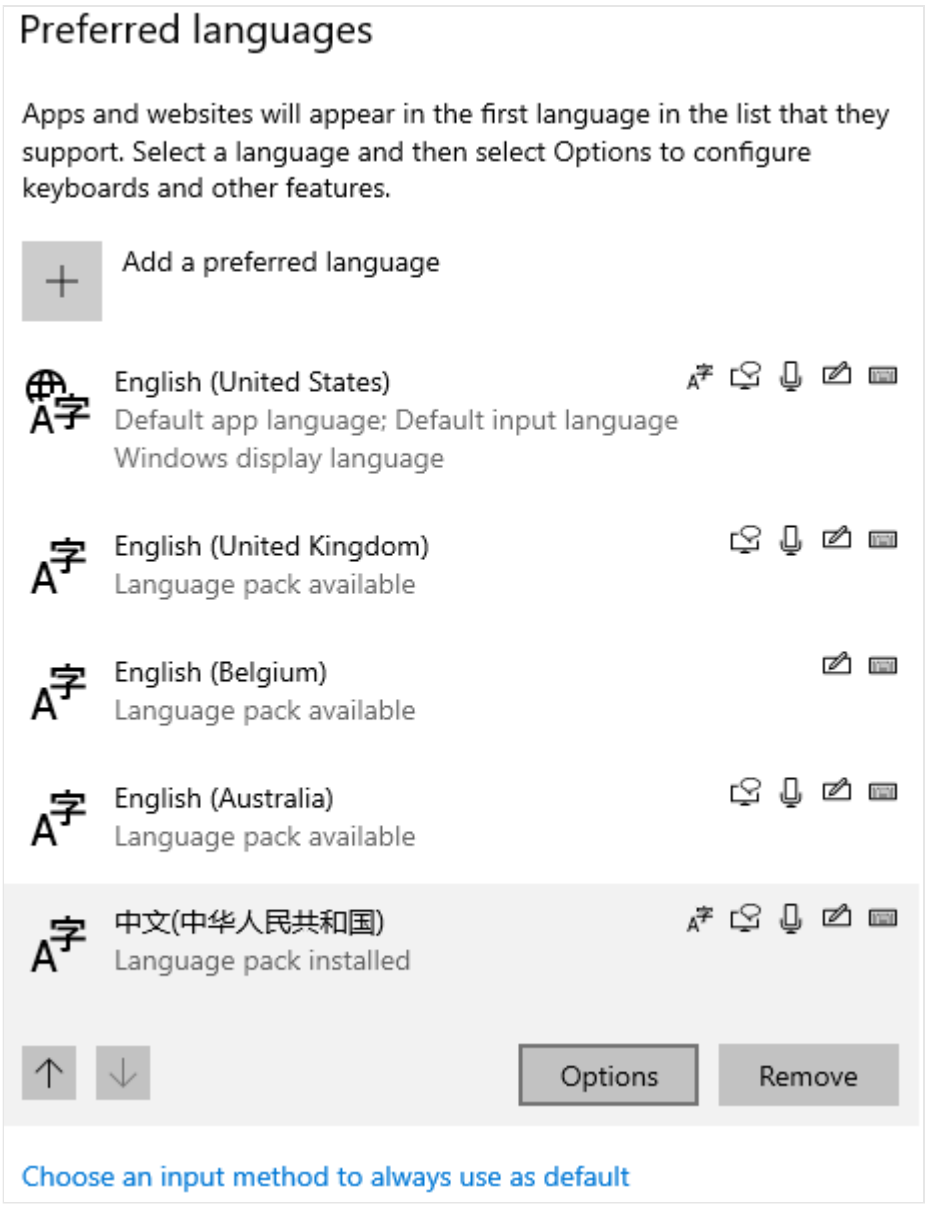
Article • 06/24/2021

These guidelines and requirements can help you to develop a custom Input Method Editor (IME) to help a user input text in a language that can't be represented easily on a standard QWERTY keyboard.

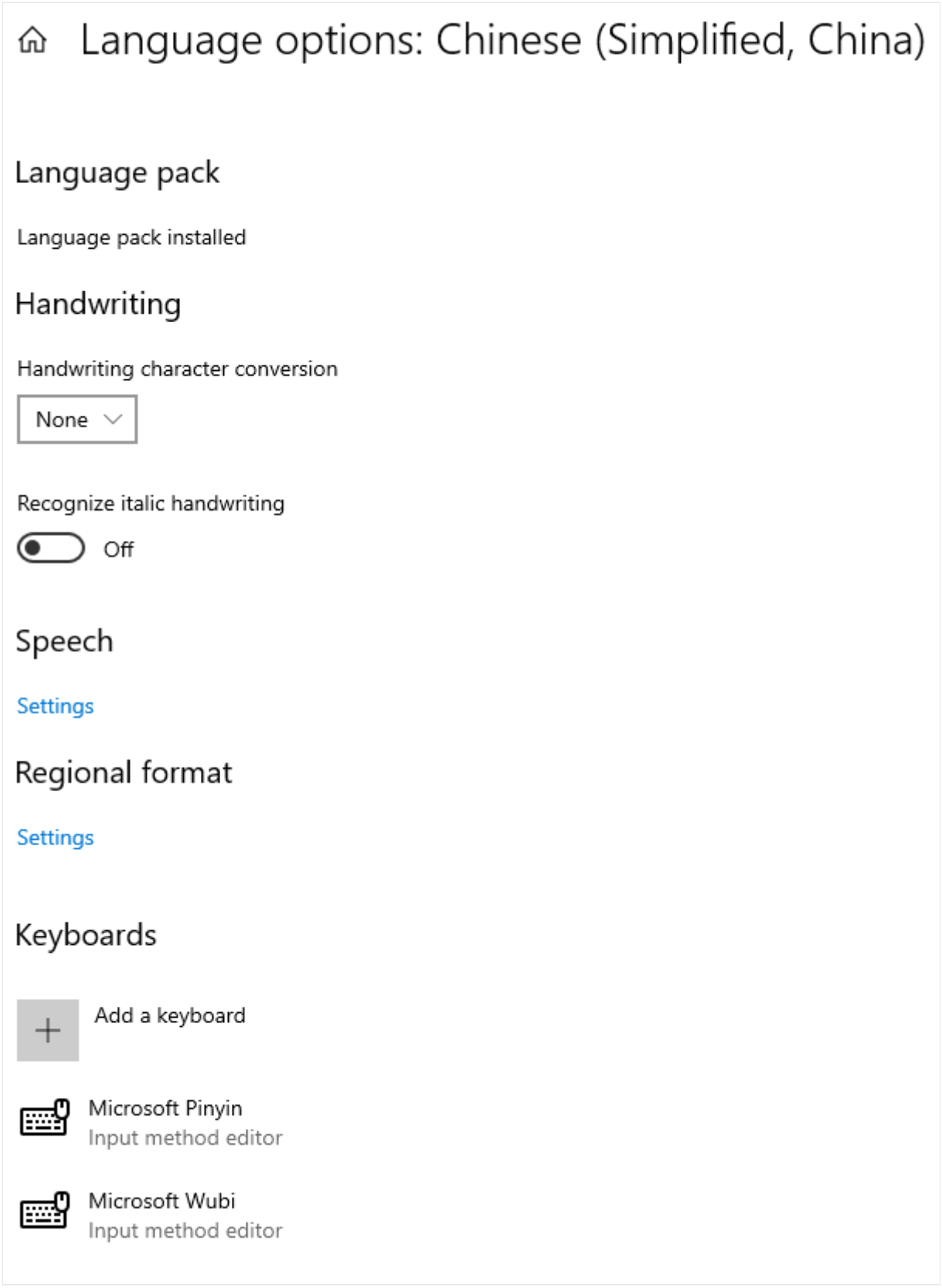
For an overview of IMEs, see Input Method Editor (IME).

Default IME

A user can select any of their active IMEs (**Settings -> Time & Language -> Language -> Preferred languages -> Language pack - Options**) to be the default IME for their preferred language.



Select the default keyboard on the Language options settings screen for the preferred language.



ⓘ Important

We do not recommend writing directly to the registry to set the default keyboard for your custom IME.

Compatibility requirements

The following are the basic compatibility requirements for a custom IME.

IME must be compatible with Windows apps

Use the Text Services Framework (TSF) to implement IMEs. Previously, you had the option of using the Input Method Manager (IMM32) for input services. Now the system blocks IMEs that are implemented by using Input Method Manager (IMM32).

When an app starts, TSF loads the IME DLL for the IME that's currently selected by the user. When an IME is loaded, it's subject to the same app container restrictions as the app. For example, an IME can't access the Internet if an app hasn't requested Internet access in its manifest. This behavior ensures that IMEs can't violate security contracts.

TSF is the intermediary between the app and your IME. TSF communicates input events to the IME and receives input characters back from the IME after the user has selected a character.

This behavior is the same as previous versions of Windows, but being loaded into a Windows app affects the potential capabilities of an IME.

If your IME needs to provide different functionality or UI between Windows apps and desktop apps, ensure that the DLL that's loaded by TSF checks which type of app it's being loaded into. Call the ITfThreadMgrEx::GetActiveFlags method in your IME and check the TF_TMF_IMMERSIVEMODE flag, so your IME triggers different application logic depending on the result.

Windows apps do not support Table Text Service (TTS) IMEs.

> **Note**
>
> Some tools for generating TTS IMEs produce IMEs that are marked as malware by Windows.

IME must be compatible with the system tray

There is no language bar to host IME icons. Instead, an Input Indicator shows on the system tray that indicates the current input option. The Input Indicator shows only the IME branding icon to indicate the currently running IME. Also, there's one IME mode icon that shows on the left of the IME branding icon for users to perform the most commonly used IME mode switch, like turning the IME on or off.

The Input Indicator shows the IME branding icon and mode icon only for compatible IMEs. IMEs that aren't compatible don't have the branding icon and mode icon displayed in the system tray. Instead, the Input Indicator shows the language abbreviation instead of the IME branding icon.

Store the IME icons in a DLL or EXE file, instead of a standalone .ico file. The design of IME icons must follow the guidelines described in the following UI design guidelines section.

IME branding icon

The Input Indicator gets the IME branding icon from the IME DLL by using the resource ID defined by the IME when it was registered on the system.

IME mode icon

Some IMEs may need to rely on the Input Indicator showing on the system tray to display the IME mode icon. In this case, the IME passes the IME mode icon to the Input Indicator by using GUID_LBI_INPUTMODE.

When passing the IME mode icons to the Input Indicator on the system tray, the default size of the IME mode icon is 16x16 pixels. The UI scaling follows DPI.

When passing the IME mode icon to the Input Indicator on UAC (User Account Control in Secure Desktop), the default size of the IME mode icon is 20x20 pixels. The UI scaling for IME mode icon on UAC follows PPI.

IME must work in app container

Some IME functions are affected in an app container.

- **Dictionary Files** - Frequently, IMEs have read-only dictionary files to map user input to specific characters. To access these files from inside an app container, your IME must place them under the Program Files or Windows directories. By default, these directories can be read from an app container, so IMEs can access dictionary files that are stored in these locations. If your IME must store the dictionary file somewhere else, it must explicitly manipulate the Access Control Lists (ACL) of the dictionary files to allow access from app containers.
- **Internet Updating** - If your IME needs to update its dictionaries using data from the Internet, it can't reliably do so inside an app container, because Internet access

isn't always permitted. Instead, your IME should run a separate desktop process that's responsible for updating the dictionary files with data from the Internet.

- **On-the-fly learning** - If an IME is running in an app container that has Internet access, there's no restriction on the endpoints that the IME can communicate with. In this case, an IME can use a cloud server to provide on-the-fly learning services. Some IMEs download and upload user input on the fly, while the user is typing. Since Internet access is not guaranteed in an app container, this may not always be allowed.
- **Sharing information between processes** - IMEs may need to share data about the user's input preferences between apps that are in different app containers. Use a web service to share data between apps.

> **Important**
>
> If you try to circumvent app container security rules, your IME may be treated as malware and blocked.

IME and touch keyboard

Your IME must ensure that its candidate pane's UI, and other UI elements, aren't drawn underneath the touch keyboard. The touch keyboard is displayed in a higher z-order band than all apps, and the IME UI is displayed in the same z-order band as the app it's active in. As a result, the touch keyboard can overlap and hide the IME UI. In most cases, the app should resize its window to account for the touch keyboard. If an app doesn't resize, the IME still can use the InputPane API to get the position of the touch keyboard. The IME queries the Location property, or it registers a handler for the touch keyboard's Show and Hide events. The Show event is raised every time the user taps in an edit field, even if the touch keyboard is displayed currently. Your IME can use this API to get the screen space used by the touch keyboard before the IME draws candidate (or other) UI, and to reflow the IMEs UI to avoid drawing beneath the touch keyboard.

Specifying the preferred touch keyboard layout

The IME can specify which touch keyboard layout to use, and the IME is enabled to work with touch-optimized layouts. This functionality is limited to IMEs for the Korean, Japanese, Chinese Simplified, and Chinese Traditional input languages.

There are seven layouts supported by the touch keyboard, three of which are classic layouts and four of which are touch-optimized layouts. The classic layouts look and behave like a physical keyboard.

All of the three classic layouts are for inputting traditional Chinese in different forms:

- Phonetic-based input
- Changjie input
- Dayi input

In addition to the classic layouts, there is one touch-optimized layout for each of the Korean, Japanese, Simplified Chinese, and Traditional Chinese input languages.

To use this functionality, your IME must implement the ITfFnGetPreferredTouchKeyboardLayout interface, which is exported by the IME by using the Text Services Framework ITfFunctionProvider API.

If your IME doesn't support the ITfFnGetPreferredTouchKeyboardLayout interface, using the IME results in the default classic layout for the language that is displayed by the touch keyboard.

If your IME needs to set one of the classic layouts as the preferred layout, no additional work is required on the IME side beyond supporting the ITfFnGetPreferredTouchKeyboardLayout and ITfFunctionProvider interfaces. But additional work is required in the IME to work with the touch-optimized layouts, and this is described in the next section.

Touch-optimized layout

The touch-optimized keyboards for the Korean, Japanese, Simplified Chinese, and Traditional Chinese input languages display a different layout for IME On and IME Off conversion modes. There's a key on the touch keyboard to set the IME conversion mode to On or Off, but the IME mode of the keyboard also may change as focus changes among edit controls.

The touch-optimized keyboards for the Japanese, Simplified Chinese, and Traditional Chinese input languages contain a key, or keys, which the IME uses to navigate through candidate pages. For Japanese and Simplified Chinese, the candidate page key displays on the touch-optimized layout. For Traditional Chinese, there are separate keys for the previous and next candidate pages.

When these keys are pressed, the touch keyboard calls the SendInput function to send the following Unicode Private Use Area characters to the focused application, which the IME can intercept and act on:

- **Next page (0xF003)** - Sent when the candidate page key is pressed on the touch-optimized keyboard for Japanese and Simplified Chinese, or when the next page

key is pressed on the touch-optimized keyboard for Traditional Chinese.

- **Previous page (0xF004)** - Sent when either the candidate page key is pressed at the same time as the Shift key on the touch-optimized keyboard for Japanese and Simplified Chinese, or when the previous page key is pressed on the touch-optimized keyboard for Traditional Chinese.

These characters are sent as Unicode input. The next paragraph details how to extract the character information during the key event sink notifications that the Text Services Framework IME will receive. These character values are not defined in any header file, so you will need to define them in your code.

To intercept keyboard input, your IME must register as a key event sink. For Unicode input that is generated by using the SendInput function, the WPARAM parameter of the ITfKeyEventSink callbacks (OnKeyDown, OnKeyUp, OnTestKeyDown, OnTestKeyUp) always contains the virtual key VK_PACKET and doesn't identify the character directly.

Implement the following call sequence to access the character:

```
C++

// Keyboard state
BYTE abKbdState[256];
if (!GetKeyboardState(abKbdState))
{
   return 0;
}

// Map virtual key to character code
WCHAR wch;
if (ToUnicode(VK_PACKET, 0, abKbdState, &wch, 1, 0) == 1)
{
   return wch;
}
```

IME search integration

Provide users with search features through the search contract and integration with the search pane.

Search pane and IME suggestions

The search pane is a central location for users to perform searches across all of their apps. For IME users, Windows provides a unique search experience that lets compatible IMEs integrate with Windows for greater efficiency and usability.

Users who type with an IME that's compatible with search get two main benefits:

- Seamless interaction between the IME and the search experience. IME candidates are shown inline under the search box without occluding search suggestions. The user can use the keyboard to navigate seamlessly between the search box, the IME conversion candidates, and the search suggestions.
- Faster access to relevant results and suggestions provided by applications. The app has access to all current conversion candidates to provide more relevant suggestions. To better prioritize search suggestions, conversions are given to apps in order of relevance. Users find and select the result they want without converting, just by typing in phonetic.

An IME is compatible with the integrated search experience if it meets the following criteria:

- Compatible with the Windows style shell.
- Implement the TSF UILess mode APIs. For more info, see UILess Mode Overview.
- Implement the TSF search integration APIs, ITfFnSearchCandidateProvider and ITfIntegratableCandidateListUIElement.

When activated in the search pane, a compatible IME is placed in UIless mode and can't show its UI. Instead, it sends conversion candidates to Windows, which displays them in the inline candidate list control, as shown in the previous screenshot.

Also, the IME sends candidates that should be used to run the current search. These candidates could be the same as the conversion candidates, or they could be tailored for search.

Good search candidates meet the following criteria:

- No prefix overlap. For example, 北京大学 and北京 are redundant because one is a prefix of the other.
- No redundant candidates. Any redundant candidate isn't useful for search because it doesn't help filter results. For example, any result that matches 北京大学 also matches 北京.
- No prediction candidate, only conversion. For example, if the user types "be", the IME can return 北 as a candidate, but not 北京大学. Usually, prediction candidates are too restrictive.

IMEs that don't meet the criteria aren't compatible with search display in the same way as other controls, and can't take advantage of UI integration and search candidates. Apps receive queries only after the user has finished composing.

When an app that supports the search contract receives a query, the query event contains a "queryTextAlternatives" array that contains all known alternatives, ranked from the most relevant (likely) to least relevant (unlikely).

When alternatives are provided, the app should treat each alternative as a query and return all results that match any of the alternatives. The app should behave as if the user had issued multiple queries at the same time, essentially issuing an "or" query to the service providing the results. For performance considerations, apps often limit matching to between 5 and 20 of the most relevant alternatives.

UI design guidelines

All IMEs must follow the user experience guidelines described in Design and code Windows apps.

Don't use sticky windows

Your IME windows should appear only when needed, and they shouldn't be visible all the time. When users don't need to type, IME windows shouldn't show. The IME window shouldn't be a full screen window. IME windows shouldn't overlap each other. The windows should be designed in a Windows style and follow UI scaling.

IME icons

There are two kinds of IME icons, branding icons and mode icons. All IME icons must be designed with black and white colors only. The new IME icons borrow from the glyphic look of the system tray icons. This style has been created so all languages can use it to complement the familial look while also differentiating from each other.

The file format for IME icons is ICO. You must provide the following icon sizes.

- 16x16 pixels
- 20x20 pixels
- 24x24 pixels
- 32x32 pixels
- 40x40 pixels
- 48x48 pixels

Ensure that 32-bit icons with alpha channel are provided in all resolutions.

IME brand icons are defined by a white box in which a typographic glyph rendered in a modern typeface is placed. Each defining glyph is chosen by each language team. The glyph is black. The box includes an outer stroke of 1 pixel in black at 50% opacity. "New" versions are defined by a rounded corner in the upper left of the box.

IME mode icons are defined by a white typographic glyph in a modern typeface which includes an outer stroke of 1 pixel in black at 50% opacity.

Icon	Description
	Example IME brand icon for Traditional Chinese ChangeJie.
	Example IME brand icon for Traditional Chinese ChangeJie.

Icon	Description
	Example IME mode icon.

Owned window

To display candidate UI, an IME must set its window to be owned-window, so it can display over the currently running app. Use the ITfContextView::GetWnd method to retrieve the window to own to. If GetWnd returns an error or a NULLHWND, call the GetFocus function.

```
if (FAILED(pView->GetWnd(&parentWndHandle)) || (parentWndHandle == nullptr)) { parentWndHandle = GetFocus(); }
```

IME candidate window interaction with light dismiss surfaces

The dismissal model for popup windows is called "light dismiss" because it's easy for a user to close such windows. For IMEs to function well in the Windows interaction model, the IME windows must participate in the light dismiss model.

In order to participate in the light dismiss model, your IME must raise three new Windows events by using the NotifyWinEvent function or a similar function. These new events are:

- **EVENT_OBJECT_IME_SHOW** - Raise this event when the IME becomes visible.
- **EVENT_OBJECT_IME_HIDE** - Raise this event when the IME is hidden.
- **EVENT_OBJECT_IME_CHANGE** - Raise this event when the IME moves or changes size.

Declaring compatibility

IMEs declare that they are compatible by registering the category GUID_TFCAT_TIPCAP_IMMERSIVESUPPORT for their IME using ITfCategoryMgr::RegisterCategory.

Set the default IME mode to on

We provide a better UX for IMEs.

DPI scaling support for desktop applications

Enhanced DPI scaling support enables querying the declared DPI awareness level of each desktop process to determine if it needs to scale the UI. In a multi-monitor scenario, Windows scales the UI appropriately for different DPI settings on each monitor.

Because your IME runs in the context of each application's process, you shouldn't declare a DPI awareness level for your IME. This ensures that your IME runs at the DPI awareness level of the current process.

To ensure that all IME UI elements have scaling parity with the UI elements of the process in which you are running, you must respond appropriately to different DPI values.

> **Note**
>
> To ensure parity with new desktop applications, your IME should support per monitor–DPI awareness, but shouldn't declare a level of awareness itself. The system determines the appropriate scaling requirements in each scenario.

For details about DPI scaling support requirements for Desktop applications, see High DPI.

IME installation

If you build your IME by using Microsoft Visual Studio, create an installation experience for your IME by using a third-party installer, like InstallShield from Flexera Software.

The following steps show how to use InstallShield to create a setup project for your IME DLL.

- Install Visual Studio.
- Start Visual Studio.
- On the **File** menu, point to **New** and select **Project**. The **New Project** dialog opens.
- In the left pane, navigate to **Templates** > **Other Project Types** > **Setup and Deployment**, click **Enable InstallShield Limited Edition**, and click **OK**. Follow the installation instructions.
- Restart Visual Studio.
- Open the IME solution (.sln) file.

- In Solution Explorer, right-click the solution, point to **Add**, and select **New Project**. The **Add New Project** dialog opens.
- In the left tree view control, navigate to **Templates** > **Other Project Types** > **InstallShield Limited Edition**.
- In the center window, click **InstallShield Limited Edition Project**.
- In the **Name** text box, type "SetupIME" and click **OK**.
- In the **Project Assistant** dialog, click **Application Information**.
- Fill in your company name and the other fields.
- Click **Application Files**.
- In the left pane, right-click the **[INSTALLDIR]** folder, and select **New Folder**. Name the folder "Plugins".
- Click **Add Files**. Navigate to your IME DLL and add it to the **Plugins** folder. Repeat this step for the IME dictionary.
- Right-click the IME DLL and select **Properties**. The **Properties** dialog opens.
- In the **Properties** dialog, click the **COM & .NET Settings** tab.
- Under **Registration Type**, select **Self-registration** and click **OK**.
- Build the solution. The IME DLL is built, and InstallShield creates a setup.exe file that enables users to install your IME on Windows.

To create your own installation experience, call the ITfInputProcessorProfileMgr::RegisterProfile method to register the IME during installation. Don't write registry entries directly.

If the IME must be usable immediately after installation, call InstallLayoutOrTip to add the IME to user-enabled input methods, using the following format for the psz parameter:

```
<LangID 1>:{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}
```

IME accessibility

Implement the following convention to make your IMEs conform to the accessibility requirements and to work with Narrator. To make candidate lists accessible, your IMEs must follow this convention.

- The candidate list must have a **UIA_AutomationIdPropertyId** equal to "IME_Candidate_Window" for lists of conversion candidates or "IME_Prediction_Window" for lists of prediction candidates.
- When the candidate list appears and disappears, it raises events of type **UIA_MenuOpenedEventId** and **UIA_MenuClosedEventId**, respectively

- When the current selected candidate changes, the candidate list raises a **UIA_SelectionItem_ElementSelectedEventId**. The selected element should have a property **UIA_SelectionItemIsSelectedPropertyId** equal to **TRUE**.
- The **UIA_NamePropertyId** for each item in the candidate list must be the name of the candidate. Optionally, you can provide additional information to disambiguate candidates through **UIA_HelpTextPropertyId**.

Related topics

- Input Method Editor (IME)
- ITfFnGetPreferredTouchKeyboardLayout
- ITfCompartmentEventSink
- ITfThreadMgrEx::GetActiveFlags
- ITfContextView::GetWnd
- TF_INPUTPROCESSORPROFILE
- ITfFnSearchCandidateProvider
- ITfIntegratableCandidateListUIElement
- SendInput
- Accessibility

Gamepad and remote control interactions

Article • 05/13/2022

Many interaction experiences are shared between gamepad, remote control, and keyboard

Build interaction experiences in your Windows applications that ensure your app is usable and accessible through both the traditional input types of PCs, laptops, and tablets (mouse, keyboard, touch, and so on), as well as the input types typical of the TV and Xbox *10-foot* experience, such as the gamepad and remote control.

See Designing for Xbox and TV for general design guidance on Windows applications in the *10-foot* experience.

Overview

In this topic, we discuss what you should consider in your interaction design (or what you don't, if the platform looks after it for you), and provide guidance, recommendations, and suggestions for building Windows applications that are enjoyable to use regardless of device, input type, or user abilities and preferences.

Bottom line, your application should be as intuitive and easy to use in the *2-foot* environment as it is in the *10-foot* environment (and vice versa). Support the user's preferred devices, make the UI focus clear and unmistakable, arrange content so navigation is consistent and predictable, and give users the shortest path possible to what they want to do.

> **Note**
>
> Most of the code snippets in this topic are in XAML/C#; however, the principles and concepts apply to all Windows apps. If you're developing an HTML/JavaScript Windows app for Xbox, check out the excellent TVHelpers library on GitHub.

Optimize for both 2-foot and 10-foot experiences

At a minimum, we recommend that you test your applications to ensure they work well in both 2-foot and 10-foot scenarios, and that all functionality is discoverable and accessible to the Xbox gamepad and remote-control.

Here are some other ways you can optimize your app for use in both 2-foot and 10-foot experiences and with all input devices (each links to the appropriate section in this topic).

> **Note**
>
> Because Xbox gamepads and remote controls support many Windows keyboard behaviors and experiences, these recommendations are appropriate for both input types. See Keyboard interactions for more detailed keyboard info.

Expand table

Feature	Description
XY focus navigation and interaction	**XY focus navigation** enables the user to navigate around your app's UI. However, this limits the user to navigating up, down, left, and right. Recommendations for dealing with this and other considerations are outlined in this section.
Mouse mode	XY focus navigation isn't practical, or even possible, for some types of applications, such as maps or drawing and painting apps. In these cases,

Feature	Description
	mouse mode lets users navigate freely with a gamepad or remote control, just like a mouse on a PC.
Focus visual	The focus visual is a border that highlights the currently focused UI element. This helps the user quickly identify the UI they are navigating through or interacting with.
Focus engagement	Focus engagement requires the user to press the **A/Select** button on a gamepad or remote control when a UI element has focus in order to interact with it.
Hardware buttons	The gamepad and remote control provide very different buttons and configurations.

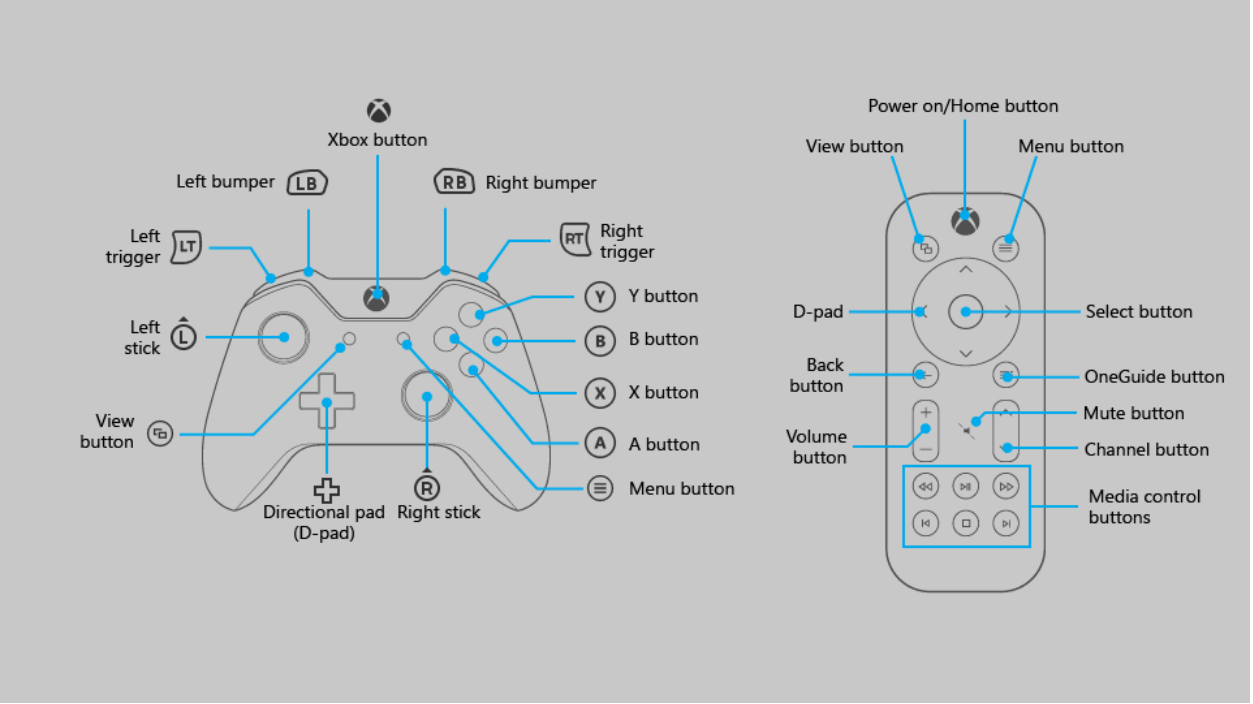
Gamepad and remote control

Just like keyboard and mouse are for PC, and touch is for phone and tablet, gamepad and remote control are the main input devices for the 10-foot experience. This section introduces what the hardware buttons are and what they do. In XY focus navigation and interaction and Mouse mode, you will learn how to optimize your app when using these input devices.

The quality of gamepad and remote behavior that you get out-of-the-box depends on how well keyboard is supported in your app. A good way to ensure that your app will work well with gamepad/remote is to make sure that it works well with keyboard on PC, and then test with gamepad/remote to find weak spots in your UI.

Hardware buttons

Throughout this document, buttons will be referred to by the names given in the following diagram.



As you can see from the diagram, there are some buttons that are supported on gamepad that are not supported on remote control, and vice versa. While you can use buttons that are only supported on one input device to make navigating the UI faster, be aware that using them for critical interactions may create a situation where the user is unable to interact with certain parts of the UI.

The following table lists all of the hardware buttons supported by Windows apps, and which input device supports them.

Expand table

Button	Gamepad	Remote control
A/Select button	Yes	Yes
B/Back button	Yes	Yes
Directional pad (D-pad)	Yes	Yes
Menu button	Yes	Yes
View button	Yes	Yes
X and Y buttons	Yes	No
Left stick	Yes	No
Right stick	Yes	No
Left and right triggers	Yes	No

Button	Gamepad	Remote control
Left and right bumpers	Yes	No
OneGuide button	No	Yes
Volume button	No	Yes
Channel button	No	Yes
Media control buttons	No	Yes
Mute button	No	Yes

Built-in button support

UWP automatically maps existing keyboard input behavior to gamepad and remote control input. The following table lists these built-in mappings.

Expand table

Keyboard	Gamepad/remote
Arrow keys	D-pad (also left stick on gamepad)
Spacebar	A/Select button
Enter	A/Select button
Escape	B/Back button*

*When neither the KeyDown nor KeyUp events for the B button are handled by the app, the SystemNavigationManager.BackRequested event will be fired, which should result in back navigation within the app. However, you have to implement this yourself, as in the following code snippet:

C#

```
// This code goes in the MainPage class

public MainPage()
{
    this.InitializeComponent();

    // Handling Page Back navigation behaviors
    SystemNavigationManager.GetForCurrentView().BackRequested +=
        SystemNavigationManager_BackRequested;
}
```

```
private void SystemNavigationManager_BackRequested(
    object sender,
    BackRequestedEventArgs e)
{
    if (!e.Handled)
    {
        e.Handled = this.BackRequested();
    }
}

public Frame AppFrame { get { return this.Frame; } }

private bool BackRequested()
{
    // Get a hold of the current frame so that we can inspect the app back
stack
    if (this.AppFrame == null)
        return false;

    // Check to see if this is the top-most page on the app back stack
    if (this.AppFrame.CanGoBack)
    {
        // If not, set the event to handled and go back to the previous page
in the
        // app.
        this.AppFrame.GoBack();
        return true;
    }
    return false;
}
```

> ⓘ **Note**
>
> If the B button is used to go back, then don't show a back button in the UI. If you're using a Navigation view, the back button will be hidden automatically. For more information about backwards navigation, see Navigation history and backwards navigation for Windows apps.

Windows apps on Xbox One also support pressing the **Menu** button to open context menus. For more information, see CommandBar and ContextFlyout.

Accelerator support

Accelerator buttons are buttons that can be used to speed up navigation through a UI. However, these buttons may be unique to a certain input device, so keep in mind that not all users will be able to use these functions. In fact, gamepad is currently the only input device that supports accelerator functions for Windows apps on Xbox One.

The following table lists the accelerator support built into the UWP, as well as that which you can implement on your own. Utilize these behaviors in your custom UI to provide a consistent and friendly user experience.

Expand table

Interaction	Keyboard/Mouse	Gamepad	Built-in for:	Recommended for:
Page up/down	Page up/down	Left/right triggers	CalendarView, ListBox, ListViewBase, ListView, `ScrollViewer`, Selector, LoopingSelector, ComboBox, FlipView	Views that support vertical scrolling
Page left/right	None	Left/right bumpers	ListBox, ListViewBase, ListView, `ScrollViewer`, Selector, LoopingSelector, FlipView	Views that support horizontal scrolling
Zoom in/out	Ctrl +/-	Left/right triggers	None	`ScrollViewer`, views that support zooming in and out
Open/close nav pane	None	View	None	Navigation panes
Search	None	Y button	None	Shortcut to the main search function in the app
Open context menu	Right-click	Menu button	ContextFlyout	Context menus

XY focus navigation and interaction

If your app supports proper focus navigation for keyboard, this will translate well to gamepad and remote control. Navigation with the arrow keys is mapped to the **D-pad** (as well as the **left stick** on gamepad), and interaction with UI elements is mapped to the **Enter/Select** key (see Gamepad and remote control).

Many events and properties are used by both keyboard and gamepad—they both fire `KeyDown` and `KeyUp` events, and they both will only navigate to controls that have the properties `IsTabStop="True"` and `Visibility="Visible"`. For keyboard design guidance, see Keyboard interactions.

If keyboard support is implemented properly, your app will work reasonably well; however, there may be some extra work required to support every scenario. Think about your app's specific needs to provide the best user experience possible.

> **ⓘ Important**
>
> Mouse mode is enabled by default for Windows apps running on Xbox One. To disable mouse mode and enable XY focus navigation, set `Application.RequiresPointerMode=WhenRequested`.

Debugging focus issues

The FocusManager.GetFocusedElement method will tell you which element currently has focus. This is useful for situations where the location of the focus visual may not be obvious. You can log this information to the Visual Studio output window like so:

C#

```
page.GotFocus += (object sender, RoutedEventArgs e) =>
{
    FrameworkElement focus = FocusManager.GetFocusedElement() as
FrameworkElement;
    if (focus != null)
    {
        Debug.WriteLine("got focus: " + focus.Name + " (" +
            focus.GetType().ToString() + ")");
    }
};
```

There are three common reasons why XY navigation might not work the way you expect:

- The IsTabStop or Visibility property is set wrong.
- The control getting focus is actually bigger than you think—XY navigation looks at the total size of the control (ActualWidth and ActualHeight), not just the portion of the control that renders something interesting.
- One focusable control is on top of another—XY navigation doesn't support controls that are overlapped.

If XY navigation is still not working the way you expect after fixing these issues, you can manually point to the element that you want to get focus using the method described in Overriding the default navigation.

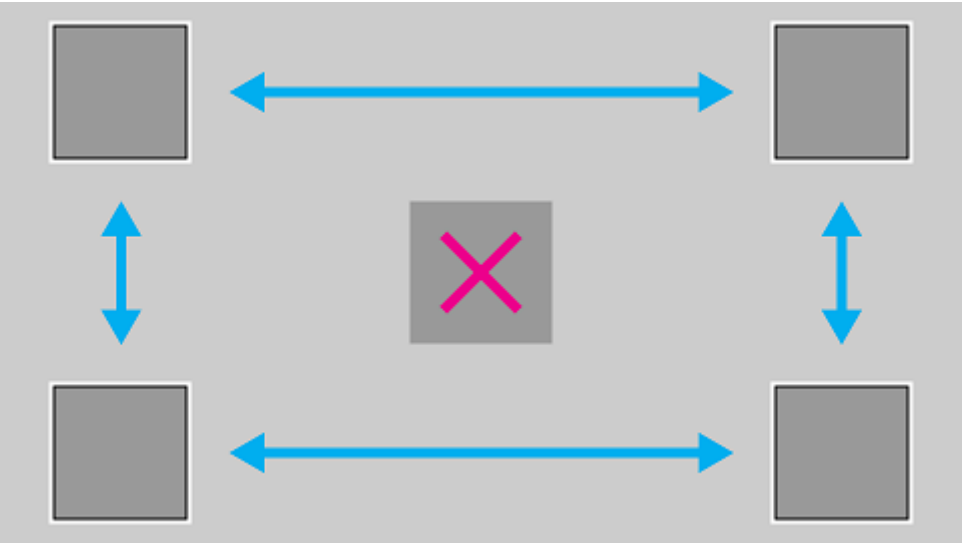
If XY navigation is working as intended but no focus visual is displayed, one of the following issues may be the cause:

- You re-templated the control and didn't include a focus visual. Set `UseSystemFocusVisuals="True"` or add a focus visual manually.
- You moved the focus by calling `Focus(FocusState.Pointer)`. The FocusState parameter controls what happens to the focus visual. Generally you should set this to `FocusState.Programmatic`, which keeps the focus visual visible if it was visible before, and hidden if it was hidden before.

The rest of this section goes into detail about common design challenges when using XY navigation, and offers several ways to solve them.

Inaccessible UI

Because XY focus navigation limits the user to moving up, down, left, and right, you may end up with scenarios where parts of the UI are inaccessible. The following diagram illustrates an example of the kind of UI layout that XY focus navigation doesn't support. Note that the element in the middle is not accessible by using gamepad/remote because the vertical and horizontal navigation will be prioritized and the middle element will never be high enough priority to get focus.



If for some reason rearranging the UI is not possible, use one of the techniques discussed in the next section to override the default focus behavior.

Overriding the default navigation

While the Universal Windows Platform tries to ensure that D-pad/left stick navigation makes sense to the user, it cannot guarantee behavior that is optimized for your app's intentions. The best way to ensure that navigation is optimized for your app is to test it with a gamepad and confirm that every UI element can be accessed by the user in a manner that makes sense for your app's scenarios. In case your app's scenarios call for a

behavior not achieved through the XY focus navigation provided, consider following the recommendations in the following sections and/or overriding the behavior to place the focus on a logical item.

The following code snippet shows how you might override the XY focus navigation behavior:

XML

```
<StackPanel>
    <Button x:Name="MyBtnLeft"
            Content="Search" />
    <Button x:Name="MyBtnRight"
            Content="Delete"/>
    <Button x:Name="MyBtnTop"
            Content="Update" />
    <Button x:Name="MyBtnDown"
            Content="Undo" />
    <Button Content="Home"
            XYFocusLeft="{x:Bind MyBtnLeft}"
            XYFocusRight="{x:Bind MyBtnRight}"
            XYFocusDown="{x:Bind MyBtnDown}"
            XYFocusUp="{x:Bind MyBtnTop}" />
</StackPanel>
```

In this case, when focus is on the `Home` button and the user navigates to the left, focus will move to the `MyBtnLeft` button; if the user navigates to the right, focus will move to the `MyBtnRight` button; and so on.

To prevent the focus from moving from a control in a certain direction, use the `XYFocus*` property to point it at the same control:

XML

```
<Button Name="HomeButton"
        Content="Home"
        XYFocusLeft ="{x:Bind HomeButton}" />
```

Using these `XYFocus` properties, a control parent can also force the navigation of its children when the next focus candidate is out of its visual tree, unless the child who has the focus uses the same `XYFocus` property.

XML

```
<StackPanel Orientation="Horizontal" Margin="300,300">
    <UserControl XYFocusRight="{x:Bind ButtonThree}">
        <StackPanel>
```

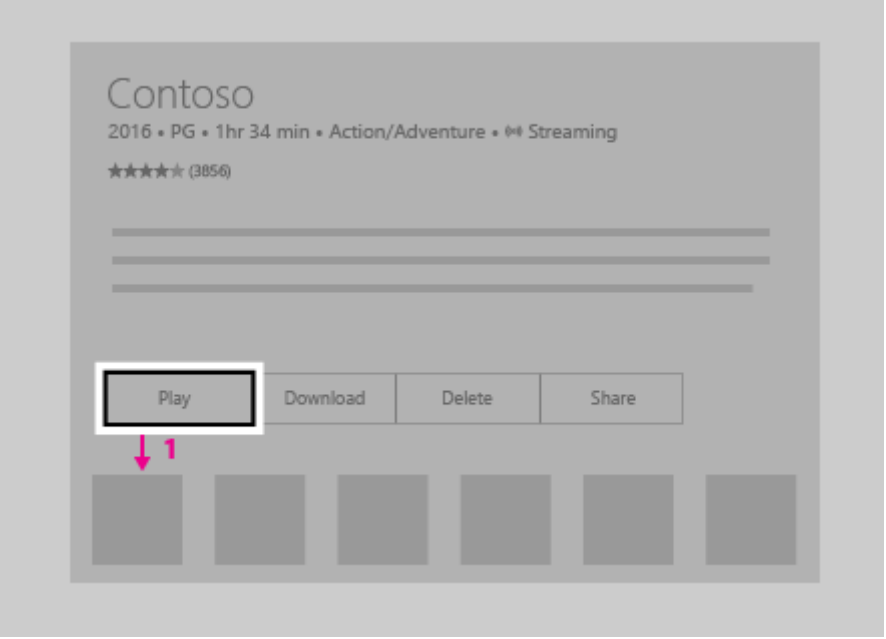
```
            <Button Content="One"/>
            <Button Content="Two"/>
        </StackPanel>
    </UserControl>
    <StackPanel>
        <Button x:Name="ButtonThree" Content="Three"/>
        <Button Content="Four"/>
    </StackPanel>
</StackPanel>
```

In the sample above, if the focus is on `Button` Two and the user navigates to the right, the best focus candidate is `Button` Four; however, the focus is moved to `Button` Three because the parent `UserControl` forces it to navigate there when it is out of its visual tree.

Path of least clicks

Try to allow the user to perform the most common tasks in the least number of clicks. In the following example, the TextBlock is placed between the **Play** button (which initially gets focus) and a commonly used element, so that an unnecessary element is placed in between priority tasks.

In the following example, the TextBlock is placed above the **Play** button instead. Simply rearranging the UI so that unnecessary elements are not placed in between priority tasks will greatly improve your app's usability.



CommandBar and ContextFlyout

When using a CommandBar, keep in mind the issue of scrolling through a list as mentioned in Problem: UI elements located after long scrolling list/grid. The following image shows a UI layout with the `CommandBar` on the bottom of a list/grid. The user would need to scroll all the way down through the list/grid to reach the `CommandBar`.

What if you put the `CommandBar` *above* the list/grid? While a user who scrolled down the list/grid would have to scroll back up to reach the `CommandBar`, it is slightly less navigation than the previous configuration. Note that this is assuming that your app's initial focus is placed next to or above the `CommandBar`; this approach won't work as well if the initial focus is below the list/grid. If these `CommandBar` items are global action items

that don't have to be accessed very often (such as a **Sync** button), it may be acceptable to have them above the list/grid.

While you can't stack a `CommandBar`'s items vertically, placing them against the scroll direction (for example, to the left or right of a vertically scrolling list, or the top or bottom of a horizontally scrolling list) is another option you may want to consider if it works well for your UI layout.

If your app has a `CommandBar` whose items need to be readily accessible by users, you may want to consider placing these items inside a ContextFlyout and removing them from the `CommandBar`. `ContextFlyout` is a property of UIElement and is the context menu associated with that element. On PC, when you right-click on an element with a `ContextFlyout`, that context menu will pop up. On Xbox One, this will happen when you press the **Menu** button while the focus is on such an element.

UI layout challenges

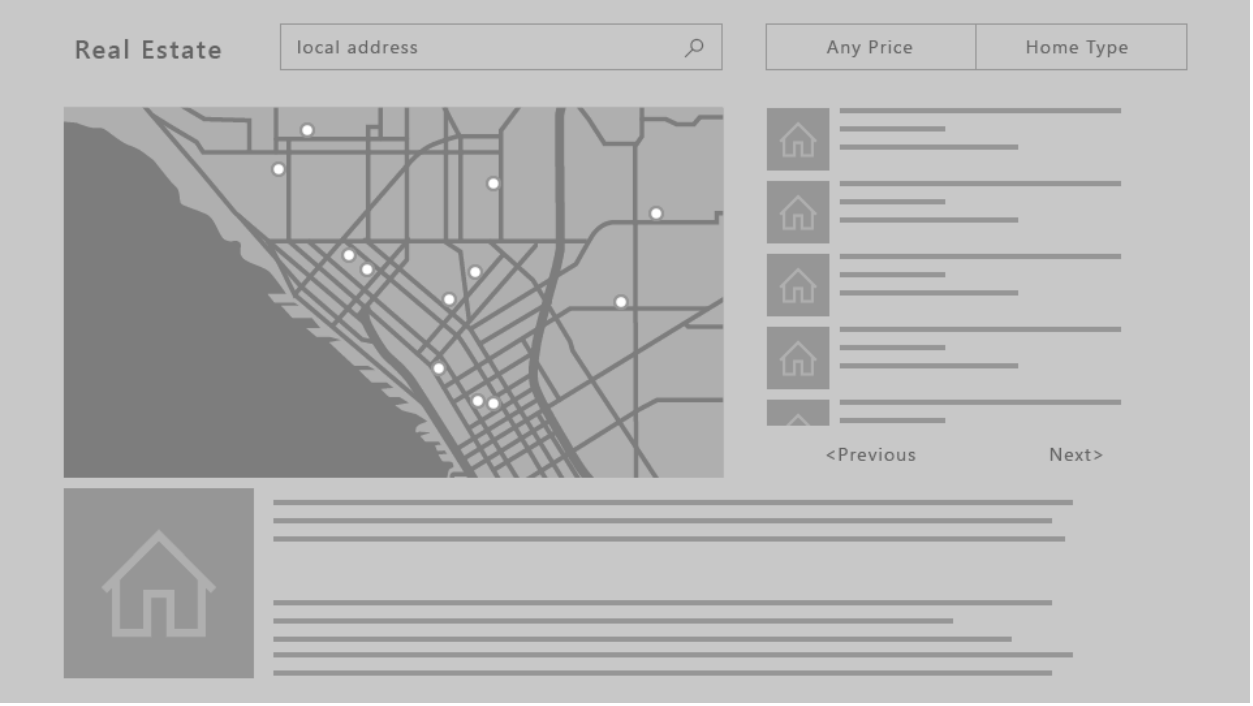
Some UI layouts are more challenging due to the nature of XY focus navigation, and should be evaluated on a case-by-case basis. While there is no single "right" way, and which solution you choose is up to your app's specific needs, there are some techniques that you can employ to make a great TV experience.

To understand this better, let's look at an imaginary app that illustrates some of these issues and techniques to overcome them.

> ⓘ **Note**
>
> This fake app is meant to illustrate UI problems and potential solutions to them, and is not intended to show the best user experience for your particular app.

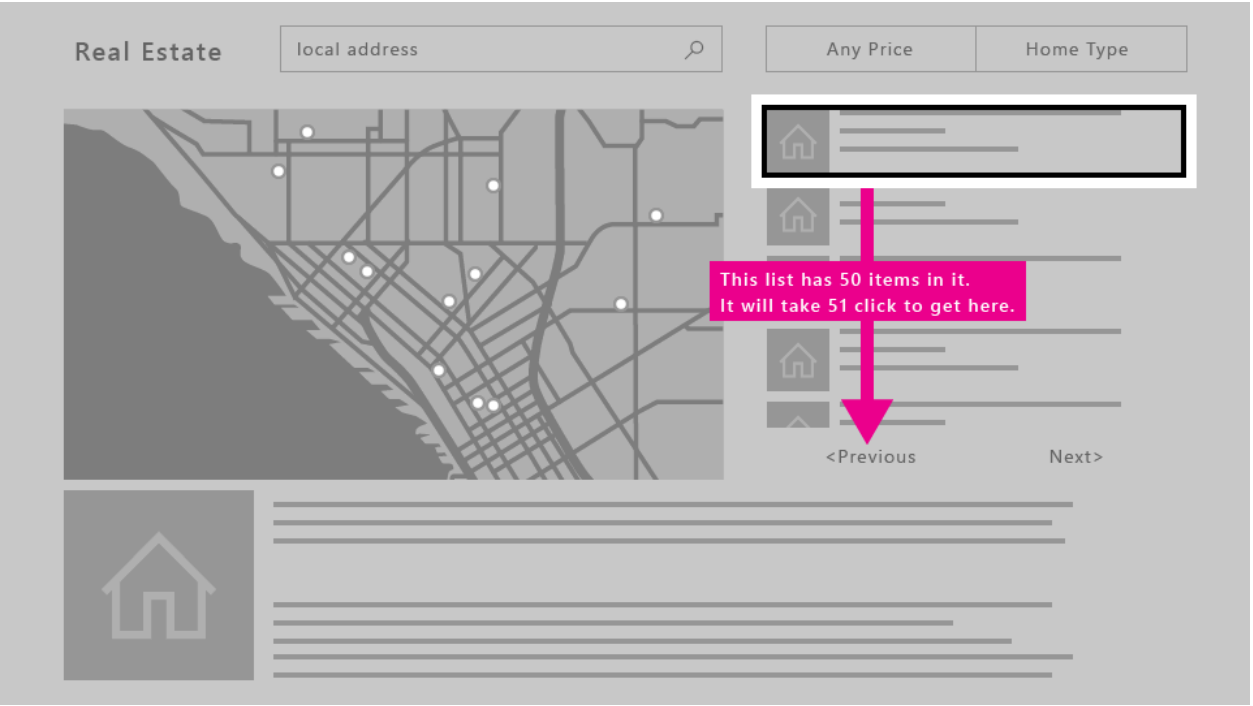
The following is an imaginary real estate app which shows a list of houses available for sale, a map, a description of a property, and other information. This app poses three challenges that you can overcome by using the following techniques:

- UI rearrange
- Focus engagement
- Mouse mode



Problem: UI elements located after long scrolling list/grid

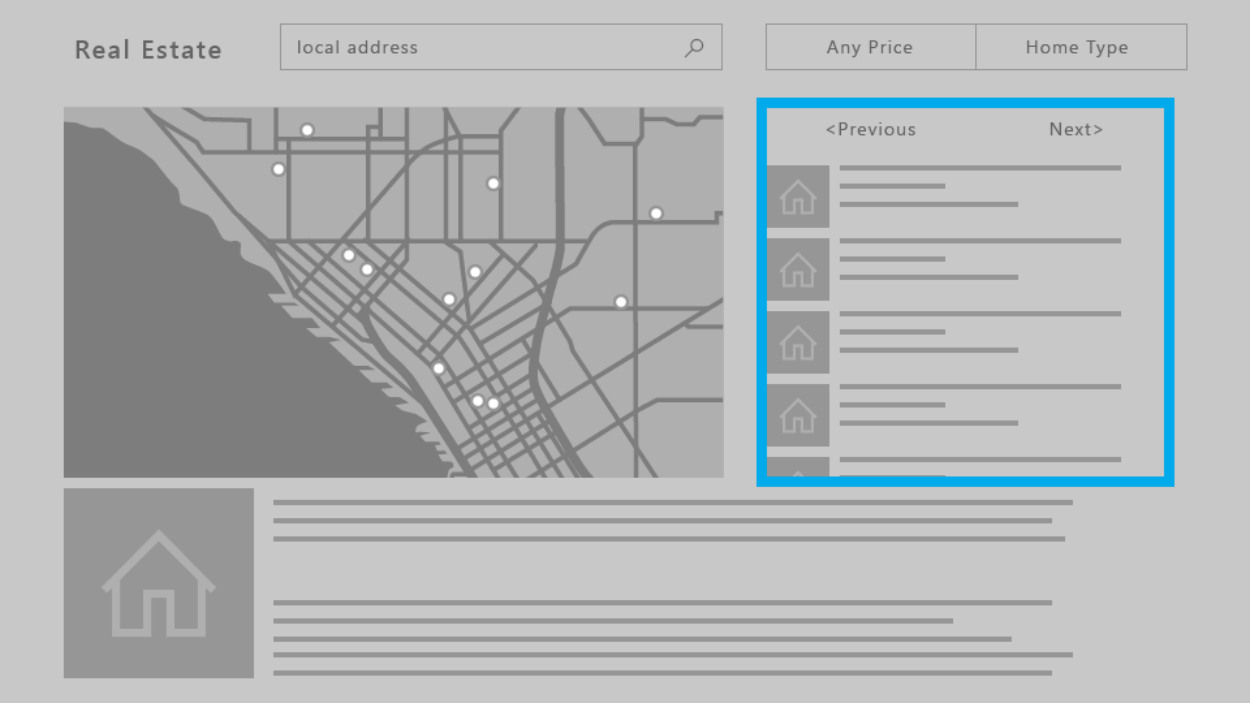
The ListView of properties shown in the following image is a very long scrolling list. If engagement is *not* required on the `ListView`, when the user navigates to the list, focus will be placed on the first item in the list. For the user to reach the **Previous** or **Next** button, they must go through all the items in the list. In cases like this where requiring the user to traverse the entire list is painful—that is, when the list is not short enough for this experience to be acceptable—you may want to consider other options.



Solutions

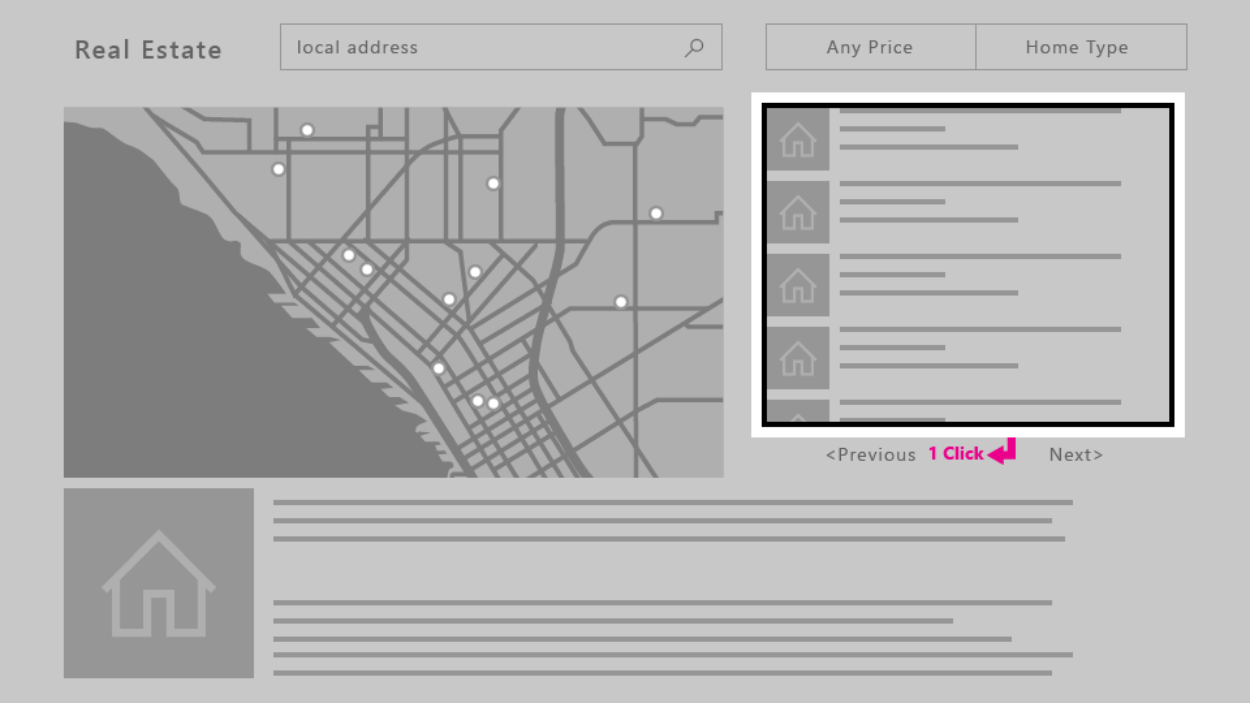
UI rearrange

Unless your initial focus is placed at the bottom of the page, UI elements placed above a long scrolling list are typically more easily accessible than if placed below. If this new layout works for other devices, changing the layout for all device families instead of doing special UI changes just for Xbox One might be a less costly approach. Additionally, placing UI elements against the scrolling direction (that is, horizontally to a vertically scrolling list, or vertically to a horizontally scrolling list) will make for even better accessibility.



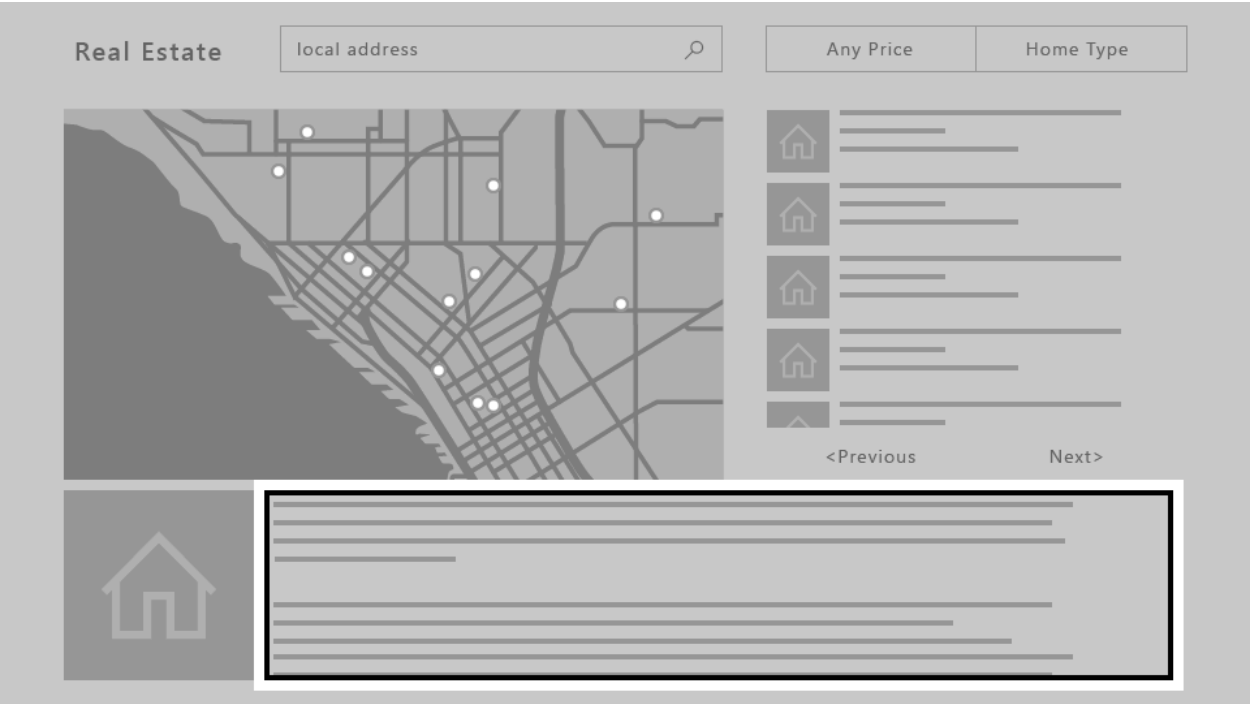
Focus engagement

When engagement is *required*, the entire `ListView` becomes a single focus target. The user will be able to bypass the contents of the list to get to the next focusable element. Read more about what controls support engagement and how to use them in Focus engagement.



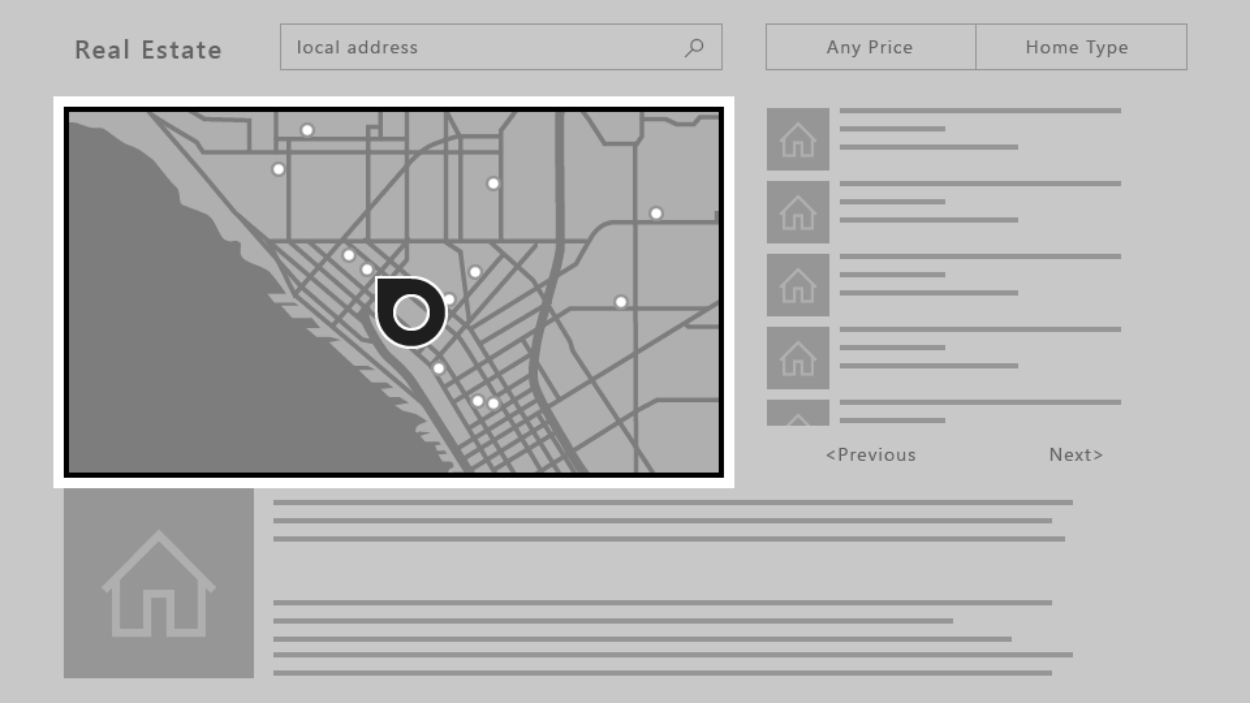
Problem: ScrollViewer without any focusable elements

Because XY focus navigation relies on navigating to one focusable UI element at a time, a ScrollViewer that doesn't contain any focusable elements (such as one with only text, as in this example) may cause a scenario where the user isn't able to view all of the content in the `ScrollViewer`. For solutions to this and other related scenarios, see Focus engagement.



Problem: Free-scrolling UI

When your app requires a freely scrolling UI, such as a drawing surface or, in this example, a map, XY focus navigation simply doesn't work. In such cases, you can turn on mouse mode to allow the user to navigate freely inside a UI element.



Mouse mode

As described in XY focus navigation and interaction, on Xbox One the focus is moved by using an XY navigation system, allowing the user to shift the focus from control to control by moving up, down, left, and right. However, some controls, such as WebView and MapControl, require a mouse-like interaction where users can freely move the pointer inside the boundaries of the control. There are also some apps where it makes sense for the user to be able to move the pointer across the entire page, having an experience with gamepad/remote similar to what users can find on a PC with a mouse.

For these scenarios, you should request a pointer (mouse mode) for the entire page, or on a control inside a page. For example, your app could have a page that has a `WebView` control that uses mouse mode only while inside the control, and XY focus navigation everywhere else. To request a pointer, you can specify whether you want it **when a control or page is engaged** or **when a page has focus**.

> **Note**
>
> Requesting a pointer when a control gets focus is not supported.

For both XAML and hosted web apps running on Xbox One, mouse mode is turned on by default for the entire app. It is highly recommended that you turn this off and

optimize your app for XY navigation. To do this, set the `Application.RequiresPointerMode` property to `WhenRequested` so that you only enable mouse mode when a control or page calls for it.

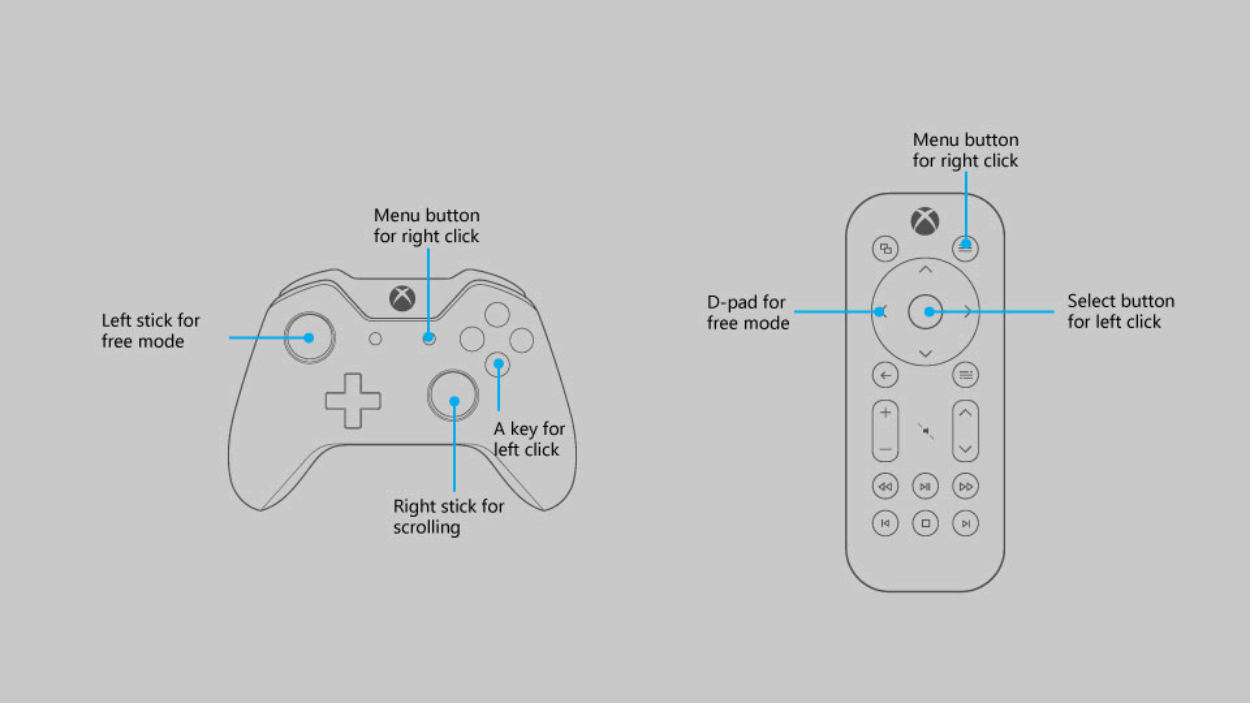
To do this in a XAML app, use the following code in your `App` class:

C#

```
public App()
{
    this.InitializeComponent();
    this.RequiresPointerMode =
        Windows.UI.Xaml.ApplicationRequiresPointerMode.WhenRequested;
    this.Suspending += OnSuspending;
}
```

For more information, including sample code for HTML/JavaScript, see How to disable mouse mode.

The following diagram shows the button mappings for gamepad/remote in mouse mode.



> **Note**
>
> Mouse mode is only supported on Xbox One with gamepad/remote. On other device families and input types it is silently ignored.

Use the RequiresPointer property on a control or page to activate mouse mode on it. This property has three possible values: `Never` (the default value), `WhenEngaged`, and `WhenFocused`.

Activating mouse mode on a control

When the user engages a control with `RequiresPointer="WhenEngaged"`, mouse mode is activated on the control until the user disengages it. The following code snippet demonstrates a simple `MapControl` that activates mouse mode when engaged:

XML

```
<Page>
    <Grid>
        <MapControl IsEngagementRequired="true"
                    RequiresPointer="WhenEngaged"/>
    </Grid>
</Page>
```

> **Note**
>
> If a control activates mouse mode when engaged, it must also require engagement with `IsEngagementRequired="true"`; otherwise, mouse mode will never be activated.

When a control is in mouse mode, its nested controls will be in mouse mode as well. The requested mode of its children will be ignored—it's impossible for a parent to be in mouse mode but a child not to be.

Additionally, the requested mode of a control is only inspected when it gets focus, so the mode won't change dynamically while it has focus.

Activating mouse mode on a page

When a page has the property `RequiresPointer="WhenFocused"`, mouse mode will be activated for the whole page when it gets focus. The following code snippet demonstrates giving a page this property:

XML

```
<Page RequiresPointer="WhenFocused">
    ...
</Page>
```

> **Note**
>
> The `WhenFocused` value is only supported on **Page** objects. If you try to set this value on a control, an exception will be thrown.

Disabling mouse mode for full screen content

Usually when displaying video or other types of content in full screen, you will want to hide the cursor because it can distract the user. This scenario occurs when the rest of the app uses mouse mode, but you want to turn it off when showing full screen content. To accomplish this, put the full screen content on its own `Page`, and follow the steps below.

1. In the `App` object, set `RequiresPointerMode="WhenRequested"`.
2. In every `Page` object *except* for the full screen `Page`, set `RequiresPointer="WhenFocused"`.
3. For the full screen `Page`, set `RequiresPointer="Never"`.

This way, the cursor will never appear when showing full screen content.

Focus visual

The focus visual is the border around the UI element that currently has focus. This helps orient the user so that they can easily navigate your UI without getting lost.

With a visual update and numerous customization options added to focus visual, developers can trust that a single focus visual will work well on PCs and Xbox One, as well as on any other Windows devices that support keyboard and/or gamepad/remote.

While the same focus visual can be used across different platforms, the context in which the user encounters it is slightly different for the 10-foot experience. You should assume that the user is not paying full attention to the entire TV screen, and therefore it is important that the currently focused element is clearly visible to the user at all times to avoid the frustration of searching for the visual.

It is also important to keep in mind that the focus visual is displayed by default when using a gamepad or remote control, but *not* a keyboard. Thus, even if you don't implement it, it will appear when you run your app on Xbox One.

Initial focus visual placement

When launching an app or navigating to a page, place the focus on a UI element that makes sense as the first element on which the user would take action. For example, a photo app may place focus on the first item in the gallery, and a music app navigated to a detailed view of a song might place focus on the play button for ease of playing music.

Try to put initial focus in the top left region of your app (or top right for a right-to-left flow). Most users tend to focus on that corner first because that's where app content flow generally begins.

Making focus clearly visible

One focus visual should always be visible on the screen so that the user can pick up where they left off without searching for the focus. Similarly, there should be a focusable item onscreen at all times—for example, don't use pop-ups with only text and no focusable elements.

An exception to this rule would be for full-screen experiences, such as watching videos or viewing images, in which cases it would not be appropriate to show the focus visual.

Reveal focus

Reveal focus is a lighting effect that animates the border of focusable elements, such as a button, when the user moves gamepad or keyboard focus to them. By animating the glow around the border of the focused elements, Reveal focus gives users a better understanding of where focus is and where focus is going.

Reveal focus is off by default. For 10 foot experiences you should opt-in to reveal focus by setting the Application.FocusVisualKind property in your app constructor.

C#

```
    if(AnalyticsInfo.VersionInfo.DeviceFamily == "Windows.Xbox")
    {
        this.FocusVisualKind = FocusVisualKind.Reveal;
    }
```

For more information see the guidance for Reveal focus.

Customizing the focus visual

If you'd like to customize the focus visual, you can do so by modifying the properties related to the focus visual for each control. There are several such properties that you

can take advantage of to personalize your app.

You can even opt out of the system-provided focus visuals by drawing your own using visual states. To learn more, see VisualState.

Light dismiss overlay

To call the user's attention to the UI elements that the user is currently manipulating with the game controller or remote control, UWP automatically adds a "smoke" layer that covers areas outside of the popup UI when the app is running on Xbox One. This requires no extra work, but is something to keep in mind when designing your UI. You can set the `LightDismissOverlayMode` property on any `FlyoutBase` to enable or disable the smoke layer; it defaults to `Auto`, meaning that it is enabled on Xbox and disabled elsewhere. For more information, see Modal vs light dismiss.

Focus engagement

Focus engagement is intended to make it easier to use a gamepad or remote to interact with an app.

> ⓘ **Note**
>
> Setting focus engagement does not impact keyboard or other input devices.

When the property `IsFocusEngagementEnabled` on a FrameworkElement object is set to `True`, it marks the control as requiring focus engagement. This means that the user must press the **A/Select** button to "engage" the control and interact with it. When they are finished, they can press the **B/Back** button to disengage the control and navigate out of it.

> ⓘ **Note**
>
> `IsFocusEngagementEnabled` is a new API and not yet documented.

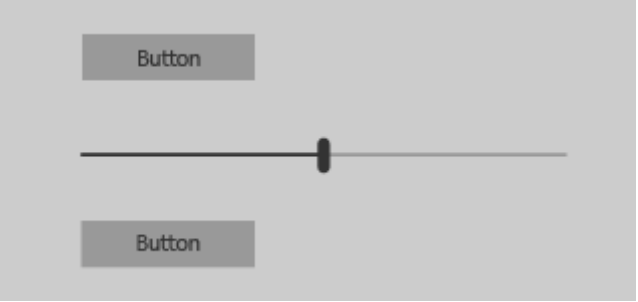
Focus trapping

Focus trapping is what happens when a user attempts to navigate an app's UI but becomes "trapped" within a control, making it difficult or even impossible to move outside of that control.

The following example shows UI that creates focus trapping.

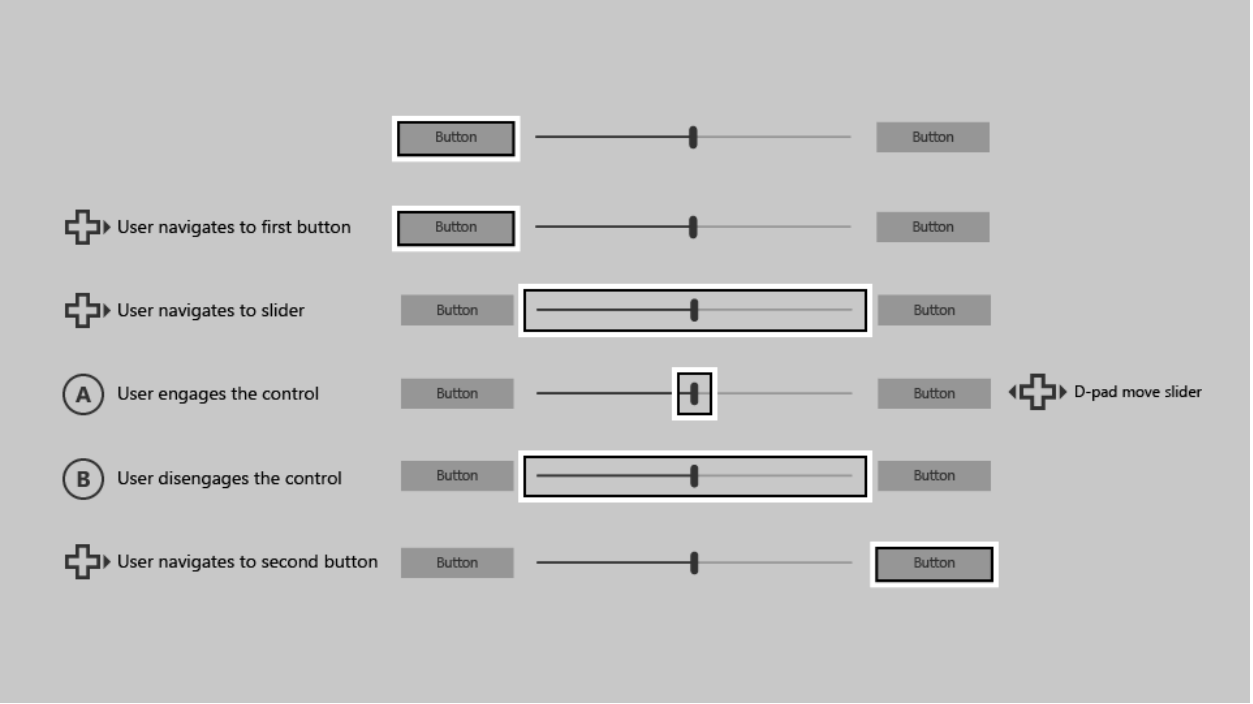
If the user wants to navigate from the left button to the right button, it would be logical to assume that all they'd have to do is press right on the D-pad/left stick twice. However, if the Slider doesn't require engagement, the following behavior would occur: when the user presses right the first time, focus would shift to the `Slider`, and when they press right again, the `Slider`'s handle would move to the right. The user would keep moving the handle to the right and wouldn't be able to get to the button.

There are several approaches to getting around this issue. One is to design a different layout, similar to the real estate app example in XY focus navigation and interaction where we relocated the **Previous** and **Next** buttons above the `ListView`. Stacking the controls vertically instead of horizontally as in the following image would solve the problem.



Now the user can navigate to each of the controls by pressing up and down on the D-pad/left stick, and when the `Slider` has focus, they can press left and right to move the `Slider` handle, as expected.

Another approach to solving this problem is to require engagement on the `Slider`. If you set `IsFocusEngagementEnabled="True"`, this will result in the following behavior.



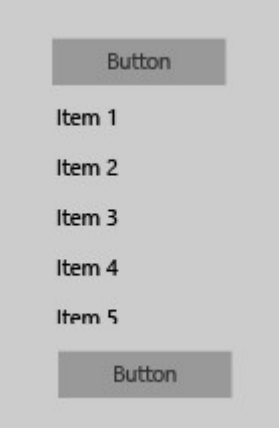
When the `Slider` requires focus engagement, the user can get to the button on the right simply by pressing right on the D-pad/left stick twice. This solution is great because it requires no UI adjustment and produces the expected behavior.

Items controls

Aside from the Slider control, there are other controls which you may want to require engagement, such as:

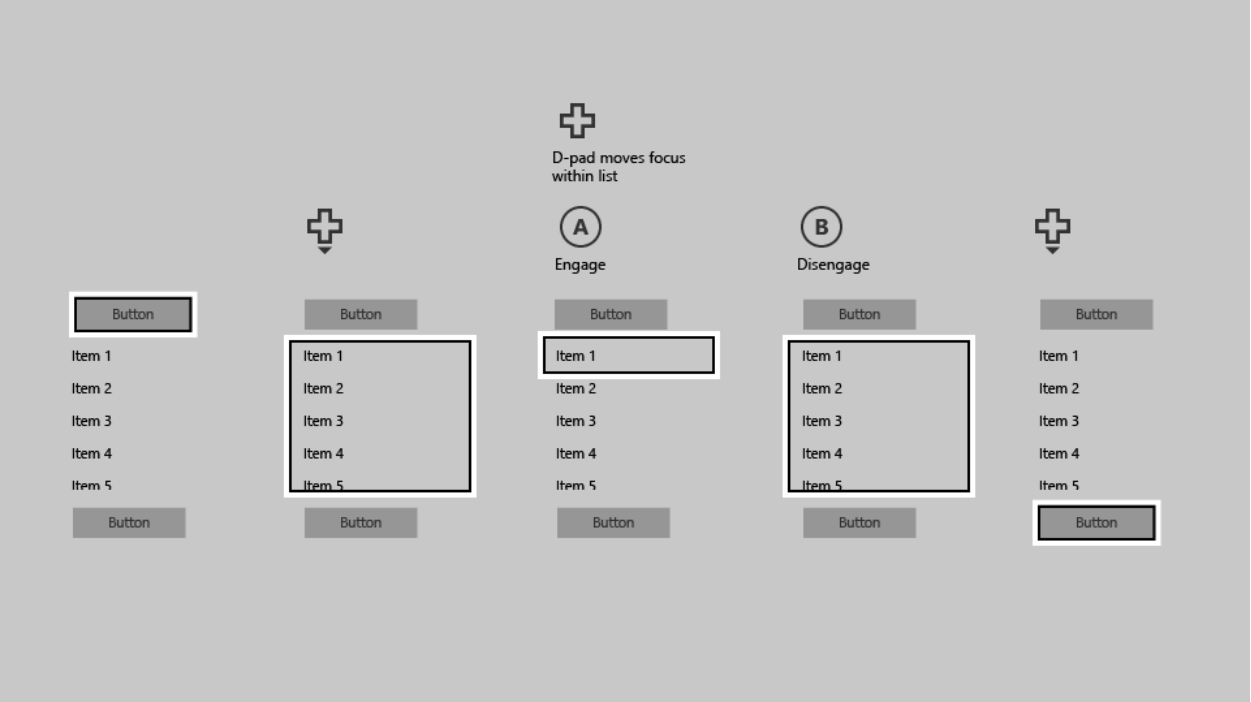
- ListBox
- ListView
- GridView
- FlipView

Unlike the `Slider` control, these controls don't trap focus within themselves; however, they can cause usability issues when they contain large amounts of data. The following is an example of a `ListView` that contains a large amount of data.



Similar to the `Slider` example, let's try to navigate from the button at the top to the button at the bottom with a gamepad/remote. Starting with focus on the top button, pressing down on the D-pad/stick will place the focus on the first item in the `ListView` ("Item 1"). When the user presses down again, the next item in the list gets focus, not the button on the bottom. To get to the button, the user must navigate through every item in the `ListView` first. If the `ListView` contains a large amount of data, this could be inconvenient and not an optimal user experience.

To solve this problem, set the property `IsFocusEngagementEnabled="True"` on the `ListView` to require engagement on it. This will allow the user to quickly skip over the `ListView` by simply pressing down. However, they will not be able to scroll through the list or choose an item from it unless they engage it by pressing the **A/Select** button when it has focus, and then pressing the **B/Back** button to disengage.



ScrollViewer

Slightly different from these controls is the ScrollViewer, which has its own quirks to consider. If you have a `ScrollViewer` with focusable content, by default navigating to the `ScrollViewer` will allow you to move through its focusable elements. Like in a `ListView`, you must scroll through each item to navigate outside of the `ScrollViewer`.

If the `ScrollViewer` has *no* focusable content—for example, if it only contains text—you can set `IsFocusEngagementEnabled="True"` so the user can engage the `ScrollViewer` by using the **A/Select** button. After they have engaged, they can scroll through the text by using the **D-pad/left stick**, and then press the **B/Back** button to disengage when they're finished.

Another approach would be to set `IsTabStop="True"` on the `ScrollViewer` so that the user doesn't have to engage the control—they can simply place focus on it and then scroll by using the **D-pad/left stick** when there are no focusable elements within the `ScrollViewer`.

Focus engagement defaults

Some controls cause focus trapping commonly enough to warrant their default settings to require focus engagement, while others have focus engagement turned off by default but can benefit from turning it on. The following table lists these controls and their default focus engagement behaviors.

Expand table

Control	Focus engagement default
CalendarDatePicker	On
FlipView	Off
GridView	Off
ListBox	Off
ListView	Off
ScrollViewer	Off
SemanticZoom	Off
Slider	On

All other Windows controls will result in no behavioral or visual changes when `IsFocusEngagementEnabled="True"`.

Summary

You can build Windows applications that are optimized for a specific device or experience, but the Universal Windows Platform also enables you to build apps that can be used successfully across devices, in both 2-foot and 10-foot experiences, and regardless of input device or user ability. Using the recommendations in this article can ensure that your app is as good as it can be on both the TV and a PC.

Related articles

- Designing for Xbox and TV
- Device primer for Windows apps

Touchpad design guidelines

Article • 06/24/2021

Design your app so that users can interact with it through a touchpad. A touchpad combines both indirect multi-touch input with the precision input of a pointing device, such as a mouse. This combination makes the touchpad suited to both a touch-optimized UI and the smaller targets of productivity apps.

Touchpad interactions require three things:

- A standard touchpad or a Windows Precision Touchpad.

  Precision touchpads are optimized for Windows app devices. They enable the system to handle certain aspects of the touchpad experience natively, such as finger tracking and palm detection, for a more consistent experience across devices.

- The direct contact of one or more fingers on the touchpad.

- Movement of the touch contacts (or lack thereof, based on a time threshold).

The input data provided by the touchpad sensor can be:

- Interpreted as a physical gesture for direct manipulation of one or more UI elements (such as panning, rotating, resizing, or moving). In contrast, interacting with an element through its properties window or other dialog box is considered indirect manipulation.
- Recognized as an alternative input method, such as mouse or pen.
- Used to complement or modify aspects of other input methods, such as smudging an ink stroke drawn with a pen.

A touchpad combines indirect multi-touch input with the precision input of a pointing device, such as a mouse. This combination makes the touchpad suited to both touch-optimized UI and the typically smaller targets of productivity apps and the desktop environment. Optimize your Windows app design for touch input and get touchpad support by default.

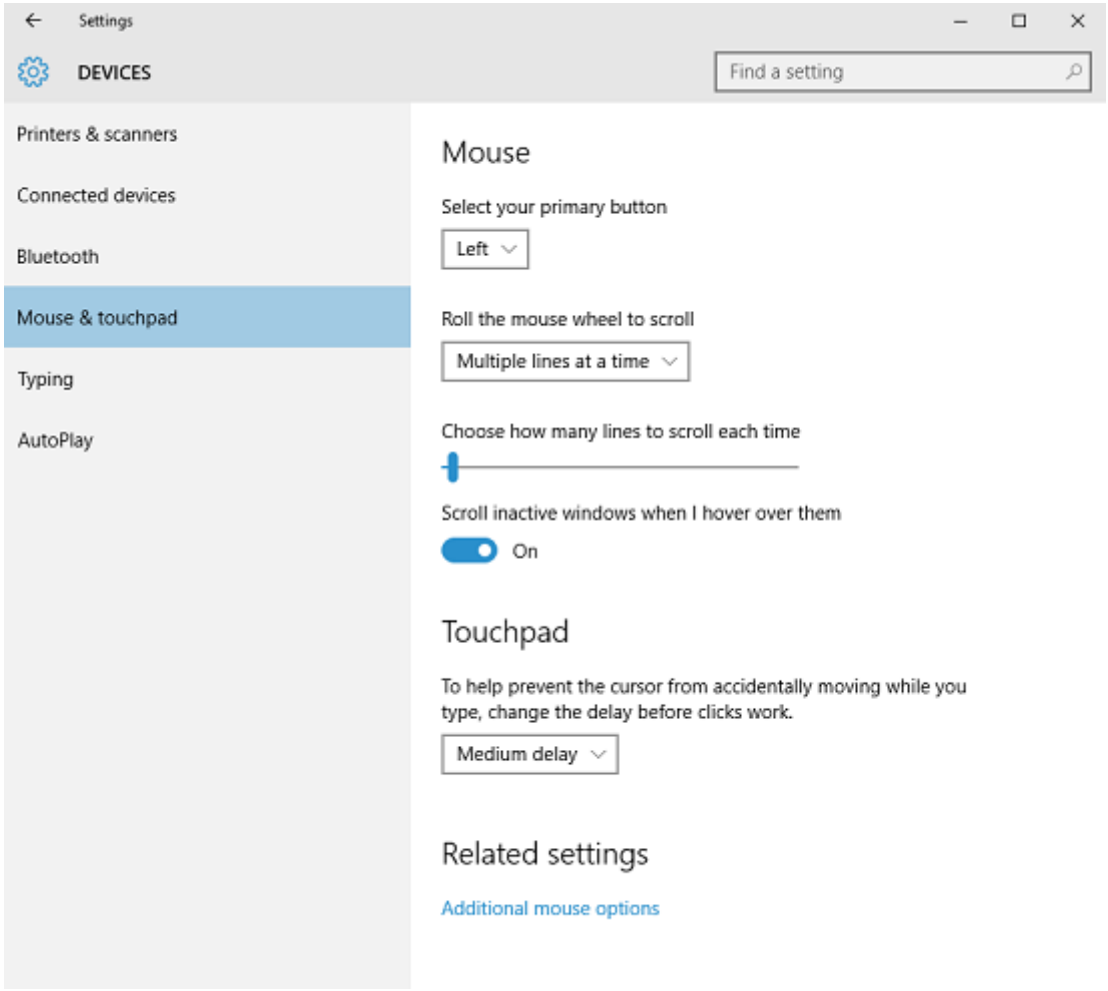
Because of the convergence of interaction experiences supported by touchpads, we recommend using the **PointerEntered** event to provide mouse-style UI commands in addition to the built-in support for touch input. For example, use previous and next buttons to let users flip through pages of content as well as pan through the content.

The gestures and guidelines discussed in this topic can help to ensure that your app supports touchpad input seamlessly and with minimal code.

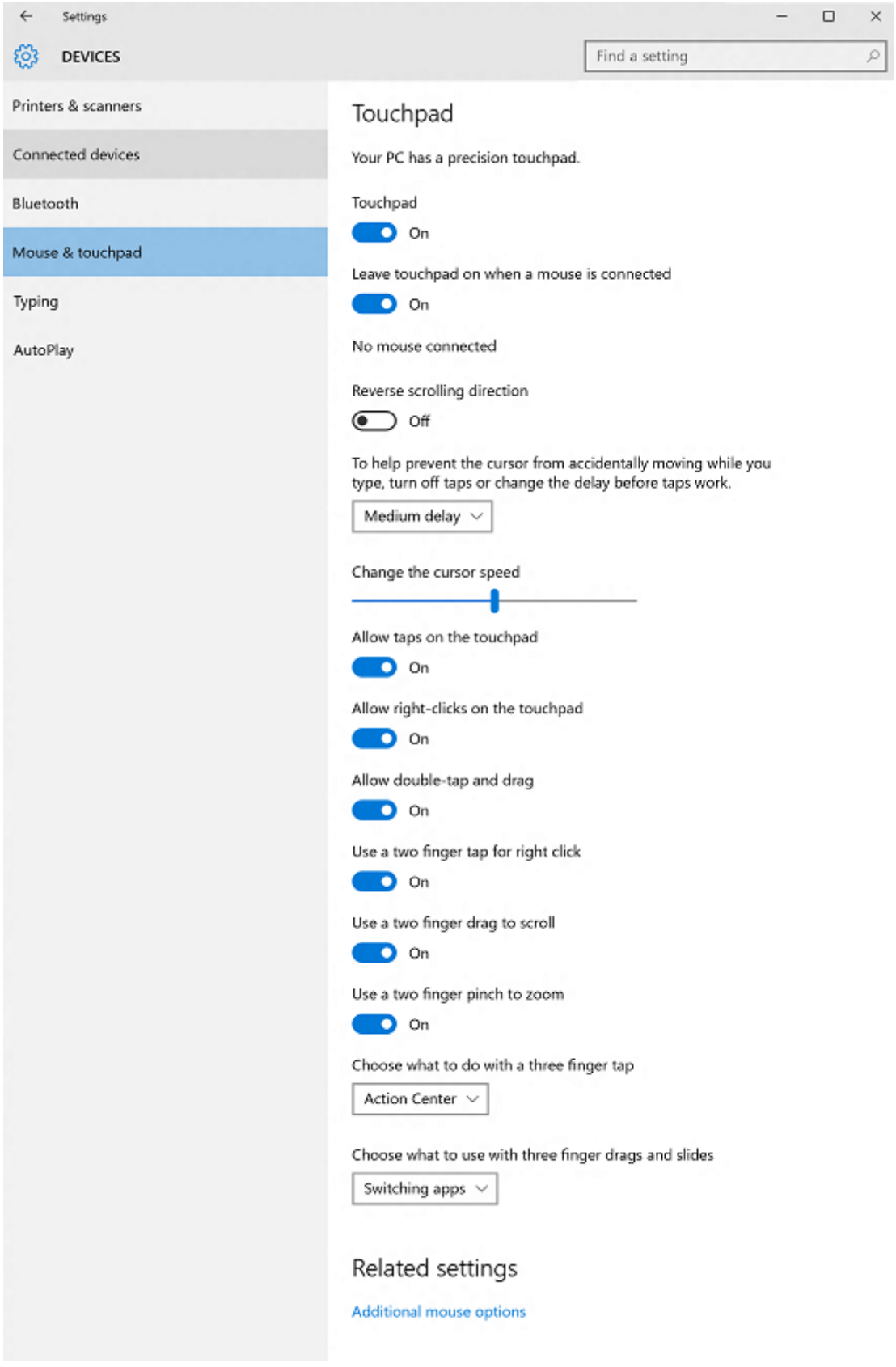
The touchpad language

A concise set of touchpad interactions are used consistently throughout the system. Optimize your app for touch and mouse input and this language makes your app feel instantly familiar for your users, increasing their confidence and making your app easier to learn and use.

Users can set far more Precision Touchpad gestures and interaction behaviors than they can for a standard touchpad. These two images show the different touchpad settings pages from Settings > Devices > Mouse & touchpad for a standard touchpad and a Precision Touchpad, respectively.



Standard\ touchpad\ settings



Windows\ Precision\ Touchpad\ settings

Here are some examples of touchpad-optimized gestures for performing common tasks.

Term	Description
Three-finger tap	User preference to search with **Cortana** or show **Action Center**.

Term	Description
Three finger slide	User preference to open the virtual desktop Task View, show Desktop, or switch between open apps.
Single finger tap for primary action	Use a single finger to tap an element and invoke its primary action (such as launching an app or executing a command).
Two finger tap to right-click	Tap with two fingers simultaneously on an element to select it and display contextual commands.
Two finger slide to pan	Slide is used primarily for panning interactions but can also be used for moving, drawing, or writing.
Pinch and stretch to zoom	The pinch and stretch gestures are commonly used for resizing and Semantic Zoom.
Single finger press and slide to rearrange	Drag an element.
Single finger press and slide to select text	Press within selectable text and slide to select it. Double-tap to select a word.
Left and right click zone	Emulate the left and right button functionality of a mouse device.

Hardware

Query the mouse device capabilities (**MouseCapabilities**) to identify what aspects of your app UI the touchpad hardware can access directly. We recommend providing UI for both touch and mouse input.

For more info about querying device capabilities, see Identify input devices.

Visual feedback

- When a touchpad cursor is detected (through move or hover events), show mouse-specific UI to indicate functionality exposed by the element. If the touchpad cursor doesn't move for a certain amount of time, or if the user initiates a touch interaction, make the touchpad UI gradually fade away. This keeps the UI clean and uncluttered.

- Don't use the cursor for hover feedback, the feedback provided by the element is sufficient (see the Cursors section below).
- Don't display visual feedback if an element doesn't support interaction (such as static text).
- Don't use focus rectangles with touchpad interactions. Reserve these for keyboard interactions.
- Display visual feedback concurrently for all elements that represent the same input target.

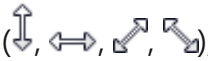
For more general guidance about visual feedback, see Guidelines for visual feedback.

Cursors

A set of standard cursors is available for a touchpad pointer. These are used to indicate the primary action of an element.

Each standard cursor has a corresponding default image associated with it. The user or an app can replace the default image associated with any standard cursor at any time. UWP apps specify a cursor image through the **PointerCursor** function.

If you need to customize the mouse cursor:

- Always use the arrow cursor ( ) for clickable elements. don't use the pointing hand cursor ( ) for links or other interactive elements. Instead, use hover effects (described earlier).
- Use the text cursor ( ) for selectable text.
- Use the move cursor ( ) when moving is the primary action (such as dragging or cropping). Don't use the move cursor for elements where the primary action is navigation (such as Start tiles).
- Use the horizontal, vertical and diagonal resize cursors (), when an object is resizable.
- Use the grasping hand cursors ( ) when panning content within a fixed canvas (such as a map).

Related articles

- Handle pointer input
- Identify input devices

Samples

- Basic input sample
- Low latency input sample
- User interaction mode sample
- Focus visuals sample

Archive Samples

- Input: Device capabilities sample
- Input: XAML user input events sample
- XAML scrolling, panning, and zooming sample
- Input: Gestures and manipulations with GestureRecognizer

Surface Dial interactions

Article • 10/06/2022

Surface Dial with Surface Studio and Pen (available for purchase at the Microsoft Store).

Overview

Windows wheel devices, such as the Surface Dial, are a new category of input device that enable a host of compelling and unique user interaction experiences for Windows and Windows apps.

> **Important**
>
> In this topic, we refer specifically to Surface Dial interactions, but the info is applicable to all Windows wheel devices.

With a form factor based on a *rotate* action (or gesture), the Surface Dial is intended as a secondary, multi-modal input device that complements input from a primary device. In most cases, the device is manipulated by a user's non-dominant hand while performing a task with their dominant hand (such as inking with a pen). It is not designed for precision pointer input (like touch, pen, or mouse).

The Surface Dial also supports both a *press and hold* action and a *click* action. Press and hold has a single function: display a menu of commands. If the menu is active, the rotate and click input is processed by the menu. Otherwise, the input is passed to your app for processing.

As with all Windows input devices, you can customize and tailor the Surface Dial interaction experience to suit the functionality in your apps.

Tip

Used together, the Surface Dial and the new Surface Studio can provide an even more distinctive user experience.

In addition to the default press and hold menu experience described, the Surface Dial can also be placed directly on the screen of the Surface Studio. This enables a special "on-screen" menu.

By detecting both the contact location and bounds of the Surface Dial, the system uses this info to handle occlusion by the device and display a larger version of the menu that wraps around the outside of the Dial. This same info can also be used by your app to adapt the UI for both the presence of the device and its anticipated usage, such as the placement of the user's hand and arm.

Surface Dial off-screen menu

Surface Dial on-screen menu

System integration

The Surface Dial is tightly integrated with Windows and supports a set of built-in tools on the menu: system volume, scroll, zoom in/out, and undo/redo.

This collection of built-in tools adapts to the current system context to include:

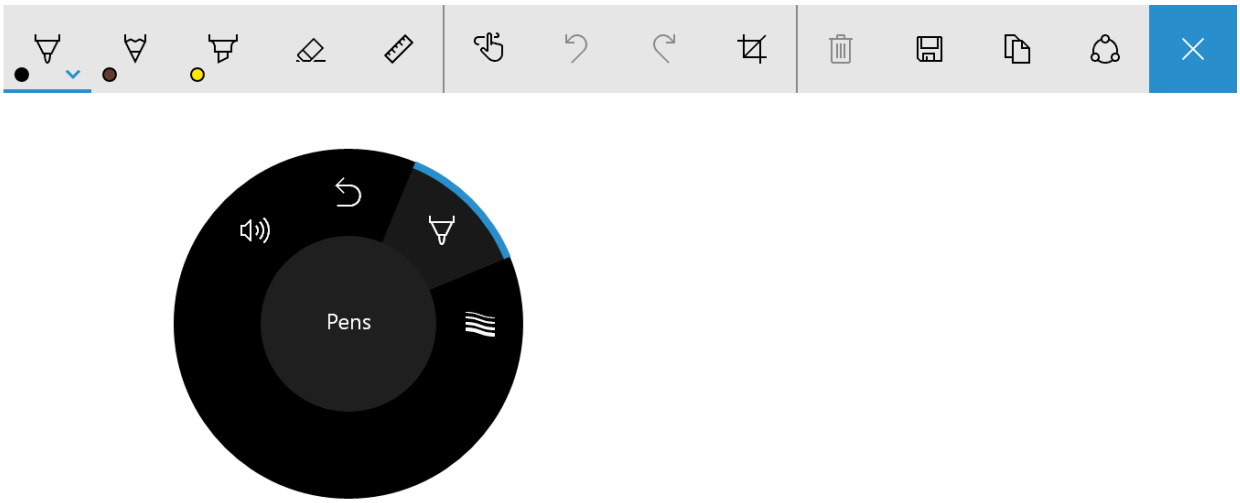
- A system brightness tool when the user is on the Windows Desktop
- A previous/next track tool when media is playing

In addition to this general platform support, the Surface Dial is also tightly integrated with the Windows Ink platform controls (**InkCanvas** and **InkToolbar**).

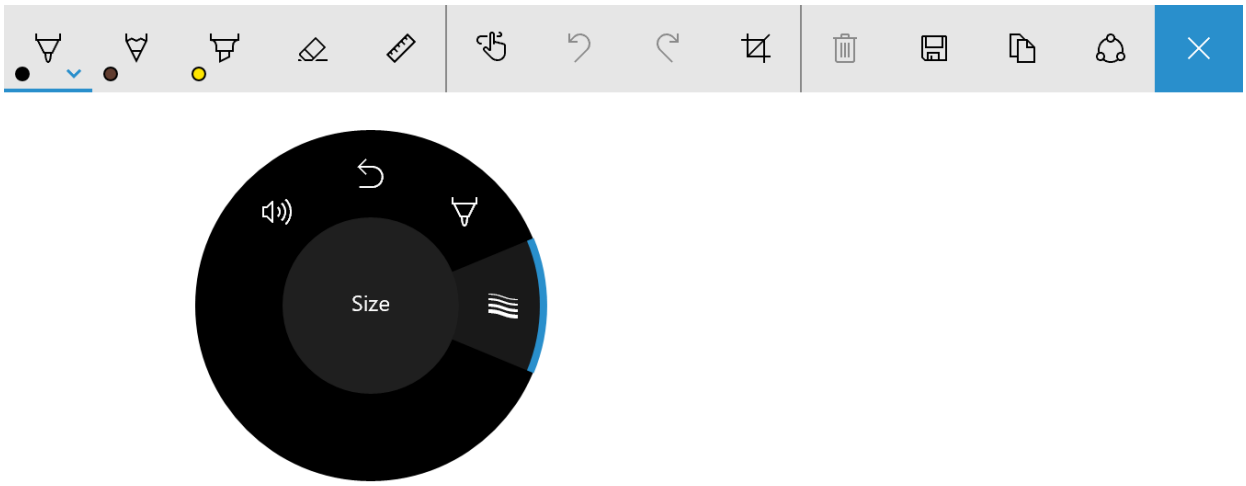
Surface Dial with Surface Pen

When used with the Surface Dial, these controls enable additional functionality for modifying ink attributes and controlling the ink toolbar's ruler stencil.

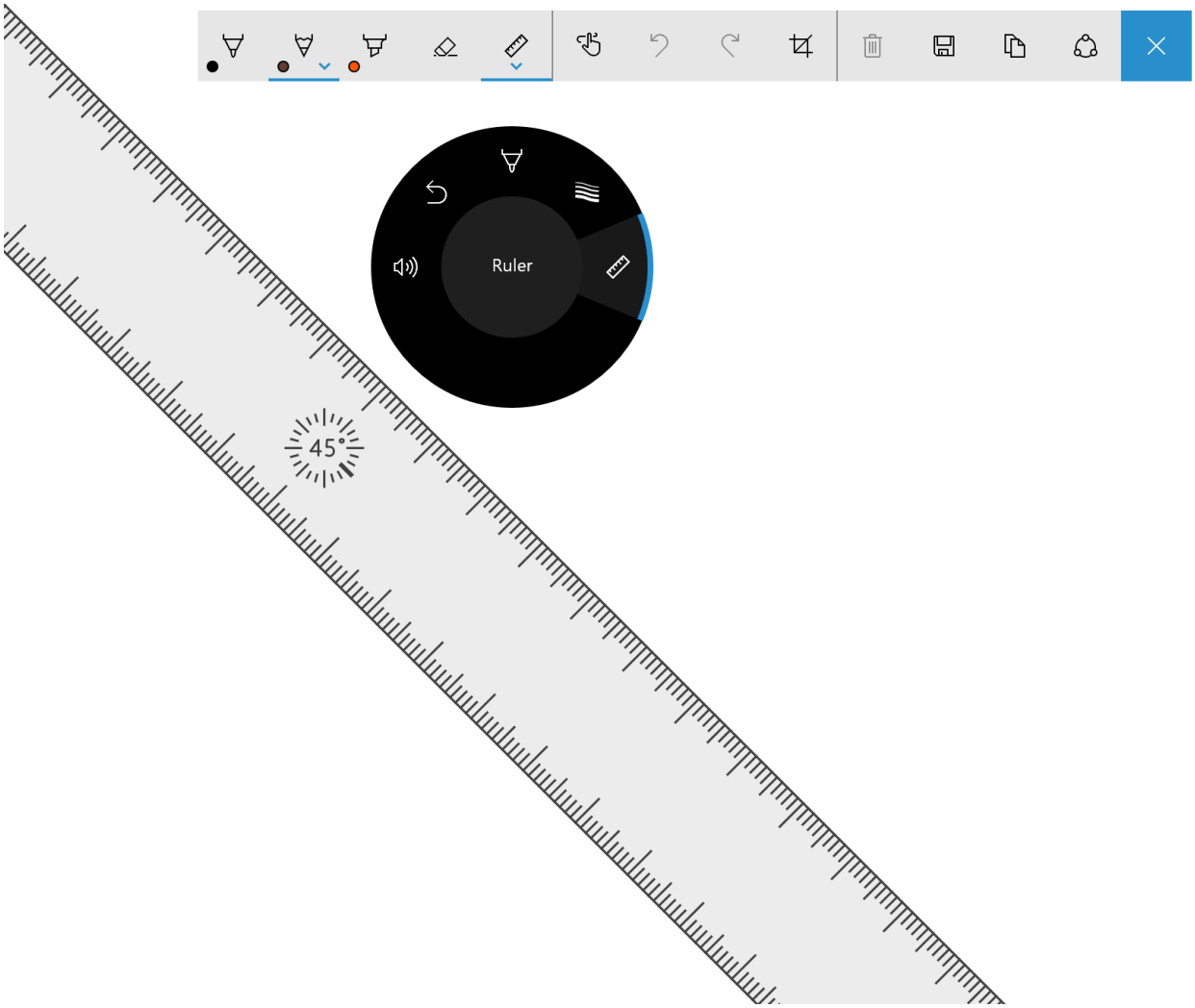
When you open the Surface Dial Menu in an inking application that uses the ink toolbar, the menu now includes tools for controlling pen type and brush thickness. When the ruler is enabled, a corresponding tool is added to the menu that lets the device control the position and angle of the ruler.



Surface Dial menu with pen selection tool for the Windows Ink toolbar



Surface Dial menu with stroke size tool for the Windows Ink toolbar



Surface Dial menu with ruler tool for the Windows Ink toolbar

User customization

Users can customize some aspects of their Dial experience through the **Windows Settings** -> **Devices** -> **Wheel** page, including default tools, vibration (or haptic feedback), and writing (or dominant) hand.

When customizing the Surface Dial user experience, you should always ensure that a particular function or behavior is available and enabled by the user.

Custom tools

Here we discuss both UX and developer guidance for customizing the tools exposed on the Surface Dial menu.

UX guidance for custom tools

Ensure your tools correspond to the current context When you make it clear and intuitive what a tool does and how the Surface Dial interaction works, you help users learn quickly and stay focused on their task.

Minimize the number of app tools as much as possible
The Surface Dial menu has room for seven items. If there are eight or more items, the user needs to turn the Dial to see which tools are available in an overflow flyout, making the menu difficult to navigate and tools difficult to discover and select.

We recommend providing a single custom tool for your app or app context. Doing so enables you to set that tool based on what the user is doing without requiring them to activate the Surface Dial menu and select a tool.

Dynamically update the collection of tools
Because Surface Dial menu items do not support a disabled state, you should dynamically add and remove tools (including built-in, default tools) based on user context (current view or focused window). If a tool is not relevant to the current activity or it's redundant, remove it.

> **ⓘ Important**
>
> When you add an item to the menu, ensure the item does not already exist.

Don't remove the built-in system volume setting tool
Volume control is typically always required by user. They might be listening to music while using your app, so volume and next track tools should always be accessible from the Surface Dial menu. (The next track tool is automatically added to the menu when media is playing.)

Be consistent with menu organization
This helps users with discovering and learning what tools are available when using your

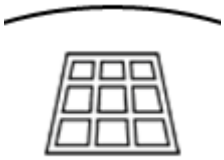
app, and helps improve their efficiency when switching tools.

Provide high-quality icons consistent with the built-in icons
Icons can convey professionalism and excellence, and inspire trust in users.

- Provide a high-quality 64 x 64 pixel PNG image (44 x 44 is the smallest supported)
- Ensure the background is transparent
- The icon should fill most of the image
- A white icon should have a black outline to be visible in high contrast mode

Icon with alpha background

Icon displayed on wheel menu with default theme



Icon displayed on wheel menu with High Contrast White theme

Use concise and descriptive names
The tool name is displayed in the tool menu along with the tool icon and is also used by screen readers.

- Names should be short to fit inside the central circle of the wheel menu
- Names should clearly identify the primary action (a complementary action can be implied):
  - Scroll indicates the effect of both rotation directions
  - Undo specifies a primary action, but redo (the complementary action) can be inferred and easily discovered by the user

Developer guidance

You can customize the Surface Dial experience to complement the functionality in your apps through a comprehensive set of Windows Runtime APIs.

As previously mentioned, the default Surface Dial menu is pre-populated with a set of built-in tools covering a broad range of basic system features (system volume, system brightness, scroll, zoom, undo, and media control when the system detects ongoing audio or video playback). However, these default tools might not provide the functionality required by your app.

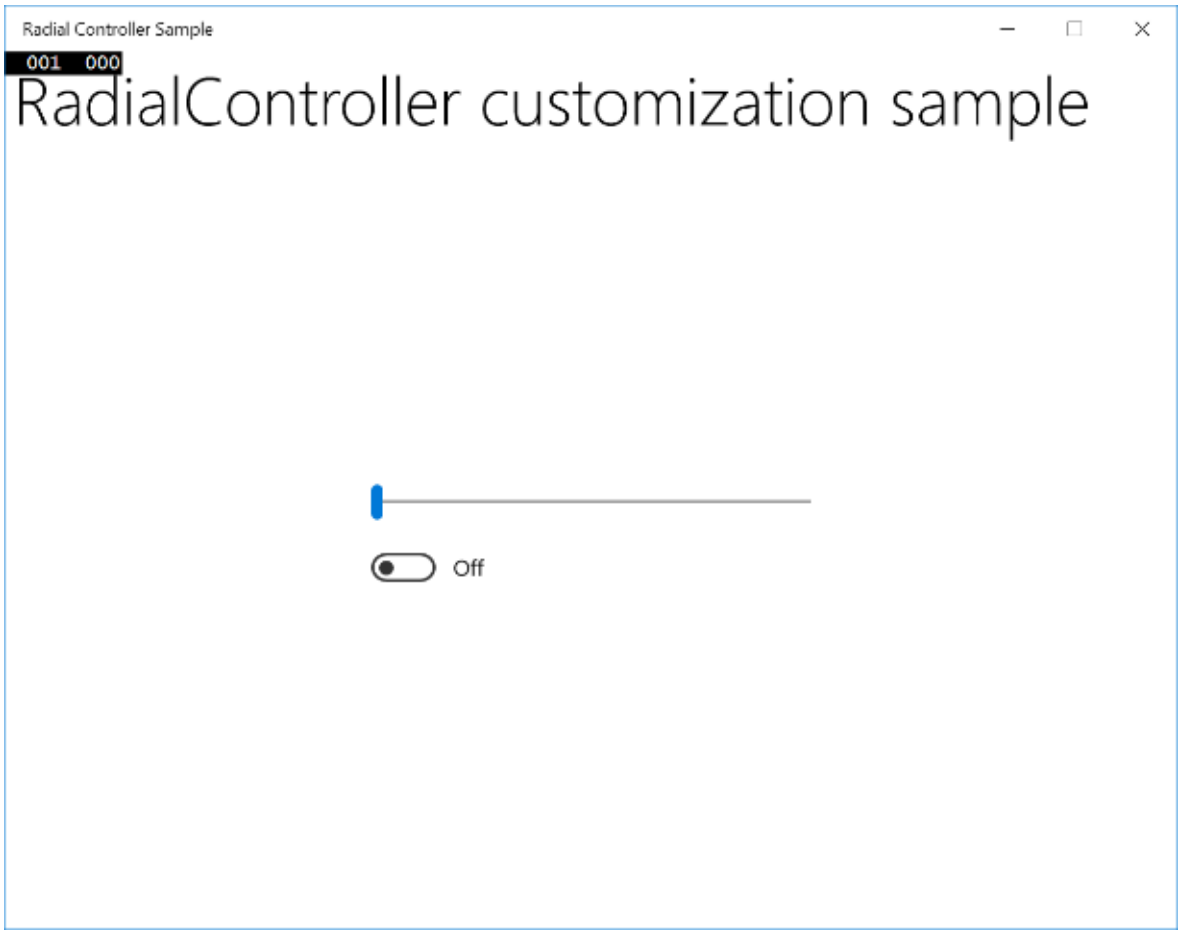
In the following sections, we describe how to add a custom tool to the Surface Dial menu and specify which built-in tools are exposed.

Download a more robust version of this sample from RadialController customization.

Add a custom tool

In this example, we add a basic custom tool that passes the input data from both the rotation and click events to some XAML UI controls.

1. First, we declare our UI (just a slider and toggle button) in XAML.



The sample app UI

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
  </Grid.RowDefinitions>
  <StackPanel x:Name="HeaderPanel"
    Orientation="Horizontal"
    Grid.Row="0">
      <TextBlock x:Name="Header"
        Text="RadialController customization sample"
        VerticalAlignment="Center"
        Style="{ThemeResource HeaderTextBlockStyle}"
        Margin="10,0,0,0" />
  </StackPanel>
  <StackPanel Orientation="Vertical"
    VerticalAlignment="Center"
    HorizontalAlignment="Center"
    Grid.Row="1">
      <!-- Slider for rotation input -->
      <Slider x:Name="RotationSlider"
        Width="300"
        HorizontalAlignment="Left"/>
      <!-- Switch for click input -->
      <ToggleSwitch x:Name="ButtonToggle"
        HorizontalAlignment="Left"/>
  </StackPanel>
</Grid>
```

2. Then, in code-behind, we add a custom tool to the Surface Dial menu and declare the **RadialController** input handlers.

   We get a reference to the **RadialController** object for the Surface Dial (myController) by calling **CreateForCurrentView**.

   We then create an instance of a **RadialControllerMenuItem** (myItem) by calling **RadialControllerMenuItem.CreateFromIcon**.

   Next, we append that item to the collection of menu items.

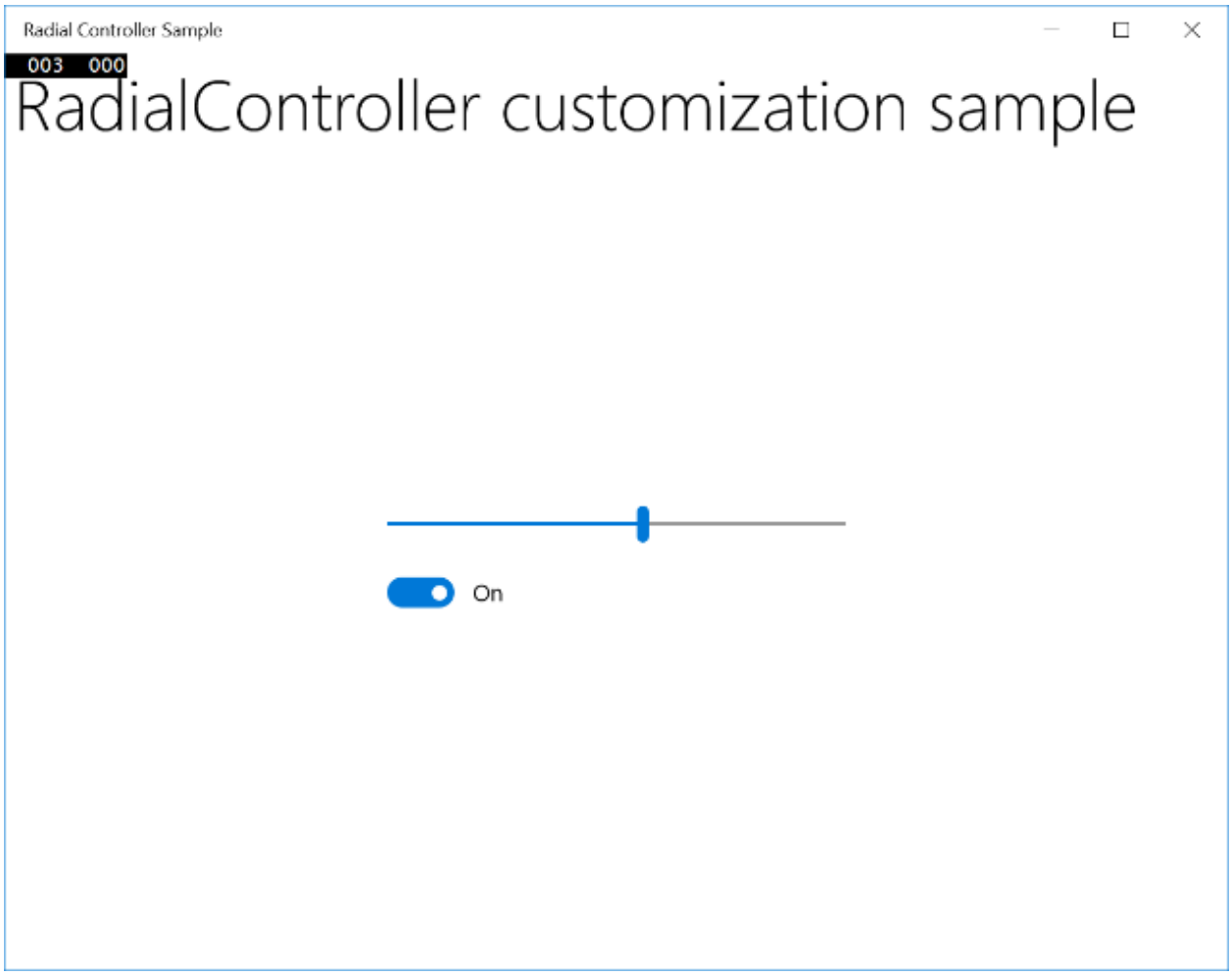
   We declare the input event handlers (**ButtonClicked** and **RotationChanged**) for the **RadialController** object.

   Finally, we define the event handlers.

```
C#

public sealed partial class MainPage : Page
{
    RadialController myController;
```

```
    public MainPage()
    {
        this.InitializeComponent();
        // Create a reference to the RadialController.
        myController = RadialController.CreateForCurrentView();

        // Create an icon for the custom tool.
        RandomAccessStreamReference icon =
          RandomAccessStreamReference.CreateFromUri(
            new Uri("ms-appx:///Assets/StoreLogo.png"));

        // Create a menu item for the custom tool.
        RadialControllerMenuItem myItem =
          RadialControllerMenuItem.CreateFromIcon("Sample", icon);

        // Add the custom tool to the RadialController menu.
        myController.Menu.Items.Add(myItem);

        // Declare input handlers for the RadialController.
        myController.ButtonClicked += MyController_ButtonClicked;
        myController.RotationChanged += MyController_RotationChanged;
    }

    // Handler for rotation input from the RadialController.
    private void MyController_RotationChanged(RadialController sender,
      RadialControllerRotationChangedEventArgs args)
    {
        if (RotationSlider.Value + args.RotationDeltaInDegrees > 100)
        {
            RotationSlider.Value = 100;
            return;
        }
        else if (RotationSlider.Value + args.RotationDeltaInDegrees <
0)
        {
            RotationSlider.Value = 0;
            return;
        }
        RotationSlider.Value += args.RotationDeltaInDegrees;
    }

    // Handler for click input from the RadialController.
    private void MyController_ButtonClicked(RadialController sender,
      RadialControllerButtonClickedEventArgs args)
    {
        ButtonToggle.IsOn = !ButtonToggle.IsOn;
    }
}
```

When we run the app, we use the Surface Dial to interact with it. First, we press and hold to open the menu and select our custom tool. Once the custom tool is activated, the slider control can be adjusted by rotating the Dial and the switch can be toggled by clicking the Dial.



The sample app UI activated using the Surface Dial custom tool

Specify the built-in tools

You can use the **RadialControllerConfiguration** class to customize the collection of built-in menu items for your app.

For example, if your app doesn't have any scrolling or zooming regions and doesn't require undo/redo functionality, these tools can be removed from the menu. This opens space on the menu to add custom tools for your app.

> **ⓘ Important**
>
> The Surface Dial menu must have at least one menu item. If all default tools are removed before you add one of your custom tools, the default tools are restored and your tool is appended to the default collection.

Per the design guidelines, we do not recommend removing the media control tools (volume and previous/next track) as users often have background music playing while they perform other tasks.

Here, we show how to configure the Surface Dial menu to include only media controls for volume and next/previous track.

```C#
public MainPage()
{
  ...
  //Remove a subset of the default system tools
  RadialControllerConfiguration myConfiguration =
  RadialControllerConfiguration.GetForCurrentView();
  myConfiguration.SetDefaultMenuItems(new[]
  {
    RadialControllerSystemMenuItemKind.Volume,
      RadialControllerSystemMenuItemKind.NextPreviousTrack
  });
}
```

Custom interactions

As mentioned, the Surface Dial supports three gestures (press and hold, rotate, click) with corresponding default interactions.

Ensure any custom interactions based on these gestures make sense for the selected action or tool.

> ⓘ **Note**
>
> The interaction experience is dependent on the state of the Surface Dial menu. If the menu is active, it processes the input; otherwise, your app does.

Press and hold

This gesture activates and shows the Surface Dial menu, there is no app functionality associated with this gesture.

By default, the menu is displayed at the center of the user's screen. However, the user can grab it and move it anywhere they choose.

> ⓘ **Note**
>
> When the Surface Dial is placed on the screen of the Surface Studio, the menu is centered at the on-screen location of the Surface Dial.

Rotate

The Surface Dial is primarily designed to support rotation for interactions that involve smooth, incremental adjustments to analog values or controls.

The device can be rotated both clockwise and counter-clockwise, and can also provide haptic feedback to indicate discrete distances.

> ⓘ **Note**
>
> Haptic feedback can be disabled by the user in the **Windows Settings -> Devices -> Wheel** page.

UX guidance for custom interactions

Tools with continuous or high rotational sensitivity should disable haptic feedback

Haptic feedback matches the rotational sensitivity of the active tool. We recommend disabling haptic feedback for tools with continuous or high rotational sensitivity as the user experience can get uncomfortable.

Dominant hand should not affect rotation-based interactions

The Surface Dial cannot detect which hand is being used, but the user can set the writing (or dominant hand) in **Windows Settings -> Device -> Pen & Windows Ink**.

Locale should be considered for all rotation interactions

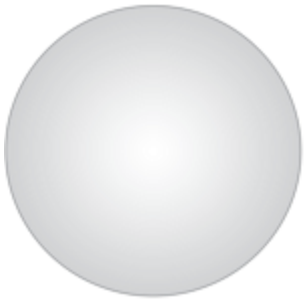
Maximize customer satisfaction by accomodating and adapting your interactions to locale and right-to-left layouts.

The built-in tools and commands on the Dial menu follow these guidelines for rotation-based interactions:

Left

Up

Out



Right

Down

In

Conceptual direction	Mapping to Surface Dial	Clockwise rotation	Counter-clockwise rotation
Horizontal	Left and right mapping based on the top of the Surface Dial	Right	Left
Vertical	Up and down mapping based on the left side of the Surface Dial	Down	Up
Z-axis	In (or nearer) mapped to up/right Out (or further) mapped to down/left	In	Out

Developer guidance

As the user rotates the device, RadialController.RotationChanged events are fired based on a delta (RadialControllerRotationChangedEventArgs.RotationDeltaInDegrees) relative to the direction of rotation. The sensitivity (or resolution) of the data can be set with the RadialController.RotationResolutionInDegrees property.

> **Note**
>
> By default, a rotational input event is delivered to a RadialController object only when the device is rotated a minimum of 10 degrees. Each input event causes the device to vibrate.

In general, we recommend disabling haptic feedback when the rotation resolution is set to less than 5 degrees. This provides a smoother experience for continuous interactions.

You can enable and disable haptic feedback for custom tools by setting the RadialController.UseAutomaticHapticFeedback property.

> **Note**
>
> You cannot override the haptic behavior for system tools such as the volume control. For these tools, haptic feedback can be disabled only by the user from the wheel settings page.

Here's an example of how to customize the resolution of the rotation data and enable or disable haptic feedback.

C#

```
private void MyController_ButtonClicked(RadialController sender,
  RadialControllerButtonClickedEventArgs args)
{
  ButtonToggle.IsOn = !ButtonToggle.IsOn;

  if(ButtonToggle.IsOn)
  {
    //high resolution mode
    RotationSlider.LargeChange = 1;
    myController.UseAutomaticHapticFeedback = false;
    myController.RotationResolutionInDegrees = 1;
  }
  else
  {
    //low resolution mode
    RotationSlider.LargeChange = 10;
    myController.UseAutomaticHapticFeedback = true;
    myController.RotationResolutionInDegrees = 10;
  }
}
```

Click

Clicking the Surface Dial is similar to clicking the left mouse button (the rotation state of the device has no effect on this action).

UX guidance

Do not map an action or command to this gesture if the user cannot easily recover from the result

Any action taken by your app based on the user clicking the Surface Dial must be reversible. Always enable the user to easily traverse the app back stack and restore a previous app state.

Binary operations such as mute/unmute or show/hide provide good user experiences with the click gesture.

Modal tools should not be enabled or disabled by clicking the Surface Dial

Some app/tool modes can conflict with, or disable, interactions that rely on rotation. Tools such as the ruler in the Windows Ink toolbar, should be toggled on or off through other UI affordances (the Ink Toolbar provides a built-in **ToggleButton** control).

For modal tools, map the active Surface Dial menu item to the target tool or to the previously selected menu item.

Developer guidance

When the Surface Dial is clicked, a **RadialController.ButtonClicked** event is fired. The **RadialControllerButtonClickedEventArgs** include a **Contact** property that contains the location and bounding area of the Surface Dial contact on the Surface Studio screen. If the Surface Dial is not in contact with the screen, this property is null.

On-screen

As described earlier, the Surface Dial can be used in conjunction with the Surface Studio to display the Surface Dial menu in a special on-screen mode.

When in this mode, you can integrate and customize your Dial interaction experiences with your apps even further. Examples of unique experiences only possible with the Surface Dial and Surface Studio include:

- Displaying contextual tools (such as a color palette) based on the position of the Surface Dial, which makes them easier to find and use
- Setting the active tool based on the UI the Surface Dial is placed on
- Magnifying a screen area based on location of the Surface Dial
- Unique game interactions based on screen location

UX guidance for on-screen interactions

Apps should respond when the Surface Dial is detected on-screen

Visual feedback helps indicate to users that your app has detected the device on the screen of the Surface Studio.

Adjust Surface Dial-related UI based on device location

The device (and the user's body) can occlude critical UI depending on where the user places it.

Adjust Surface Dial-related UI based on user interaction

In addition to hardware occlusion, a user's hand and arm can occlude part of the screen when using the device.

The occluded area depends on which hand is being used with the device. As the device is designed to be used primarily with the non-dominant hand, Surface Dial-related UI should adjust for the opposite hand specified by the user (**Windows Settings** > **Devices** > **Pen & Windows Ink** > **Choose which hand you write with** setting).

Interactions should respond to Surface Dial position rather than movement

The foot of the device is designed to stick to the screen rather than slide, as it is not a precision pointing device. Therefore, we expect it to be more common for users to lift and place the Surface Dial rather than drag it across the screen.

Use screen position to determine user intent

Setting the active tool based on UI context, such as proximity to a control, canvas, or window, can improve the user experience by reducing the steps required to perform a task.

Developer guidance

When the Surface Dial is placed onto the digitizer surface of the Surface Studio, a **RadialController.ScreenContactStarted** event is fired and the contact info (**RadialControllerScreenContactStartedEventArgs.Contact**) is provided to your app.

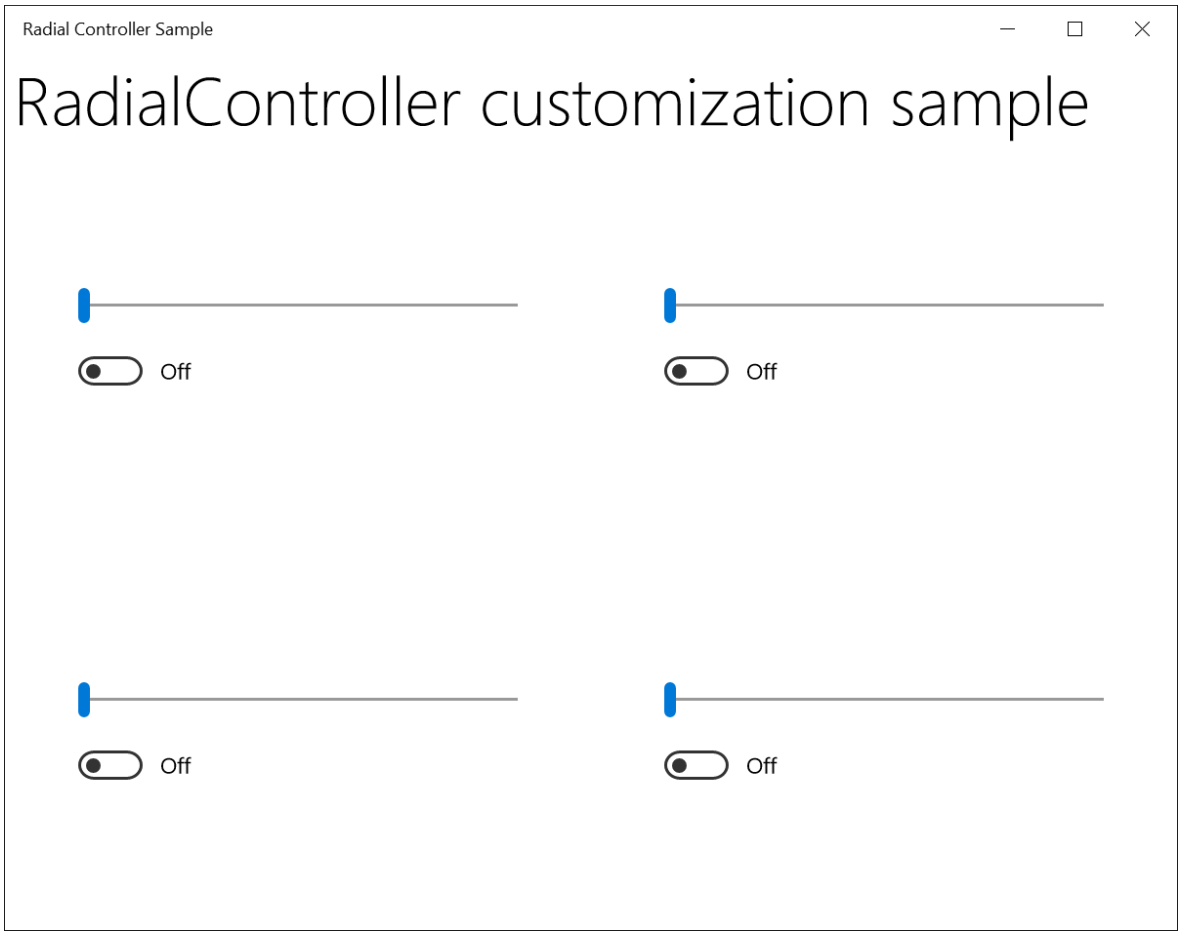
Similarly, if the Surface Dial is clicked when in contact with the digitizer surface of the Surface Studio, a **RadialController.ButtonClicked** event is fired and the contact info (**RadialControllerButtonClickedEventArgs.Contact**) is provided to your app.

The contact info (**RadialControllerScreenContact**) includes the X/Y coordinate of the center of the Surface Dial in the coordinate space of the app (**RadialControllerScreenContact.Position**), as well as the bounding rectangle (**RadialControllerScreenContact.Bounds**) in Device Independent Pixels (DIPs). This info is very useful for providing context to the active tool and providing device-related visual feedback to the user.

In the following example, we've created a basic app with four different sections, each of which includes one slider and one toggle. We then use the onscreen position of the

Surface Dial to dictate which set of sliders and toggles are controlled by the Surface Dial.

1. First, we declare our UI (four sections, each with a slider and toggle button) in XAML.



The sample app UI

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
  <Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="*"/>
  </Grid.RowDefinitions>
  <StackPanel x:Name="HeaderPanel"
        Orientation="Horizontal"
        Grid.Row="0">
    <TextBlock x:Name="Header"
      Text="RadialController customization sample"
      VerticalAlignment="Center"
      Style="{ThemeResource HeaderTextBlockStyle}"
      Margin="10,0,0,0" />
  </StackPanel>
  <Grid Grid.Row="1" x:Name="RootGrid">
    <Grid.RowDefinitions>
      <RowDefinition Height="*"/>
      <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
```

```
      <ColumnDefinition Width="*"/>
      <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <Grid x:Name="Grid0"
      Grid.Row="0"
      Grid.Column="0">
      <StackPanel Orientation="Vertical"
        VerticalAlignment="Center"
        HorizontalAlignment="Center">
        <!-- Slider for rotational input -->
        <Slider x:Name="RotationSlider0"
          Width="300"
          HorizontalAlignment="Left"/>
        <!-- Switch for button input -->
        <ToggleSwitch x:Name="ButtonToggle0"
            HorizontalAlignment="Left"/>
      </StackPanel>
    </Grid>
    <Grid x:Name="Grid1"
      Grid.Row="0"
      Grid.Column="1">
      <StackPanel Orientation="Vertical"
        VerticalAlignment="Center"
        HorizontalAlignment="Center">
        <!-- Slider for rotational input -->
        <Slider x:Name="RotationSlider1"
          Width="300"
          HorizontalAlignment="Left"/>
        <!-- Switch for button input -->
        <ToggleSwitch x:Name="ButtonToggle1"
            HorizontalAlignment="Left"/>
      </StackPanel>
    </Grid>
    <Grid x:Name="Grid2"
      Grid.Row="1"
      Grid.Column="0">
      <StackPanel Orientation="Vertical"
        VerticalAlignment="Center"
        HorizontalAlignment="Center">
        <!-- Slider for rotational input -->
        <Slider x:Name="RotationSlider2"
          Width="300"
          HorizontalAlignment="Left"/>
        <!-- Switch for button input -->
        <ToggleSwitch x:Name="ButtonToggle2"
            HorizontalAlignment="Left"/>
      </StackPanel>
    </Grid>
    <Grid x:Name="Grid3"
      Grid.Row="1"
      Grid.Column="1">
      <StackPanel Orientation="Vertical"
        VerticalAlignment="Center"
        HorizontalAlignment="Center">
        <!-- Slider for rotational input -->
```

```
        <Slider x:Name="RotationSlider3"
          Width="300"
          HorizontalAlignment="Left"/>
        <!-- Switch for button input -->
        <ToggleSwitch x:Name="ButtonToggle3"
            HorizontalAlignment="Left"/>
      </StackPanel>
    </Grid>
  </Grid>
</Grid>
```

2. Here's the code-behind with handlers defined for Surface Dial screen position.

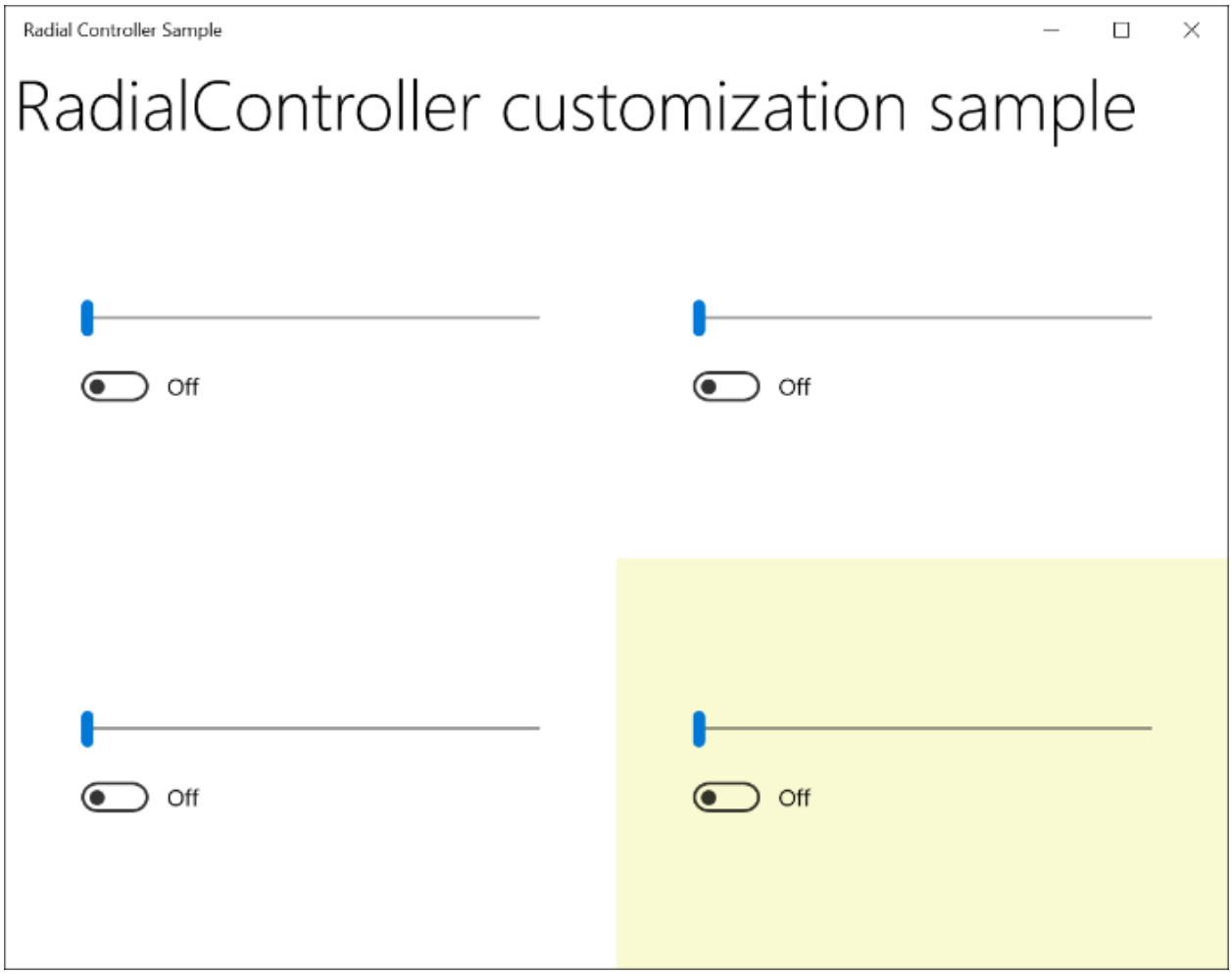
C#

```
Slider ActiveSlider;
ToggleSwitch ActiveSwitch;
Grid ActiveGrid;

public MainPage()
{
  ...

  myController.ScreenContactStarted +=
    MyController_ScreenContactStarted;
  myController.ScreenContactContinued +=
    MyController_ScreenContactContinued;
  myController.ScreenContactEnded +=
    MyController_ScreenContactEnded;
  myController.ControlLost += MyController_ControlLost;

  //Set initial grid for Surface Dial input.
  ActiveGrid = Grid0;
  ActiveSlider = RotationSlider0;
  ActiveSwitch = ButtonToggle0;
}

private void MyController_ScreenContactStarted(RadialController sender,
  RadialControllerScreenContactStartedEventArgs args)
{
  //find grid at contact location, update visuals, selection
  ActivateGridAtLocation(args.Contact.Position);
}

private void MyController_ScreenContactContinued(RadialController
sender,
  RadialControllerScreenContactContinuedEventArgs args)
{
  //if a new grid is under contact location, update visuals, selection
  if (!VisualTreeHelper.FindElementsInHostCoordinates(
    args.Contact.Position, RootGrid).Contains(ActiveGrid))
  {
    ActiveGrid.Background = new
```

```
      SolidColorBrush(Windows.UI.Colors.White);
    ActivateGridAtLocation(args.Contact.Position);
  }
}

private void MyController_ScreenContactEnded(RadialController sender,
object args)
{
  //return grid color to normal when contact leaves screen
  ActiveGrid.Background = new
  SolidColorBrush(Windows.UI.Colors.White);
}

private void MyController_ControlLost(RadialController sender, object
args)
{
  //return grid color to normal when focus lost
  ActiveGrid.Background = new
    SolidColorBrush(Windows.UI.Colors.White);
}

private void ActivateGridAtLocation(Point Location)
{
  var elementsAtContactLocation =
    VisualTreeHelper.FindElementsInHostCoordinates(Location,
      RootGrid);

  foreach (UIElement element in elementsAtContactLocation)
  {
    if (element as Grid == Grid0)
    {
      ActiveSlider = RotationSlider0;
      ActiveSwitch = ButtonToggle0;
      ActiveGrid = Grid0;
      ActiveGrid.Background = new SolidColorBrush(
        Windows.UI.Colors.LightGoldenrodYellow);
      return;
    }
    else if (element as Grid == Grid1)
    {
      ActiveSlider = RotationSlider1;
      ActiveSwitch = ButtonToggle1;
      ActiveGrid = Grid1;
      ActiveGrid.Background = new SolidColorBrush(
        Windows.UI.Colors.LightGoldenrodYellow);
      return;
    }
    else if (element as Grid == Grid2)
    {
      ActiveSlider = RotationSlider2;
      ActiveSwitch = ButtonToggle2;
      ActiveGrid = Grid2;
      ActiveGrid.Background = new SolidColorBrush(
        Windows.UI.Colors.LightGoldenrodYellow);
      return;
```

```
    }
    else if (element as Grid == Grid3)
    {
      ActiveSlider = RotationSlider3;
      ActiveSwitch = ButtonToggle3;
      ActiveGrid = Grid3;
      ActiveGrid.Background = new SolidColorBrush(
        Windows.UI.Colors.LightGoldenrodYellow);
      return;
    }
  }
}
```

When we run the app, we use the Surface Dial to interact with it. First, we place the device on the Surface Studio screen, which the app detects and associates with the lower right section (see image). We then press and hold the Surface Dial to open the menu and select our custom tool. Once the custom tool is activated, the slider control can be adjusted by rotating the Surface Dial and the switch can be toggled by clicking the Surface Dial.



The sample app UI activated using the Surface Dial custom tool

Summary

This topic provides an overview of the Surface Dial input device with UX and developer guidance on how to customize the user experience for off-screen scenarios as well as on-screen scenarios when used with Surface Studio.

Please send your questions, suggestions, and feedback to radialcontroller@microsoft.com.

Related articles

Tutorial: Support the Surface Dial (and other wheel devices) in your Windows app

API reference

- **RadialController** class
- **RadialControllerButtonClickedEventArgs** class
- **RadialControllerConfiguration** class
- **RadialControllerControlAcquiredEventArgs** class
- **RadialControllerMenu** class
- **RadialControllerMenuItem** class
- **RadialControllerRotationChangedEventArgs** class
- **RadialControllerScreenContact** class
- **RadialControllerScreenContactContinuedEventArgs** class
- **RadialControllerScreenContactStartedEventArgs** class
- **RadialControllerMenuKnownIcon** enum
- **RadialControllerSystemMenuItemKind** enum

Samples

Topic samples

RadialController customization

Other samples

Coloring Book sample

Get Started Tutorial: Support the Surface Dial (and other wheel devices) in your Windows app

Universal Windows Platform samples (C# and C++)

Windows desktop sample

Tutorial: Support the Surface Dial (and other wheel devices) in your Windows app

Article • 10/06/2022

Surface Dial with Surface Studio and Surface Pen (available for purchase at the Microsoft Store).

This tutorial steps through how to customize the user interaction experiences supported by wheel devices such as the Surface Dial. We use snippets from a sample app, which you can download from GitHub (see Sample code), to demonstrate the various features and associated **RadialController** APIs discussed in each step.

We focus on the following:

- Specifying which built-in tools are displayed on the **RadialController** menu
- Adding a custom tool to the menu
- Controlling haptic feedback
- Customizing click interactions
- Customizing rotation interactions

For more about implementing these and other features, see Surface Dial interactions in Windows apps.

Introduction

The Surface Dial is a secondary input device that helps users to be more productive when used together with a primary input device such as pen, touch, or mouse. As a secondary input device, the Dial is typically used with the non-dominant hand to provide access both to system commands and to other, more contextual, tools and functionality.

The Dial supports three basic gestures:

- Press and hold to display the built-in menu of commands.
- Rotate to highlight a menu item (if the menu is active) or to modify the current action in the app (if the menu is not active).
- Click to select the highlighted menu item (if the menu is active) or to invoke a command in the app (if the menu is not active).

Prerequisites

- A computer (or a virtual machine) running Windows 10 Creators Update or later
- Visual Studio 2019
- Windows 10 SDK (10.0.15063.0)
- A wheel device (only the Surface Dial at this time)
- If you're new to Windows app development with Visual Studio, have a look through these topics before you start this tutorial:
  - Install tools for the Windows App SDK
  - Create a "Hello, world" app (XAML)

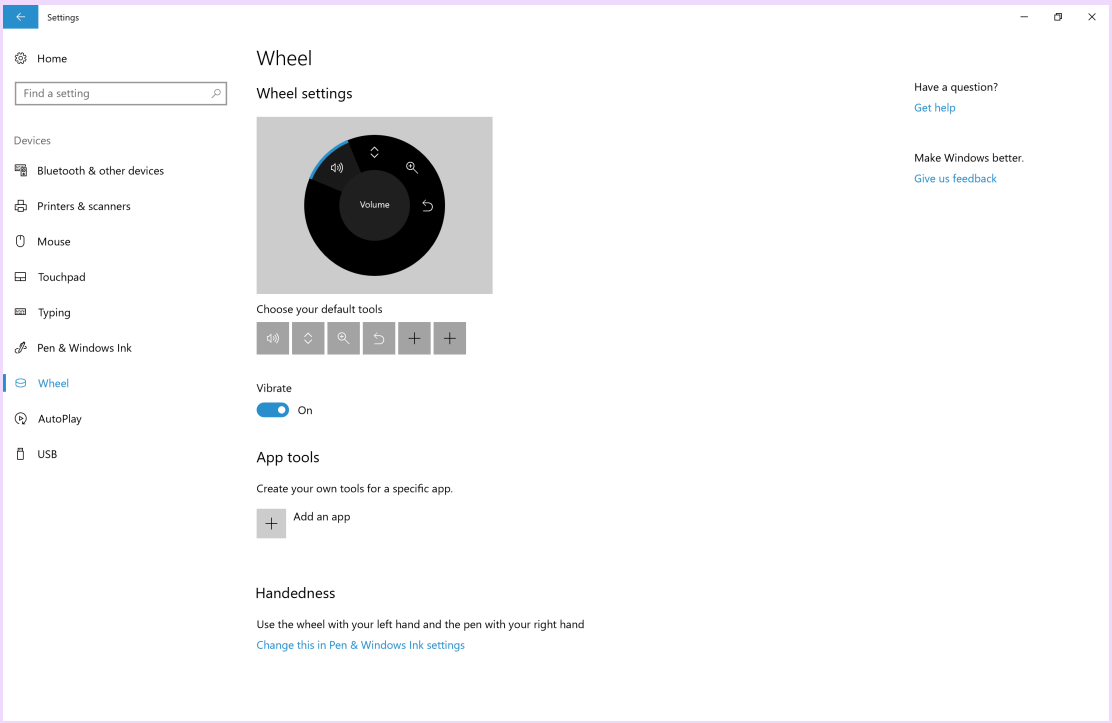
Set up your devices

1. Make sure your Windows device is on.
2. Go to **Start**, select **Settings** > **Devices** > **Bluetooth & other devices**, and then turn **Bluetooth** on.
3. Remove the bottom of the Surface Dial to open the battery compartment, and make sure that there are two AAA batteries inside.
4. If the battery tab is present on the underside of the Dial, remove it.
5. Press and hold the small, inset button next to the batteries until the Bluetooth light flashes.
6. Go back to your Windows device and select **Add Bluetooth or other device**.
7. In the **Add a device** dialog, select **Bluetooth** > **Surface Dial**. Your Surface Dial should now connect and be added to the list of devices under **Mouse, keyboard, & pen** on the **Bluetooth & other devices** settings page.
8. Test the Dial by pressing and holding it down for a few seconds to display the built-in menu.
9. If the menu isn't displayed on your screen (the Dial should also vibrate), go back to the Bluetooth settings, remove the device, and try connecting the device again.

> **Note**

Wheel devices can be configured through the **Wheel** settings:

1. On the **Start** menu, select **Settings**.
2. Select **Devices** > **Wheel**.



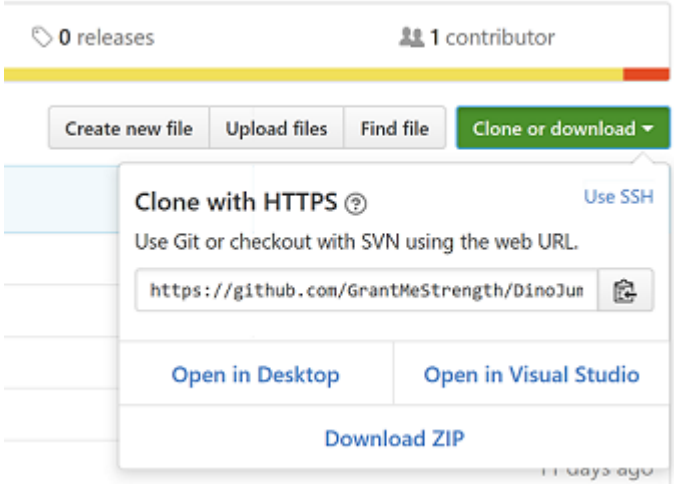
Now you're ready to start this tutorial.

Sample code

Throughout this tutorial, we use a sample app to demonstrate the concepts and functionality discussed.

Download this Visual Studio sample and source code from GitHub at windows-appsample-get-started-radialcontroller sample:

1. Select the green **Clone or download** button.



2. If you have a GitHub account, you can clone the repo to your local machine by choosing **Open in Visual Studio**.
3. If you don't have a GitHub account, or you just want a local copy of the project, choose **Download ZIP** (you'll have to check back regularly to download the latest updates).

> **Important**
>
> Most of the code in the sample is commented out. As we go through each step in this topic, you'll be asked to uncomment various sections of the code. In Visual Studio, just highlight the lines of code, and press CTRL-K and then CTRL-U.

Components that support wheel functionality

These objects provide the bulk of the wheel device experience for Windows apps.

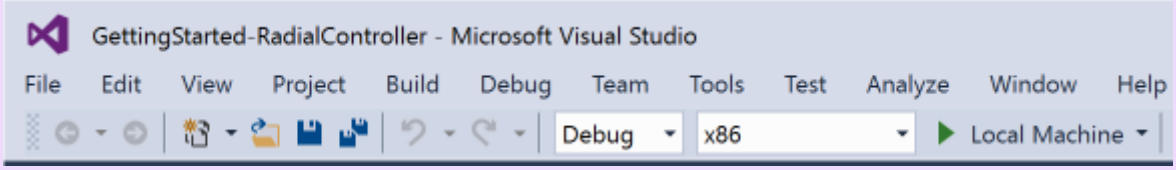
Component	Description
RadialController class and related	Represents a wheel input device or accessory such as the Surface Dial.
IRadialControllerConfigurationInterop / IRadialControllerInterop We do not cover this functionality here, for more information, see the Windows desktop sample.	Enables interoperability with a Windows app.

Step 1: Run the sample

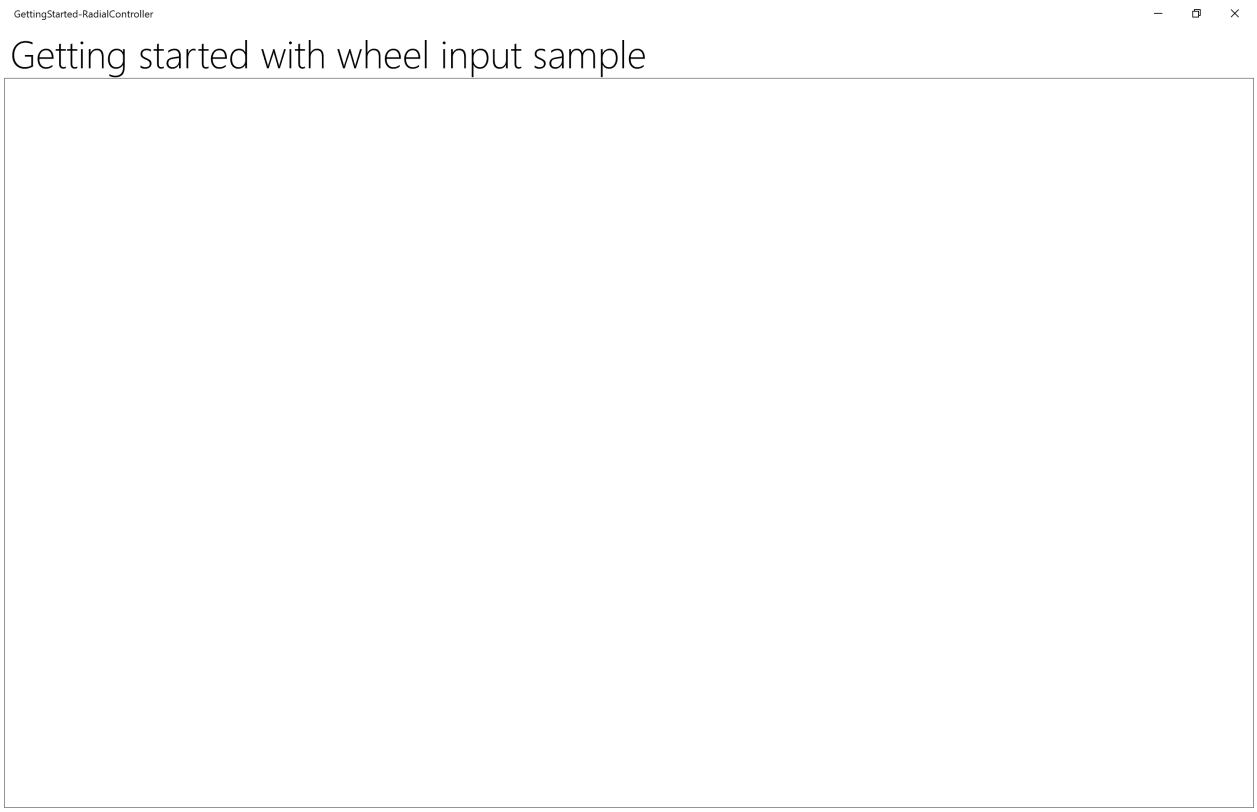
After you've downloaded the RadialController sample app, verify that it runs:

1. Open the sample project in Visual Studio .
2. Set the **Solution Platforms** dropdown to a non-Arm selection.
3. Press F5 to compile, deploy, and run.

> **Note**
>
> Alternatively, you can select **Debug** > **Start debugging** menu item, or select the **Local Machine** Run button shown here:



The app window opens, and after a splash screen appears for a few seconds, you'll see this initial screen.



Okay, we now have the basic Windows app that we'll use throughout the rest of this tutorial. In the following steps, we add our **RadialController** functionality.

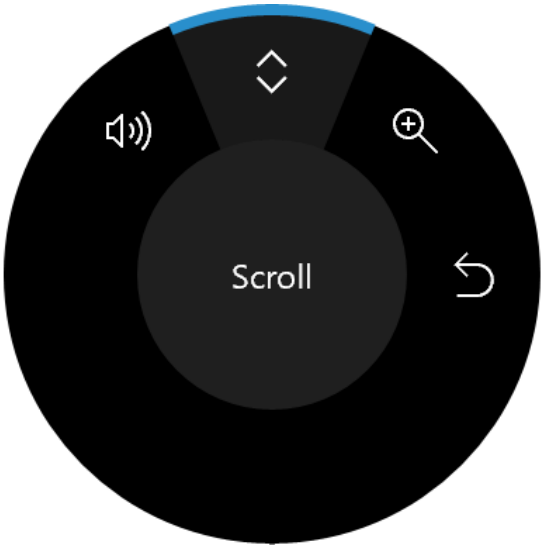
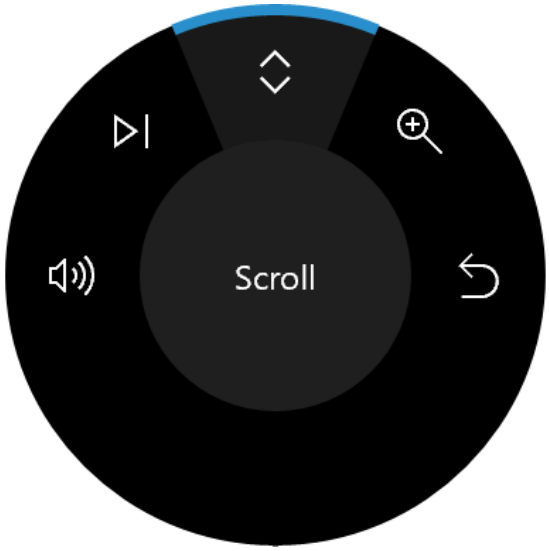
Step 2: Basic RadialController functionality

With the app running and in the foreground, press and hold the Surface Dial to display the **RadialController** menu.

We haven't done any customization for our app yet, so the menu contains a default set of contextual tools.

These images show two variations of the default menu. (There are many others, including just basic system tools when the Windows Desktop is active and no apps are in the foreground, additional inking tools when an InkToolbar is present, and mapping tools when you're using the Maps app.

RadialController menu (default)	**RadialController menu (default with media playing)**

RadialController menu (default)	RadialController menu (default with media playing)
	

Now we'll start with some basic customization.

Step 3: Add controls for wheel input

First, let's add the UI for our app:

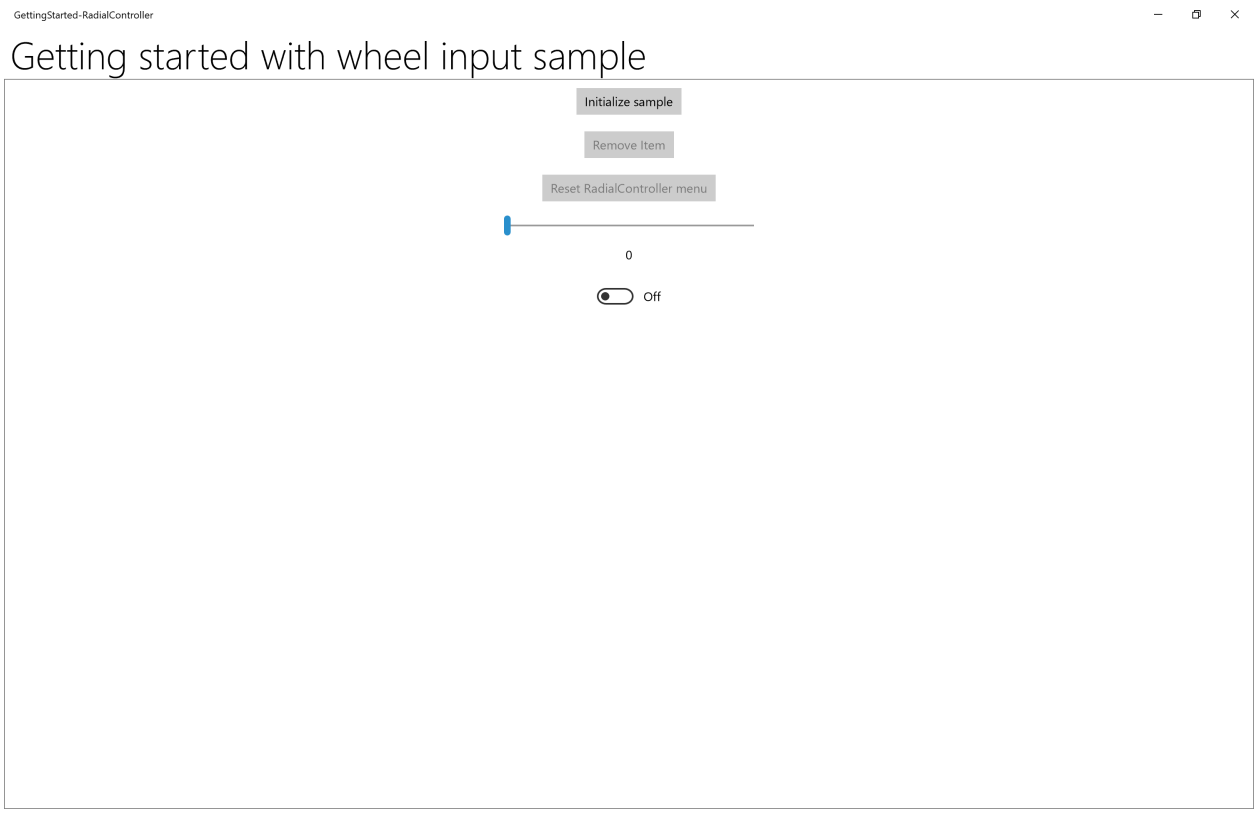
1. Open the MainPage_Basic.xaml file.
2. Find the code marked with the title of this step ("<!-- Step 3: Add controls for wheel input -->").
3. Uncomment the following lines.

XAML

```
<Button x:Name="InitializeSampleButton"
        HorizontalAlignment="Center"
        Margin="10"
        Content="Initialize sample" />
<ToggleButton x:Name="AddRemoveToggleButton"
                HorizontalAlignment="Center"
                Margin="10"
                Content="Remove Item"
                IsChecked="True"
                IsEnabled="False"/>
<Button x:Name="ResetControllerButton"
        HorizontalAlignment="Center"
        Margin="10"
        Content="Reset RadialController menu"
        IsEnabled="False"/>
<Slider x:Name="RotationSlider" Minimum="0" Maximum="10"
        Width="300"
```

```
        HorizontalAlignment="Center"/>
<TextBlock Text="{Binding ElementName=RotationSlider, Mode=OneWay,
Path=Value}"
           Margin="0,0,0,20"
           HorizontalAlignment="Center"/>
<ToggleSwitch x:Name="ClickToggle"
              MinWidth="0"
              Margin="0,0,0,20"
              HorizontalAlignment="center"/>
```

At this point, only the **Initialize sample** button, slider, and toggle switch are enabled. The other buttons are used in later steps to add and remove **RadialController** menu items that provide access to the slider and toggle switch.



Step 4: Customize the basic RadialController menu

Now let's add the code required to enable **RadialController** access to our controls.

1. Open the MainPage_Basic.xaml.cs file.
2. Find the code marked with the title of this step ("// Step 4: Basic RadialController menu customization").
3. Uncomment the following lines:

    - The Windows.UI.Input and Windows.Storage.Streams type references are used for functionality in subsequent steps:

```
C#

// Using directives for RadialController functionality.
using Windows.UI.Input;
```

- These global objects (RadialController, RadialControllerConfiguration, RadialControllerMenuItem) are used throughout our app.

```
C#

private RadialController radialController;
private RadialControllerConfiguration radialControllerConfig;
private RadialControllerMenuItem radialControllerMenuItem;
```

- Here, we specify the **Click** handler for the button that enables our controls and initializes our custom **RadialController** menu item.

```
C#

InitializeSampleButton.Click += (sender, args) =>
{ InitializeSample(sender, args); };
```

- Next, we initialize our RadialController object and set up handlers for the RotationChanged and ButtonClicked events.

```
C#

// Set up the app UI and RadialController.
private void InitializeSample(object sender, RoutedEventArgs e)
{
    ResetControllerButton.IsEnabled = true;
    AddRemoveToggleButton.IsEnabled = true;

    ResetControllerButton.Click += (resetsender, args) =>
    { ResetController(resetsender, args); };
    AddRemoveToggleButton.Click += (togglesender, args) =>
    { AddRemoveItem(togglesender, args); };

    InitializeController(sender, e);
}
```

- Here, we initialize our custom RadialController menu item. We use CreateForCurrentView to get a reference to our RadialController object, we set the rotation sensitivity to "1" by using the RotationResolutionInDegrees property, we then create our RadialControllerMenuItem by using CreateFromFontGlyph, we add the menu item to the **RadialController** menu

item collection, and finally, we use SetDefaultMenuItems to clear the default menu items and leave only our custom tool.

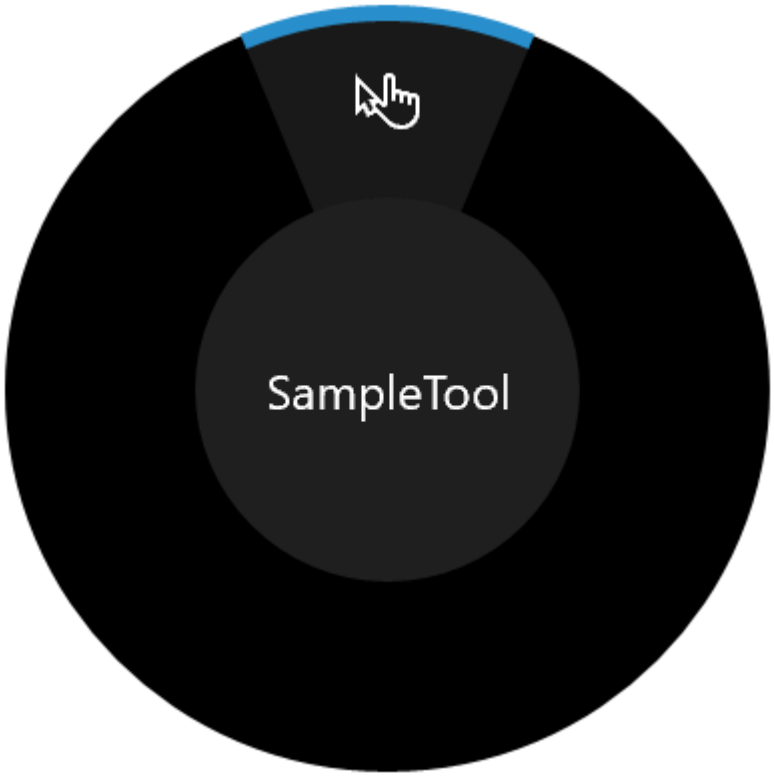
C#

```
// Configure RadialController menu and custom tool.
private void InitializeController(object sender, RoutedEventArgs
args)
{
    // Create a reference to the RadialController.
    radialController = RadialController.CreateForCurrentView();
    // Set rotation resolution to 1 degree of sensitivity.
    radialController.RotationResolutionInDegrees = 1;

    // Create the custom menu items.
    // Here, we use a font glyph for our custom tool.
    radialControllerMenuItem =
        RadialControllerMenuItem.CreateFromFontGlyph("SampleTool",
"\xE1E3", "Segoe MDL2 Assets");

    // Add the item to the RadialController menu.
    radialController.Menu.Items.Add(radialControllerMenuItem);

    // Remove built-in tools to declutter the menu.
    // NOTE: The Surface Dial menu must have at least one menu
item.
    // If all built-in tools are removed before you add a custom
    // tool, the default tools are restored and your tool is
appended
    // to the default collection.
    radialControllerConfig =
        RadialControllerConfiguration.GetForCurrentView();
    radialControllerConfig.SetDefaultMenuItems(
        new RadialControllerSystemMenuItemKind[] { });

    // Declare input handlers for the RadialController.
    // NOTE: These events are only fired when a custom tool is
active.
    radialController.ButtonClicked += (clicksender, clickargs) =>
    { RadialController_ButtonClicked(clicksender, clickargs); };
    radialController.RotationChanged += (rotationsender,
rotationargs) =>
    { RadialController_RotationChanged(rotationsender,
rotationargs); };
}

// Connect wheel device rotation to slider control.
private void RadialController_RotationChanged(
    object sender, RadialControllerRotationChangedEventArgs args)
{
    if (RotationSlider.Value + args.RotationDeltaInDegrees >=
RotationSlider.Maximum)
    {
        RotationSlider.Value = RotationSlider.Maximum;
```

```
    }
    else if (RotationSlider.Value + args.RotationDeltaInDegrees <
RotationSlider.Minimum)
    {
        RotationSlider.Value = RotationSlider.Minimum;
    }
    else
    {
        RotationSlider.Value += args.RotationDeltaInDegrees;
    }
}

// Connect wheel device click to toggle switch control.
private void RadialController_ButtonClicked(
    object sender, RadialControllerButtonClickedEventArgs args)
{
    ClickToggle.IsOn = !ClickToggle.IsOn;
}
```

4. Now, run the app again.
5. Select the **Initialize radial controller** button.
6. With the app in the foreground, press and hold the Surface Dial to display the menu. Notice that all default tools have been removed (by using the **RadialControllerConfiguration.SetDefaultMenuItems** method), leaving only the custom tool. Here's the menu with our custom tool.

RadialController menu (custom)



7. Select the custom tool and try out the interactions now supported through the Surface Dial:

    - A rotate action moves the slider.
    - A click sets the toggle to on or off.

Ok, let's hook up those buttons.

Step 5: Configure menu at runtime

In this step, we hook up the **Add/Remove item** and **Reset RadialController menu** buttons to show how you can dynamically customize the menu.

1. Open the MainPage_Basic.xaml.cs file.
2. Find the code marked with the title of this step ("// Step 5: Configure menu at runtime").
3. Uncomment the code in the following methods and run the app again, but don't select any buttons (save that for the next step).

```
C#
```

```
// Add or remove the custom tool.
private void AddRemoveItem(object sender, RoutedEventArgs args)
{
    if (AddRemoveToggleButton?.IsChecked == true)
    {
        AddRemoveToggleButton.Content = "Remove item";
        if
(!radialController.Menu.Items.Contains(radialControllerMenuItem))
        {
            radialController.Menu.Items.Add(radialControllerMenuItem);
        }
    }
    else if (AddRemoveToggleButton?.IsChecked == false)
    {
        AddRemoveToggleButton.Content = "Add item";
        if
(radialController.Menu.Items.Contains(radialControllerMenuItem))
        {

radialController.Menu.Items.Remove(radialControllerMenuItem);
            // Attempts to select and activate the previously selected
tool.
            // NOTE: Does not differentiate between built-in and custom
tools.

radialController.Menu.TrySelectPreviouslySelectedMenuItem();
```

```
        }
    }
}

// Reset the RadialController to initial state.
private void ResetController(object sender, RoutedEventArgs arg)
{
    if
(!radialController.Menu.Items.Contains(radialControllerMenuItem))
    {
        radialController.Menu.Items.Add(radialControllerMenuItem);
    }
    AddRemoveToggleButton.Content = "Remove item";
    AddRemoveToggleButton.IsChecked = true;
    radialControllerConfig.SetDefaultMenuItems(
        new RadialControllerSystemMenuItemKind[] { });
}
```

4. Select the **Remove item** button and then press and hold the Dial to display the menu again.

   Notice that the menu now contains the default collection of tools. Recall that, in Step 3, while setting up our custom menu, we removed all the default tools and added just our custom tool. We also noted that, when the menu is set to an empty collection, the default items for the current context are reinstated. (We added our custom tool before removing the default tools.)

5. Select the **Add item** button and then press and hold the Dial.

   Notice that the menu now contains both the default collection of tools and our custom tool.

6. Select the **Reset RadialController menu** button and then press and hold the Dial.

   Notice that the menu is back to its original state.

Step 6: Customize the device haptics

The Surface Dial, and other wheel devices, can provide users with haptic feedback corresponding to the current interaction (based on either click or rotation).

In this step, we show how you can customize haptic feedback by associating our slider and toggle switch controls and using them to dynamically specify haptic feedback behavior. For this example, the toggle switch must be set to on for feedback to be enabled while the slider value specifies how often the click feedback is repeated.

> **Note**
>
> Haptic feedback can be disabled by the user in the **Settings** > **Devices** > **Wheel** page.

1. Open the App.xaml.cs file.
2. Find the code marked with the title of this step ("Step 6: Customize the device haptics").
3. Comment the first and third lines ("MainPage_Basic" and "MainPage") and uncomment the second ("MainPage_Haptics").

```C#
rootFrame.Navigate(typeof(MainPage_Basic), e.Arguments);
rootFrame.Navigate(typeof(MainPage_Haptics), e.Arguments);
rootFrame.Navigate(typeof(MainPage), e.Arguments);
```

4. Open the MainPage_Haptics.xaml file.
5. Find the code marked with the title of this step ("<!-- Step 6: Customize the device haptics -->").
6. Uncomment the following lines. (This UI code simply indicates what haptics features are supported by the current device.)

```XAML
<StackPanel x:Name="HapticsStack"
            Orientation="Vertical"
            HorizontalAlignment="Center"
            BorderBrush="Gray"
            BorderThickness="1">
    <TextBlock Padding="10"
               Text="Supported haptics properties:" />
    <CheckBox x:Name="CBDefault"
              Content="Default"
              Padding="10"
              IsEnabled="False"
              IsChecked="True" />
    <CheckBox x:Name="CBIntensity"
              Content="Intensity"
              Padding="10"
              IsEnabled="False"
              IsThreeState="True"
              IsChecked="{x:Null}" />
    <CheckBox x:Name="CBPlayCount"
```

```
                Content="Play count"
                Padding="10"
                IsEnabled="False"
                IsThreeState="True"
                IsChecked="{x:Null}" />
    <CheckBox x:Name="CBPlayDuration"
                Content="Play duration"
                Padding="10"
                IsEnabled="False"
                IsThreeState="True"
                IsChecked="{x:Null}" />
    <CheckBox x:Name="CBReplayPauseInterval"
                Content="Replay/pause interval"
                Padding="10"
                IsEnabled="False"
                IsThreeState="True"
                IsChecked="{x:Null}" />
    <CheckBox x:Name="CBBuzzContinuous"
                Content="Buzz continuous"
                Padding="10"
                IsEnabled="False"
                IsThreeState="True"
                IsChecked="{x:Null}" />
    <CheckBox x:Name="CBClick"
                Content="Click"
                Padding="10"
                IsEnabled="False"
                IsThreeState="True"
                IsChecked="{x:Null}" />
    <CheckBox x:Name="CBPress"
                Content="Press"
                Padding="10"
                IsEnabled="False"
                IsThreeState="True"
                IsChecked="{x:Null}" />
    <CheckBox x:Name="CBRelease"
                Content="Release"
                Padding="10"
                IsEnabled="False"
                IsThreeState="True"
                IsChecked="{x:Null}" />
    <CheckBox x:Name="CBRumbleContinuous"
                Content="Rumble continuous"
                Padding="10"
                IsEnabled="False"
                IsThreeState="True"
                IsChecked="{x:Null}" />
</StackPanel>
```

7. Open the MainPage_Haptics.xaml.cs file

8. Find the code marked with the title of this step ("Step 6: Haptics customization")

9. Uncomment the following lines:

- The Windows.Devices.Haptics type reference is used for functionality in subsequent steps.

```
C#

using Windows.Devices.Haptics;
```

- Here, we specify the handler for the ControlAcquired event that is triggered when our custom **RadialController** menu item is selected.

```
C#

radialController.ControlAcquired += (rc_sender, args) =>
{ RadialController_ControlAcquired(rc_sender, args); };
```

- Next, we define the ControlAcquired handler, where we disable default haptic feedback and initialize our haptics UI.

```
C#

private void RadialController_ControlAcquired(
    RadialController rc_sender,
    RadialControllerControlAcquiredEventArgs args)
{
    // Turn off default haptic feedback.
    radialController.UseAutomaticHapticFeedback = false;

    SimpleHapticsController hapticsController =
        args.SimpleHapticsController;

    // Enumerate haptic support.
    IReadOnlyCollection<SimpleHapticsControllerFeedback>
supportedFeedback =
        hapticsController.SupportedFeedback;

    foreach (SimpleHapticsControllerFeedback feedback in
supportedFeedback)
    {
        if (feedback.Waveform ==
KnownSimpleHapticsControllerWaveforms.BuzzContinuous)
        {
            CBBuzzContinuous.IsEnabled = true;
            CBBuzzContinuous.IsChecked = true;
        }
        else if (feedback.Waveform ==
KnownSimpleHapticsControllerWaveforms.Click)
        {
            CBClick.IsEnabled = true;
```

```
            CBClick.IsChecked = true;
        }
        else if (feedback.Waveform ==
KnownSimpleHapticsControllerWaveforms.Press)
        {
            CBPress.IsEnabled = true;
            CBPress.IsChecked = true;
        }
        else if (feedback.Waveform ==
KnownSimpleHapticsControllerWaveforms.Release)
        {
            CBRelease.IsEnabled = true;
            CBRelease.IsChecked = true;
        }
        else if (feedback.Waveform ==
KnownSimpleHapticsControllerWaveforms.RumbleContinuous)
        {
            CBRumbleContinuous.IsEnabled = true;
            CBRumbleContinuous.IsChecked = true;
        }
    }

    if (hapticsController?.IsIntensitySupported == true)
    {
        CBIntensity.IsEnabled = true;
        CBIntensity.IsChecked = true;
    }
    if (hapticsController?.IsPlayCountSupported == true)
    {
        CBPlayCount.IsEnabled = true;
        CBPlayCount.IsChecked = true;
    }
    if (hapticsController?.IsPlayDurationSupported == true)
    {
        CBPlayDuration.IsEnabled = true;
        CBPlayDuration.IsChecked = true;
    }
    if (hapticsController?.IsReplayPauseIntervalSupported == true)
    {
        CBReplayPauseInterval.IsEnabled = true;
        CBReplayPauseInterval.IsChecked = true;
    }
}
```

- In our RotationChanged and ButtonClicked event handlers, we connect the corresponding slider and toggle button controls to our custom haptics.

```
C#

// Connect wheel device rotation to slider control.
private void RadialController_RotationChanged(
    object sender, RadialControllerRotationChangedEventArgs args)
{
```

```
    ...
    if (ClickToggle.IsOn &&
        (RotationSlider.Value > RotationSlider.Minimum) &&
        (RotationSlider.Value < RotationSlider.Maximum))
    {
        SimpleHapticsControllerFeedback waveform =
            FindWaveform(args.SimpleHapticsController,
            KnownSimpleHapticsControllerWaveforms.BuzzContinuous);
        if (waveform != null)
        {

args.SimpleHapticsController.SendHapticFeedback(waveform);
        }
    }
}

private void RadialController_ButtonClicked(
    object sender, RadialControllerButtonClickedEventArgs args)
{
    ...

    if (RotationSlider?.Value > 0)
    {
        SimpleHapticsControllerFeedback waveform =
            FindWaveform(args.SimpleHapticsController,
            KnownSimpleHapticsControllerWaveforms.Click);

        if (waveform != null)
        {

args.SimpleHapticsController.SendHapticFeedbackForPlayCount(
                waveform, 1.0,
                (int)RotationSlider.Value,
                TimeSpan.Parse("1"));
        }
    }
}
```

- Finally, we get the requested **Waveform** (if supported) for the haptic feedback.

C#

```
// Get the requested waveform.
private SimpleHapticsControllerFeedback FindWaveform(
    SimpleHapticsController hapticsController,
    ushort waveform)
{
    foreach (var hapticInfo in
hapticsController.SupportedFeedback)
    {
        if (hapticInfo.Waveform == waveform)
        {
```

```
                return hapticInfo;
            }
        }
        return null;
    }
```

Now run the app again to try out the custom haptics by changing the slider value and toggle-switch state.

Step 7: Define on-screen interactions for Surface Studio and similar devices

Paired with the Surface Studio, the Surface Dial can provide an even more distinctive user experience.

In addition to the default press and hold menu experience described, the Surface Dial can also be placed directly on the screen of the Surface Studio. This enables a special "on-screen" menu.

By detecting both the contact location and bounds of the Surface Dial, the system handles occlusion by the device and displays a larger version of the menu that wraps around the outside of the Dial. This same info can also be used by your app to adapt the UI for both the presence of the device and its anticipated usage, such as the placement of the user's hand and arm.

The sample that accompanies this tutorial includes a slightly more complex example that demonstrates some of these capabilities.

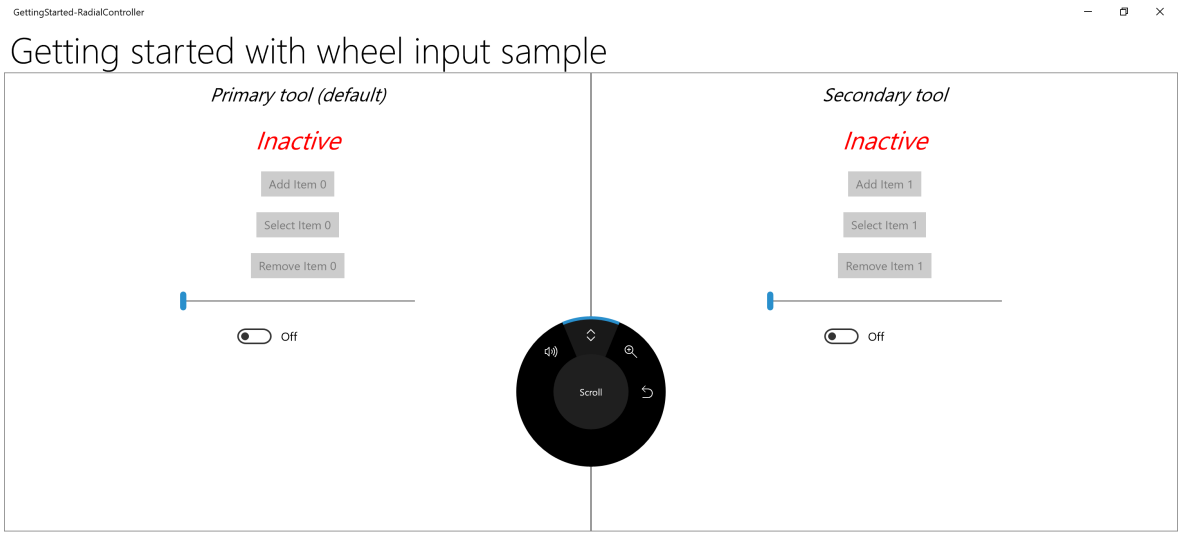
To see this in action (you'll need a Surface Studio):

1. Download the sample on a Surface Studio device (with Visual Studio installed)
2. Open the sample in Visual Studio
3. Open the App.xaml.cs file
4. Find the code marked with the title of this step ("Step 7: Define on-screen interactions for Surface Studio and similar devices")
5. Comment the first and second lines ("MainPage_Basic" and "MainPage_Haptics") and uncomment the third ("MainPage")

```
C#
```

```
rootFrame.Navigate(typeof(MainPage_Basic), e.Arguments);
rootFrame.Navigate(typeof(MainPage_Haptics), e.Arguments);
rootFrame.Navigate(typeof(MainPage), e.Arguments);
```

6. Run the app and place the Surface Dial in each of the two control regions, alternating between them.



Summary

Congratulations, you've completed the *Get Started Tutorial: Support the Surface Dial (and other wheel devices) in your Windows app*! We showed you the basic code required for supporting a wheel device in your Windows apps, and how to provide some of the richer user experiences supported by the **RadialController** APIs.

Related articles

Surface Dial interactions

API reference

- **RadialController** class
- **RadialControllerButtonClickedEventArgs** class
- **RadialControllerConfiguration** class
- **RadialControllerControlAcquiredEventArgs** class
- **RadialControllerMenu** class
- **RadialControllerMenuItem** class
- **RadialControllerRotationChangedEventArgs** class
- **RadialControllerScreenContact** class
- **RadialControllerScreenContactContinuedEventArgs** class

- **RadialControllerScreenContactStartedEventArgs** class
- **RadialControllerMenuKnownIcon** enum
- **RadialControllerSystemMenuItemKind** enum

Samples

Topic samples

RadialController customization

Other samples

Coloring Book sample

Universal Windows Platform samples (C# and C++)

Windows desktop sample

Speech interactions

Article • 06/24/2021

Integrate speech recognition and text-to-speech (also known as TTS, or speech synthesis) directly into the user experience of your app.

Speech recognition Speech recognition converts words spoken by the user into text for form input, for text dictation, to specify an action or command, and to accomplish tasks. Both pre-defined grammars for free-text dictation and web search, and custom grammars authored using Speech Recognition Grammar Specification (SRGS) Version 1.0 are supported.

TTS TTS uses a speech synthesis engine (voice) to convert a text string into spoken words. The input string can be either basic, unadorned text or more complex Speech Synthesis Markup Language (SSML). SSML provides a standard way to control characteristics of speech output, such as pronunciation, volume, pitch, rate or speed, and emphasis.

Other speech-related components: Cortana in Windows applications uses customized voice commands (spoken or typed) to launch your app to the foreground (the app takes focus, just as if it was launched from the Start menu) or activate as a background service (**Cortana** retains focus but provides results from the app). See Cortana interactions in Windows apps.

Speech interaction design

Designed and implemented thoughtfully, speech can be a robust and enjoyable way for people to interact with your app, complementing, or even replacing, keyboard, mouse, touch, and gestures.

These guidelines and recommendations describe how to best integrate both speech recognition and TTS into the interaction experience of your app.

If you are considering supporting speech interactions in your app:

- What actions can be taken through speech? Can a user navigate between pages, invoke commands, or enter data as text fields, brief notes, or long messages?
- Is speech input a good option for completing a task?
- How does a user know when speech input is available?
- Is the app always listening, or does the user need to take an action for the app to enter listening mode?

- What phrases initiate an action or behavior? Do the phrases and actions need to be enumerated on screen?
- Are prompt, confirmation, and disambiguation screens or TTS required?
- What is the interaction dialog between app and user?
- Is a custom or constrained vocabulary required (such as medicine, science, or locale) for the context of your app?
- Is network connectivity required?

Text input

Speech for text input can range from short form (single word or phrase) to long form (continuous dictation). Short form input must be less than 10 seconds in length, while long form input session can be up to two minutes in length. (Long form input can be restarted without user intervention to give the impression of continuous dictation.)

You should provide a visual cue to indicate that speech recognition is supported and available to the user and whether the user needs to turn it on. For example, a command bar button with a microphone glyph (see Command bars) can be used to show both availability and state.

Provide ongoing recognition feedback to minimize any apparent lack of response while recognition is being performed.

Let users revise recognition text using keyboard input, disambiguation prompts, suggestions, or additional speech recognition.

Stop recognition if input is detected from a device other than speech recognition, such as touch or keyboard. This probably indicates that the user has moved onto another task, such as correcting the recognition text or interacting with other form fields.

Specify the length of time for which no speech input indicates that recognition is over. Do not automatically restart recognition after this period of time as it typically indicates the user has stopped engaging with your app.

Disable all continuous recognition UI and terminate the recognition session if a network connection is not available. Continuous recognition requires a network connection.

Commanding

Speech input can initiate actions, invoke commands, and accomplish tasks.

If space permits, consider displaying the supported responses for the current app context, with examples of valid input. This reduces the potential responses your app has to process and also eliminates confusion for the user.

Try to frame your questions such that they elicit as specific a response as possible. For example, "What do you want to do today?" is very open ended and would require a very large grammar definition due to how varied the responses could be. Alternatively, "Would you like to play a game or listen to music?" constrains the response to one of two valid answers with a correspondingly small grammar definition. A small grammar is much easier to author and results in much more accurate recognition results.

Request confirmation from the user when speech recognition confidence is low. If the user's intent is unclear, it's better to get clarification than to initiate an unintended action.

You should provide a visual cue to indicate that speech recognition is supported and available to the user and whether the user needs to turn it on. For example, a command bar button with a microphone glyph (see Guidelines for command bars) can be used to show both availability and state.

If the speech recognition switch is typically out of view, consider displaying a state indicator in the content area of the app.

If recognition is initiated by the user, consider using the built-in recognition experience for consistency. The built-in experience includes customizable screens with prompts, examples, disambiguations, confirmations, and errors.

The screens vary depending on the specified constraints:

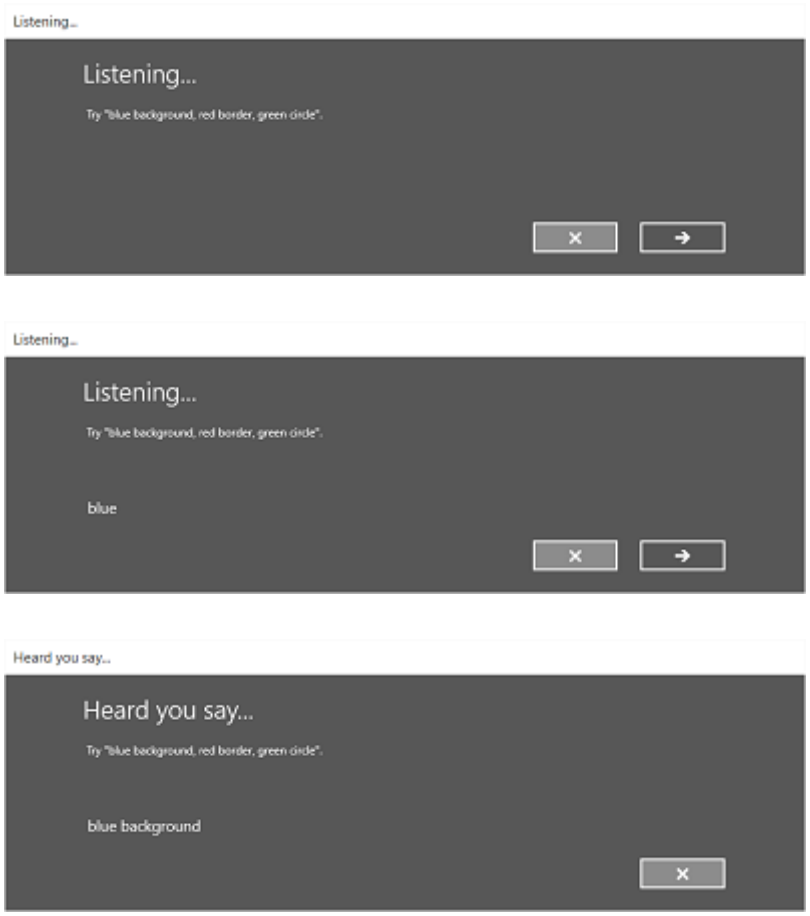
- Pre-defined grammar (dictation or web search)
  - The **Listening** screen.
  - The **Thinking** screen.
  - The **Heard you say** screen or the error screen.
- List of words or phrases, or a SRGS grammar file
  - The **Listening** screen.
  - The **Did you say** screen, if what the user said could be interpreted as more than one potential result.
  - The **Heard you say** screen or the error screen.

On the **Listening** screen you can:

- Customize the heading text.
- Provide example text of what the user can say.

- Specify whether the **Heard you say** screen is shown.
- Read the recognized string back to the user on the **Heard you say** screen.

Here is an example of the built-in recognition flow for a speech recognizer that uses a SRGS-defined constraint. In this example, speech recognition is successful.



Always listening

Your app can listen for and recognize speech input as soon as the app is launched, without user intervention.

You should customize the grammar constraints based on the app context. This keeps the speech recognition experience very targeted and relevant to the current task, and minimizes errors.

"What can I say?"

When speech input is enabled, it's important to help users discover what exactly can be understood and what actions can be performed.

If speech recognition is user enabled, consider using the command bar or a menu command to show all words and phrases supported in the current context.

If speech recognition is always on, consider adding the phrase "What can I say?" to every page. When the user says this phrase, display all words and phrases supported in the current context. Using this phrase provides a consistent way for users to discover speech capabilities across the system.

Recognition failures

Speech recognition will fail. Failures happen when audio quality is poor, when only part of a phrase is recognized, or when no input is detected at all.

Handle failure gracefully, help a user understand why recognition failed, and recover.

Your app should inform the user that they weren't understood and that they need to try again.

Consider providing examples of one or more supported phrases. The user is likely to repeat a suggested phrase, which increases recognition success.

You should display a list of potential matches for a user to select from. This can be far more efficient than going through the recognition process again.

You should always support alternative input types, which is especially helpful for handling repeated recognition failures. For example, you could suggest that the user try to use a keyboard, or use touch or a mouse to select from a list of potential matches.

Use the built-in speech recognition experience as it includes screens that inform the user that recognition was not successful and lets the user make another recognition attempt.

Listen for and try to correct issues in the audio input. The speech recognizer can detect issues with the audio quality that might adversely affect speech recognition accuracy. You can use the information provided by the speech recognizer to inform the user of the issue and let them take corrective action, if possible. For example, if the volume setting on the microphone is too low, you can prompt the user to speak louder or turn the volume up.

Constraints

Constraints, or grammars, define the spoken words and phrases that can be matched by the speech recognizer. You can specify one of the pre-defined web service grammars or you can create a custom grammar that is installed with your app.

Predefined grammars

Predefined dictation and web-search grammars provide speech recognition for your app without requiring you to author a grammar. When using these grammars, speech recognition is performed by a remote web service and the results are returned to the device

- The default free-text dictation grammar can recognize most words and phrases that a user can say in a particular language, and is optimized to recognize short phrases. Free-text dictation is useful when you don't want to limit the kinds of things a user can say. Typical uses include creating notes or dictating the content for a message.
- The web-search grammar, like a dictation grammar, contains a large number of words and phrases that a user might say. However, it is optimized to recognize terms that people typically use when searching the web.

> **Note**
>
> Because predefined dictation and web-search grammars can be large, and because they are online (not on the device), performance might not be as fast as with a custom grammar installed on the device.

These predefined grammars can be used to recognize up to 10 seconds of speech input and require no authoring effort on your part. However, they do require connection to a network.

Custom grammars

A custom grammar is designed and authored by you and is installed with your app. Speech recognition using a custom constraint is performed on the device.

- Programmatic list constraints provide a lightweight approach to creating simple grammars using a list of words or phrases. A list constraint works well for recognizing short, distinct phrases. Explicitly specifying all words in a grammar also improves recognition accuracy, as the speech recognition engine must only process speech to confirm a match. The list can also be programmatically updated.

- An SRGS grammar is a static document that, unlike a programmatic list constraint, uses the XML format defined by the SRGS Version 1.0. An SRGS grammar provides the greatest control over the speech recognition experience by letting you capture multiple semantic meanings in a single recognition.

Here are some tips for authoring SRGS grammars:

- Keep each grammar small. Grammars that contain fewer phrases tend to provide more accurate recognition than larger grammars that contain many phrases. It's better to have several smaller grammars for specific scenarios than to have a single grammar for your entire app.
- Let users know what to say for each app context and enable and disable grammars as needed.
- Design each grammar so users can speak a command in a variety of ways. For example, you can use the **GARBAGE** rule to match speech input that your grammar does not define. This lets users speak additional words that have no meaning to your app. For example, "give me", "and", "uh", "maybe", and so on.
- Use the sapi:subset element to help match speech input. This is a Microsoft extension to the SRGS specification to help match partial phrases.
- Try to avoid defining phrases in your grammar that contain only one syllable. Recognition tends to be more accurate for phrases containing two or more syllables.
- Avoid using phrases that sound similar. For example, phrases such as "hello", "bellow", and "fellow" can confuse the recognition engine and result in poor recognition accuracy.

> **Note**
>
> Which type of constraint type you use depends on the complexity of the recognition experience you want to create. Any could be the best choice for a specific recognition task, and you might find uses for all types of constraints in your app.

Custom pronunciations

If your app contains specialized vocabulary with unusual or fictional words, or words with uncommon pronunciations, you might be able to improve recognition performance for those words by defining custom pronunciations.

For a small list of words and phrases, or a list of infrequently used words and phrases, you can create custom pronunciations in a SRGS grammar. See token Element for more info.

For larger lists of words and phrases, or frequently used words and phrases, you can create separate pronunciation lexicon documents. See About Lexicons and Phonetic Alphabets for more info.

Testing

Test speech recognition accuracy and any supporting UI with your app's target audience. This is the best way to determine the effectiveness of the speech interaction experience in your app. For example, are users getting poor recognition results because your app isn't listening for a common phrase?

Either modify the grammar to support this phrase or provide users with a list of supported phrases. If you already provide the list of supported phrases, ensure it is easily discoverable.

Text-to-speech (TTS)

TTS generates speech output from plain text or SSML.

Try to design prompts that are polite and encouraging.

Consider whether you should read long strings of text. It's one thing to listen to a text message, but quite another to listen to a long list of search results that are difficult to remember.

You should provide media controls to let users pause, or stop, TTS.

You should listen to all TTS strings to ensure they are intelligible and sound natural.

- Stringing together an unusual sequence of words or speaking part numbers or punctuation might cause a phrase to become unintelligible.
- Speech can sound unnatural when the prosody or cadence is different from how a native speaker would say a phrase.

Both issues can be addressed by using SSML instead of plain text as input to the speech synthesizer. For more info about SSML, see Use SSML to Control Synthesized Speech and Speech Synthesis Markup Language Reference.

Other articles in this section

Expand table

Topic	Description
Speech recognition	Use speech recognition to provide input, specify an action or command, and accomplish tasks.

Topic	Description
Specify the speech recognizer language	Learn how to select an installed language to use for speech recognition.
Define custom recognition constraints	Learn how to define and use custom constraints for speech recognition.
Enable continuous dictation	Learn how to capture and recognize long-form, continuous dictation speech input.
Manage issues with audio input	Learn how to manage issues with speech-recognition accuracy caused by audio-input quality.
Set speech recognition timeouts	Set how long a speech recognizer ignores silence or unrecognizable sounds (babble) and continues listening for speech input.

Related articles

- Speech interactions
- Cortana interactions

Samples

- Speech recognition and speech synthesis sample

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Speech recognition

Article • 06/24/2021

Use speech recognition to provide input, specify an action or command, and accomplish tasks.

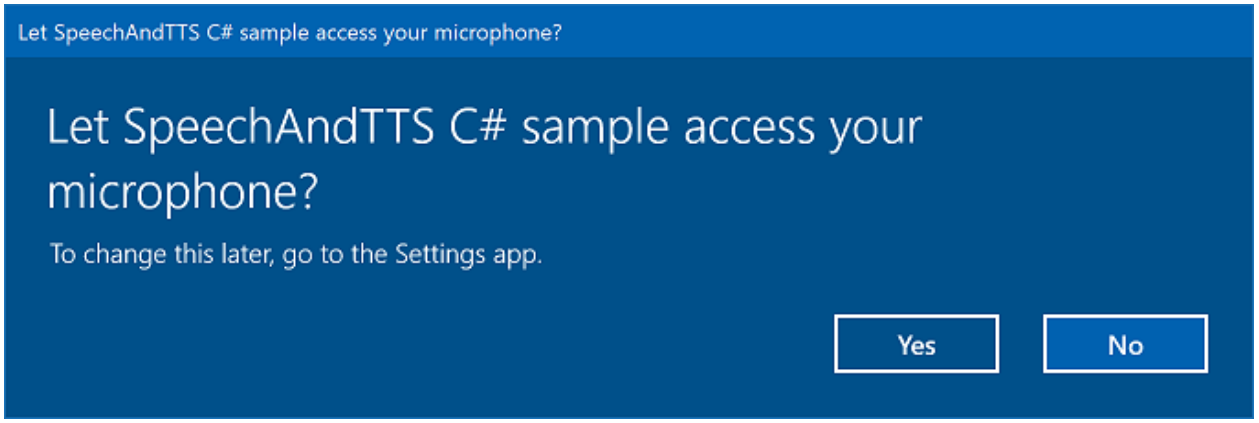
> **Important APIs**: Windows.Media.SpeechRecognition

Speech recognition is made up of a speech runtime, recognition APIs for programming the runtime, ready-to-use grammars for dictation and web search, and a default system UI that helps users discover and use speech recognition features.

Configure speech recognition

To support speech recognition with your app, the user must connect and enable a microphone on their device, and accept the Microsoft Privacy Policy granting permission for your app to use it.

To automatically prompt the user with a system dialog requesting permission to access and use the microphone's audio feed (example from the Speech recognition and speech synthesis sample shown below), just set the **Microphone** device capability in the App package manifest. For more detail, see App capability declarations.



If the user clicks Yes to grant access to the microphone, your app is added to the list of approved applications on the Settings -> Privacy -> Microphone page. However, as the user can choose to turn this setting off at any time, you should confirm that your app has access to the microphone before attempting to use it.

If you also want to support dictation, Cortana, or other speech recognition services (such as a predefined grammar defined in a topic constraint), you must also confirm that **Online speech recognition** (Settings -> Privacy -> Speech) is enabled.

This snippet shows how your app can check if a microphone is present and if it has permission to use it.

C#

```
public class AudioCapturePermissions
{
    // If no microphone is present, an exception is thrown with the
following HResult value.
    private static int NoCaptureDevicesHResult = -1072845856;

    /// <summary>
    /// Note that this method only checks the Settings->Privacy->Microphone
setting, it does not handle
    /// the Cortana/Dictation privacy check.
    ///
    /// You should perform this check every time the app gets focus, in case
the user has changed
    /// the setting while the app was suspended or not in focus.
    /// </summary>
    /// <returns>True, if the microphone is available.</returns>
    public async static Task<bool> RequestMicrophonePermission()
    {
        try
        {
            // Request access to the audio capture device.
            MediaCaptureInitializationSettings settings = new
MediaCaptureInitializationSettings();
            settings.StreamingCaptureMode = StreamingCaptureMode.Audio;
            settings.MediaCategory = MediaCategory.Speech;
            MediaCapture capture = new MediaCapture();

            await capture.InitializeAsync(settings);
        }
        catch (TypeLoadException)
        {
            // Thrown when a media player is not available.
            var messageDialog = new Windows.UI.Popups.MessageDialog("Media
player components are unavailable.");
            await messageDialog.ShowAsync();
            return false;
        }
        catch (UnauthorizedAccessException)
        {
            // Thrown when permission to use the audio capture device is
denied.
            // If this occurs, show an error or disable recognition
functionality.
            return false;
        }
        catch (Exception exception)
        {
            // Thrown when an audio capture device is not present.
            if (exception.HResult == NoCaptureDevicesHResult)
```

```
            {
                var messageDialog = new Windows.UI.Popups.MessageDialog("No
Audio Capture devices are present on this system.");
                await messageDialog.ShowAsync();
                return false;
            }
            else
            {
                throw;
            }
        }
        return true;
    }
}
```

C++

```
/// <summary>
/// Note that this method only checks the Settings->Privacy->Microphone
setting, it does not handle
/// the Cortana/Dictation privacy check.
///
/// You should perform this check every time the app gets focus, in case the
user has changed
/// the setting while the app was suspended or not in focus.
/// </summary>
/// <returns>True, if the microphone is available.</returns>
IAsyncOperation<bool>^
AudioCapturePermissions::RequestMicrophonePermissionAsync()
{
    return create_async([]()
    {
        try
        {
            // Request access to the audio capture device.
            MediaCaptureInitializationSettings^ settings = ref new
MediaCaptureInitializationSettings();
            settings->StreamingCaptureMode = StreamingCaptureMode::Audio;
            settings->MediaCategory = MediaCategory::Speech;
            MediaCapture^ capture = ref new MediaCapture();

            return create_task(capture->InitializeAsync(settings))
                .then([](task<void> previousTask) -> bool
            {
                try
                {
                    previousTask.get();
                }
                catch (AccessDeniedException^)
                {
                    // Thrown when permission to use the audio capture
device is denied.
                    // If this occurs, show an error or disable recognition
```

```
functionality.
                    return false;
                }
                catch (Exception^ exception)
                {
                    // Thrown when an audio capture device is not present.
                    if (exception->HResult ==
AudioCapturePermissions::NoCaptureDevicesHResult)
                    {
                        auto messageDialog = ref new
Windows::UI::Popups::MessageDialog("No Audio Capture devices are present on
this system.");
                        create_task(messageDialog->ShowAsync());
                        return false;
                    }

                    throw;
                }
                return true;
            });
        }
        catch (Platform::ClassNotRegisteredException^ ex)
        {
            // Thrown when a media player is not available.
            auto messageDialog = ref new
Windows::UI::Popups::MessageDialog("Media Player Components unavailable.");
            create_task(messageDialog->ShowAsync());
            return create_task([] {return false; });
        }
    });
}
```

JavaScript

```
var AudioCapturePermissions = WinJS.Class.define(
    function () { }, {},
    {
        requestMicrophonePermission: function () {
            /// <summary>
            /// Note that this method only checks the Settings->Privacy-
>Microphone setting, it does not handle
            /// the Cortana/Dictation privacy check.
            ///
            /// You should perform this check every time the app gets focus,
in case the user has changed
            /// the setting while the app was suspended or not in focus.
            /// </summary>
            /// <returns>True, if the microphone is available.</returns>
            return new WinJS.Promise(function (completed, error) {

                try {
                    // Request access to the audio capture device.
                    var captureSettings = new
```

```
Windows.Media.Capture.MediaCaptureInitializationSettings();
                    captureSettings.streamingCaptureMode =
Windows.Media.Capture.StreamingCaptureMode.audio;
                    captureSettings.mediaCategory =
Windows.Media.Capture.MediaCategory.speech;

                    var capture = new Windows.Media.Capture.MediaCapture();
                    capture.initializeAsync(captureSettings).then(function
() {
                        completed(true);
                    },
                    function (error) {
                        // Audio Capture can fail to initialize if there's
no audio devices on the system, or if
                        // the user has disabled permission to access the
microphone in the Privacy settings.
                        if (error.number == -2147024891) { // Access denied
(microphone disabled in settings)
                            completed(false);
                        } else if (error.number == -1072845856) { // No
recording device present.
                            var messageDialog = new
Windows.UI.Popups.MessageDialog("No Audio Capture devices are present on
this system.");
                            messageDialog.showAsync();
                            completed(false);
                        } else {
                            error(error);
                        }
                    });
                } catch (exception) {
                    if (exception.number == -2147221164) { //
REGDB_E_CLASSNOTREG
                        var messageDialog = new
Windows.UI.Popups.MessageDialog("Media Player components not available on
this system.");
                        messageDialog.showAsync();
                        return false;
                    }
                }
            });
        }
    })
```

Recognize speech input

A *constraint* defines the words and phrases (vocabulary) that an app recognizes in speech input. Constraints are at the core of speech recognition and give your app greater control over the accuracy of speech recognition.

You can use the following types of constraints for recognizing speech input.

Predefined grammars

Predefined dictation and web-search grammars provide speech recognition for your app without requiring you to author a grammar. When using these grammars, speech recognition is performed by a remote web service and the results are returned to the device.

The default free-text dictation grammar can recognize most words and phrases that a user can say in a particular language, and is optimized to recognize short phrases. The predefined dictation grammar is used if you don't specify any constraints for your SpeechRecognizer object. Free-text dictation is useful when you don't want to limit the kinds of things a user can say. Typical uses include creating notes or dictating the content for a message.

The web-search grammar, like a dictation grammar, contains a large number of words and phrases that a user might say. However, it is optimized to recognize terms that people typically use when searching the web.

> ⓘ **Note**
>
> Because predefined dictation and web-search grammars can be large, and because they are online (not on the device), performance might not be as fast as with a custom grammar installed on the device.

These predefined grammars can be used to recognize up to 10 seconds of speech input and require no authoring effort on your part. However, they do require a connection to a network.

To use web-service constraints, speech input and dictation support must be enabled in **Settings** by turning on the "Get to know me" option in **Settings -> Privacy -> Speech, inking, and typing**.

Here, we show how to test whether speech input is enabled and open the Settings -> Privacy -> Speech, inking, and typing page, if not.

First, we initialize a global variable (HResultPrivacyStatementDeclined) to the HResult value of 0x80045509. See Exception handling for in C# or Visual Basic.

C#

```
private static uint HResultPrivacyStatementDeclined = 0x80045509;
```

We then catch any standard exceptions during recognition and test if the HResult value is equal to the value of the HResultPrivacyStatementDeclined variable. If so, we display a warning and call `await Windows.System.Launcher.LaunchUriAsync(new Uri("ms-settings:privacy-accounts"));` to open the Settings page.

```
C#

catch (Exception exception)
{
  // Handle the speech privacy policy error.
  if ((uint)exception.HResult == HResultPrivacyStatementDeclined)
  {
    resultTextBlock.Visibility = Visibility.Visible;
    resultTextBlock.Text = "The privacy statement was declined." +
      "Go to Settings -> Privacy -> Speech, inking and typing, and ensure
you" +
      "have viewed the privacy policy, and 'Get To Know You' is enabled.";
    // Open the privacy/speech, inking, and typing settings page.
    await Windows.System.Launcher.LaunchUriAsync(new Uri("ms-
settings:privacy-accounts"));
  }
  else
  {
    var messageDialog = new
Windows.UI.Popups.MessageDialog(exception.Message, "Exception");
    await messageDialog.ShowAsync();
  }
}
```

See SpeechRecognitionTopicConstraint.

Programmatic list constraints

Programmatic list constraints provide a lightweight approach to creating simple grammars using a list of words or phrases. A list constraint works well for recognizing short, distinct phrases. Explicitly specifying all words in a grammar also improves recognition accuracy, as the speech recognition engine must only process speech to confirm a match. The list can also be programmatically updated.

A list constraint consists of an array of strings that represents speech input that your app will accept for a recognition operation. You can create a list constraint in your app by creating a speech-recognition list-constraint object and passing an array of strings. Then, add that object to the constraints collection of the recognizer. Recognition is successful when the speech recognizer recognizes any one of the strings in the array.

See SpeechRecognitionListConstraint.

SRGS grammars

An Speech Recognition Grammar Specification (SRGS) grammar is a static document that, unlike a programmatic list constraint, uses the XML format defined by the SRGS Version 1.0. An SRGS grammar provides the greatest control over the speech recognition experience by letting you capture multiple semantic meanings in a single recognition.

See **SpeechRecognitionGrammarFileConstraint**.

Voice command constraints

Use a Voice Command Definition (VCD) XML file to define the commands that the user can say to initiate actions when activating your app. For more detail, see Activate a foreground app with voice commands through Cortana.

See **SpeechRecognitionVoiceCommandDefinitionConstraint**/

Note The type of constraint type you use depends on the complexity of the recognition experience you want to create. Any could be the best choice for a specific recognition task, and you might find uses for all types of constraints in your app. To get started with constraints, see Define custom recognition constraints.

The predefined Universal Windows app dictation grammar recognizes most words and short phrases in a language. It is activated by default when a speech recognizer object is instantiated without custom constraints.

In this example, we show how to:

- Create a speech recognizer.
- Compile the default Universal Windows app constraints (no grammars have been added to the speech recognizer's grammar set).
- Start listening for speech by using the basic recognition UI and TTS feedback provided by the **RecognizeWithUIAsync** method. Use the **RecognizeAsync** method if the default UI is not required.

CSharp

```
private async void StartRecognizing_Click(object sender, RoutedEventArgs e)
{
    // Create an instance of SpeechRecognizer.
    var speechRecognizer = new
Windows.Media.SpeechRecognition.SpeechRecognizer();

    // Compile the dictation grammar by default.
```

```
    await speechRecognizer.CompileConstraintsAsync();

    // Start recognition.
    Windows.Media.SpeechRecognition.SpeechRecognitionResult
speechRecognitionResult = await speechRecognizer.RecognizeWithUIAsync();

    // Do something with the recognition result.
    var messageDialog = new
Windows.UI.Popups.MessageDialog(speechRecognitionResult.Text, "Text
spoken");
    await messageDialog.ShowAsync();
}
```

Customize the recognition UI

When your app attempts speech recognition by calling SpeechRecognizer.RecognizeWithUIAsync, several screens are shown in the following order.

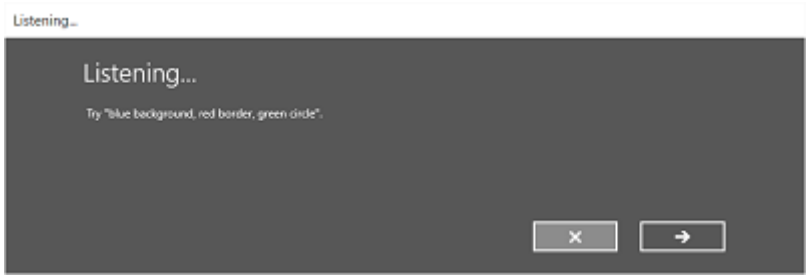
If you're using a constraint based on a predefined grammar (dictation or web search):

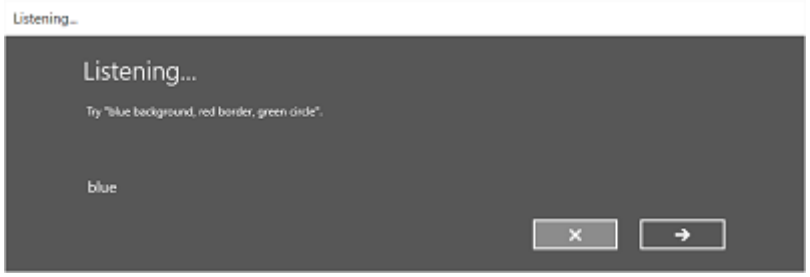
- The **Listening** screen.
- The **Thinking** screen.
- The **Heard you say** screen or the error screen.

If you're using a constraint based on a list of words or phrases, or a constraint based on a SRGS grammar file:

- The **Listening** screen.
- The **Did you say** screen, if what the user said could be interpreted as more than one potential result.
- The **Heard you say** screen or the error screen.

The following image shows an example of the flow between screens for a speech recognizer that uses a constraint based on a SRGS grammar file. In this example, speech recognition was successful.





The **Listening** screen can provide examples of words or phrases that the app can recognize. Here, we show how to use the properties of the SpeechRecognizerUIOptions class (obtained by calling the SpeechRecognizer.UIOptions property) to customize content on the **Listening** screen.

CSharp

```
private async void WeatherSearch_Click(object sender, RoutedEventArgs e)
{
    // Create an instance of SpeechRecognizer.
    var speechRecognizer = new
Windows.Media.SpeechRecognition.SpeechRecognizer();

    // Listen for audio input issues.
    speechRecognizer.RecognitionQualityDegrading +=
speechRecognizer_RecognitionQualityDegrading;

    // Add a web search grammar to the recognizer.
    var webSearchGrammar = new
Windows.Media.SpeechRecognition.SpeechRecognitionTopicConstraint(Windows.Med
ia.SpeechRecognition.SpeechRecognitionScenario.WebSearch, "webSearch");


    speechRecognizer.UIOptions.AudiblePrompt = "Say what you want to search
for...";
    speechRecognizer.UIOptions.ExampleText = @"Ex. 'weather for London'";
    speechRecognizer.Constraints.Add(webSearchGrammar);

    // Compile the constraint.
    await speechRecognizer.CompileConstraintsAsync();

    // Start recognition.
    Windows.Media.SpeechRecognition.SpeechRecognitionResult
speechRecognitionResult = await speechRecognizer.RecognizeWithUIAsync();
    //await speechRecognizer.RecognizeWithUIAsync();
```

```
    // Do something with the recognition result.
    var messageDialog = new
Windows.UI.Popups.MessageDialog(speechRecognitionResult.Text, "Text
spoken");
    await messageDialog.ShowAsync();
}
```

Related articles

- Speech interactions

Samples

- Speech recognition and speech synthesis sample

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Specify the speech recognizer language

Article • 06/24/2021

Learn how to select an installed language to use for speech recognition.

> **Important APIs**: SupportedTopicLanguages, SupportedGrammarLanguages, Language

Here, we enumerate the languages installed on a system, identify which is the default language, and select a different language for recognition.

Prerequisites:

This topic builds on Speech recognition.

You should have a basic understanding of speech recognition and recognition constraints.

If you're new to developing Windows apps, have a look through these topics to get familiar with the technologies discussed here.

- Create your first app
- Learn about events with Events and routed events overview

User experience guidelines:

For helpful tips about designing a useful and engaging speech-enabled app, see Speech design guidelines .

Identify the default language

A speech recognizer uses the system speech language as its default recognition language. This language is set by the user on the device Settings > System > Speech > Speech Language screen.

We identify the default language by checking the **SystemSpeechLanguage** static property.

CSharp

```
var language = SpeechRecognizer.SystemSpeechLanguage;
```

Confirm an installed language

Installed languages can vary between devices. You should verify the existence of a language if you depend on it for a particular constraint.

Note A reboot is required after a new language pack is installed. An exception with error code SPERR_NOT_FOUND (0x8004503a) is raised if the specified language is not supported or has not finished installing.

Determine the supported languages on a device by checking one of two static properties of the SpeechRecognizer class:

- SupportedTopicLanguages—The collection of Language objects used with predefined dictation and web search grammars.
- SupportedGrammarLanguages—The collection of Language objects used with a list constraint or a Speech Recognition Grammar Specification (SRGS) file.

Specify a language

To specify a language, pass a Language object in the SpeechRecognizer constructor.

Here, we specify "en-US" as the recognition language.

CSharp

```
var language = new Windows.Globalization.Language("en-US");
var recognizer = new SpeechRecognizer(language);
```

Remarks

A topic constraint can be configured by adding a SpeechRecognitionTopicConstraint to the Constraints collection of the SpeechRecognizer and then calling CompileConstraintsAsync. A SpeechRecognitionResultStatus of **TopicLanguageNotSupported** is returned if the recognizer is not initialized with a supported topic language.

A list constraint is configured by adding a SpeechRecognitionListConstraint to the Constraints collection of the SpeechRecognizer and then calling CompileConstraintsAsync. You cannot specify the language of a custom list directly. Instead, the list will be processed using the language of the recognizer.

An SRGS grammar is an open-standard XML format represented by the **SpeechRecognitionGrammarFileConstraint** class. Unlike custom lists, you can specify the language of the grammar in the SRGS markup. **CompileConstraintsAsync** fails with a **SpeechRecognitionResultStatus** of **TopicLanguageNotSupported** if the recognizer is not initialized to the same language as the SRGS markup.

Related articles

- Speech interactions

Samples

- Speech recognition and speech synthesis sample

Define custom recognition constraints

Article • 06/24/2021

Learn how to define and use custom constraints for speech recognition.

> **Important APIs**: SpeechRecognitionTopicConstraint, SpeechRecognitionListConstraint, SpeechRecognitionGrammarFileConstraint

Speech recognition requires at least one constraint to define a recognizable vocabulary. If no constraint is specified, the predefined dictation grammar of Universal Windows apps is used. See Speech recognition.

Add constraints

Use the SpeechRecognizer.Constraints property to add constraints to a speech recognizer.

Here, we cover the three kinds of speech recognition constraints used from within an app. (For Cortana voice command constraints, see Activate a foreground app with voice commands through Cortana.)

- SpeechRecognitionTopicConstraint—A constraint based on a predefined grammar (dictation or web search).
- SpeechRecognitionListConstraint—A constraint based on a list of words or phrases.
- SpeechRecognitionGrammarFileConstraint—A constraint defined in a Speech Recognition Grammar Specification (SRGS) file.

Each speech recognizer can have one constraint collection. Only these combinations of constraints are valid:

- A single-topic constraint (dictation or web search)
- For Windows 10 Fall Creators Update (10.0.16299.15) and newer, a single topic constraint can be combined with a list constraint
- A combination of list constraints and/or grammar-file constraints.

> **ⓘ Important**
>
> Call the SpeechRecognizer.CompileConstraintsAsync method to compile the constraints before starting the recognition process.

Specify a web-search grammar (SpeechRecognitionTopicConstraint)

Topic constraints (dictation or web-search grammar) must be added to the constraints collection of a speech recognizer.

> **Note**
>
> You can use a SpeechRecognitionListConstraint in conjunction with a SpeechRecognitionTopicConstraint to increase dictation accuracy by providing a set of domain-specific keywords that you think are likely to be used during dictation.

Here, we add a web-search grammar to the constraints collection.

CSharp

```
private async void WeatherSearch_Click(object sender, RoutedEventArgs e)
{
    // Create an instance of SpeechRecognizer.
    var speechRecognizer = new
Windows.Media.SpeechRecognition.SpeechRecognizer();

    // Listen for audio input issues.
    speechRecognizer.RecognitionQualityDegrading +=
speechRecognizer_RecognitionQualityDegrading;

    // Add a web search grammar to the recognizer.
    var webSearchGrammar = new
Windows.Media.SpeechRecognition.SpeechRecognitionTopicConstraint(Windows.Med
ia.SpeechRecognition.SpeechRecognitionScenario.WebSearch, "webSearch");


    speechRecognizer.UIOptions.AudiblePrompt = "Say what you want to search
for...";
    speechRecognizer.UIOptions.ExampleText = @"Ex. 'weather for London'";
    speechRecognizer.Constraints.Add(webSearchGrammar);

    // Compile the constraint.
    await speechRecognizer.CompileConstraintsAsync();

    // Start recognition.
    Windows.Media.SpeechRecognition.SpeechRecognitionResult
speechRecognitionResult = await speechRecognizer.RecognizeWithUIAsync();
    //await speechRecognizer.RecognizeWithUIAsync();

    // Do something with the recognition result.
    var messageDialog = new
```

```
Windows.UI.Popups.MessageDialog(speechRecognitionResult.Text, "Text
spoken");
    await messageDialog.ShowAsync();
}
```

Specify a programmatic list constraint (SpeechRecognitionListConstraint)

List constraints must be added to the constraints collection of a speech recognizer.

Keep the following points in mind:

- You can add multiple list constraints to a constraints collection.
- You can use any collection that implements **IIterable<String>** for the string values.

Here, we programmatically specify an array of words as a list constraint and add it to the constraints collection of a speech recognizer.

CSharp

```
private async void YesOrNo_Click(object sender, RoutedEventArgs e)
{
    // Create an instance of SpeechRecognizer.
    var speechRecognizer = new
Windows.Media.SpeechRecognition.SpeechRecognizer();

    // You could create this array dynamically.
    string[] responses = { "Yes", "No" };


    // Add a list constraint to the recognizer.
    var listConstraint = new
Windows.Media.SpeechRecognition.SpeechRecognitionListConstraint(responses,
"yesOrNo");

    speechRecognizer.UIOptions.ExampleText = @"Ex. 'yes', 'no'";
    speechRecognizer.Constraints.Add(listConstraint);

    // Compile the constraint.
    await speechRecognizer.CompileConstraintsAsync();

    // Start recognition.
    Windows.Media.SpeechRecognition.SpeechRecognitionResult
speechRecognitionResult = await speechRecognizer.RecognizeWithUIAsync();

    // Do something with the recognition result.
    var messageDialog = new
Windows.UI.Popups.MessageDialog(speechRecognitionResult.Text, "Text
spoken");
```

```
    await messageDialog.ShowAsync();
}
```

Specify an SRGS grammar constraint (SpeechRecognitionGrammarFileConstraint)

SRGS grammar files must be added to the constraints collection of a speech recognizer.

The SRGS Version 1.0 is the industry-standard markup language for creating XML-format grammars for speech recognition. Although Universal Windows apps provide alternatives to using SRGS for creating speech-recognition grammars, you might find that using SRGS to create grammars produces the best results, particularly for more involved speech recognition scenarios.

SRGS grammars provide a full set of features to help you architect complex voice interaction for your apps. For example, with SRGS grammars you can:

- Specify the order in which words and phrases must be spoken to be recognized.
- Combine words from multiple lists and phrases to be recognized.
- Link to other grammars.
- Assign a weight to an alternative word or phrase to increase or decrease the likelihood that it will be used to match speech input.
- Include optional words or phrases.
- Use special rules that help filter out unspecified or unanticipated input, such as random speech that doesn't match the grammar, or background noise.
- Use semantics to define what speech recognition means to your app.
- Specify pronunciations, either inline in a grammar or via a link to a lexicon.

For more info about SRGS elements and attributes, see the SRGS Grammar XML Reference . To get started creating an SRGS grammar, see How to Create a Basic XML Grammar.

Keep the following points in mind:

- You can add multiple grammar-file constraints to a constraints collection.
- Use the .grxml file extension for XML-based grammar documents that conform to SRGS rules.

This example uses an SRGS grammar defined in a file named srgs.grxml (described later). In the file properties, the **Package Action** is set to **Content** with **Copy to Output Directory** set to **Copy always**:

```
```

CSharp

```
private async void Colors_Click(object sender, RoutedEventArgs e)
{
    // Create an instance of SpeechRecognizer.
    var speechRecognizer = new
Windows.Media.SpeechRecognition.SpeechRecognizer();

    // Add a grammar file constraint to the recognizer.
    var storageFile = await
Windows.Storage.StorageFile.GetFileFromApplicationUriAsync(new Uri("ms-
appx:///Colors.grxml"));
    var grammarFileConstraint = new
Windows.Media.SpeechRecognition.SpeechRecognitionGrammarFileConstraint(stora
geFile, "colors");

    speechRecognizer.UIOptions.ExampleText = @"Ex. 'blue background', 'green
text'";
    speechRecognizer.Constraints.Add(grammarFileConstraint);

    // Compile the constraint.
    await speechRecognizer.CompileConstraintsAsync();

    // Start recognition.
    Windows.Media.SpeechRecognition.SpeechRecognitionResult
speechRecognitionResult = await speechRecognizer.RecognizeWithUIAsync();

    // Do something with the recognition result.
    var messageDialog = new
Windows.UI.Popups.MessageDialog(speechRecognitionResult.Text, "Text
spoken");
    await messageDialog.ShowAsync();
}
```

This SRGS file (srgs.grxml) includes semantic interpretation tags. These tags provide a mechanism for returning grammar match data to your app. Grammars must conform to the World Wide Web Consortium (W3C) Semantic Interpretation for Speech Recognition (SISR) 1.0 specification.

Here, we listen for variants of "yes" and "no".

CSharp

```
<grammar xml:lang="en-US"
         root="yesOrNo"
         version="1.0"
         tag-format="semantics/1.0"
         xmlns="http://www.w3.org/2001/06/grammar">

    <!-- The following rules recognize variants of yes and no. -->
      <rule id="yesOrNo">
         <one-of>
```

```
            <item>
              <one-of>
                 <item>yes</item>
                 <item>yeah</item>
                 <item>yep</item>
                 <item>yup</item>
                 <item>un huh</item>
                 <item>yay yus</item>
              </one-of>
              <tag>out="yes";</tag>
            </item>
            <item>
              <one-of>
                 <item>no</item>
                 <item>nope</item>
                 <item>nah</item>
                 <item>uh uh</item>
               </one-of>
               <tag>out="no";</tag>
            </item>
         </one-of>
      </rule>
</grammar>
```

Manage constraints

After a constraint collection is loaded for recognition, your app can manage which constraints are enabled for recognition operations by setting the **IsEnabled** property of a constraint to **true** or **false**. The default setting is **true**.

It's usually more efficient to load constraints once, enabling and disabling them as needed, rather than to load, unload, and compile constraints for each recognition operation. Use the **IsEnabled** property, as required.

Restricting the number of constraints serves to limit the amount of data that the speech recognizer needs to search and match against the speech input. This can improve both the performance and the accuracy of speech recognition.

Decide which constraints are enabled based on the phrases that your app can expect in the context of the current recognition operation. For example, if the current app context is to display a color, you probably don't need to enable a constraint that recognizes the names of animals.

To prompt the user for what can be spoken, use the **SpeechRecognizerUIOptions.AudiblePrompt** and **SpeechRecognizerUIOptions.ExampleText** properties, which are set by means of the **SpeechRecognizer.UIOptions** property. Preparing users for what they can say during the

recognition operation increases the likelihood that they will speak a phrase that can be matched to an active constraint.

Related articles

- Speech interactions

Samples

- Speech recognition and speech synthesis sample

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Continuous dictation

Article • 06/24/2021

Learn how to capture and recognize long-form, continuous dictation speech input.

> **Important APIs**: SpeechContinuousRecognitionSession, ContinuousRecognitionSession

In Speech recognition, you learned how to capture and recognize relatively short speech input using the RecognizeAsync or RecognizeWithUIAsync methods of a SpeechRecognizer object, for example, when composing a short message service (SMS) message or when asking a question.

For longer, continuous speech recognition sessions, such as dictation or email, use the ContinuousRecognitionSession property of a SpeechRecognizer to obtain a SpeechContinuousRecognitionSession object.

> **Note**
>
> Dictation language support depends on the device where your app is running. For PCs and laptops, only en-US is recognized, while Xbox and phones can recognize all languages supported by speech recognition. For more info, see Specify the speech recognizer language.

Set up

Your app needs a few objects to manage a continuous dictation session:

- An instance of a SpeechRecognizer object.
- A reference to a UI dispatcher to update the UI during dictation.
- A way to track the accumulated words spoken by the user.

Here, we declare a SpeechRecognizer instance as a private field of the code-behind class. Your app needs to store a reference elsewhere if you want continuous dictation to persist beyond a single Extensible Application Markup Language (XAML) page.

CSharp

```
private SpeechRecognizer speechRecognizer;
```

During dictation, the recognizer raises events from a background thread. Because a background thread cannot directly update the UI in XAML, your app must use a dispatcher to update the UI in response to recognition events.

Here, we declare a private field that will be initialized later with the UI dispatcher.

CSharp

```
// Speech events may originate from a thread other than the UI thread.
// Keep track of the UI thread dispatcher so that we can update the
// UI in a thread-safe manner.
private CoreDispatcher dispatcher;
```

To track what the user is saying, you need to handle recognition events raised by the speech recognizer. These events provide the recognition results for chunks of user utterances.

Here, we use a **StringBuilder** object to hold all the recognition results obtained during the session. New results are appended to the **StringBuilder** as they are processed.

CSharp

```
private StringBuilder dictatedTextBuilder;
```

Initialization

During the initialization of continuous speech recognition, you must:

- Fetch the dispatcher for the UI thread if you update the UI of your app in the continuous recognition event handlers.
- Initialize the speech recognizer.
- Compile the built-in dictation grammar. **Note** Speech recognition requires at least one constraint to define a recognizable vocabulary. If no constraint is specified, a predefined dictation grammar is used. See Speech recognition.
- Set up the event listeners for recognition events.

In this example, we initialize speech recognition in the **OnNavigatedTo** page event.

1. Because events raised by the speech recognizer occur on a background thread, create a reference to the dispatcher for updates to the UI thread. **OnNavigatedTo** is always invoked on the UI thread.

C#

```
this.dispatcher = CoreWindow.GetForCurrentThread().Dispatcher;
```

2. We then initialize the SpeechRecognizer instance.

```
C#

this.speechRecognizer = new SpeechRecognizer();
```

3. We then add and compile the grammar that defines all of the words and phrases that can be recognized by the SpeechRecognizer.

   If you don't specify a grammar explicitly, a predefined dictation grammar is used by default. Typically, the default grammar is best for general dictation.

   Here, we call CompileConstraintsAsync immediately without adding a grammar.

```
C#

SpeechRecognitionCompilationResult result =
      await speechRecognizer.CompileConstraintsAsync();
```

Handle recognition events

You can capture a single, brief utterance or phrase by calling RecognizeAsync or RecognizeWithUIAsync.

However, to capture a longer, continuous recognition session, we specify event listeners to run in the background as the user speaks and define handlers to build the dictation string.

We then use the ContinuousRecognitionSession property of our recognizer to obtain a SpeechContinuousRecognitionSession object that provides methods and events for managing a continuous recognition session.

Two events in particular are critical:

- ResultGenerated, which occurs when the recognizer has generated some results.
- Completed, which occurs when the continuous recognition session has ended.

The ResultGenerated event is raised as the user speaks. The recognizer continuously listens to the user and periodically raises an event that passes a chunk of speech input. You must examine the speech input, using the Result property of the event argument,

and take appropriate action in the event handler, such as appending the text to a StringBuilder object.

As an instance of SpeechRecognitionResult, the Result property is useful for determining whether you want to accept the speech input. A SpeechRecognitionResult provides two properties for this:

- Status indicates whether the recognition was successful. Recognition can fail for a variety of reasons.
- Confidence indicates the relative confidence that the recognizer understood the correct words.

Here are the basic steps for supporting continuous recognition:

1. Here, we register the handler for the ResultGenerated continuous recognition event in the OnNavigatedTo page event.

C#

```
speechRecognizer.ContinuousRecognitionSession.ResultGenerated +=
        ContinuousRecognitionSession_ResultGenerated;
```

2. We then check the Confidence property. If the value of Confidence is Medium or better, we append the text to the StringBuilder. We also update the UI as we collect input.

   Note the ResultGenerated event is raised on a background thread that cannot update the UI directly. If a handler needs to update the UI (as the [Speech and TTS sample] does), you must dispatch the updates to the UI thread through the RunAsync method of the dispatcher.

C#

```
private async void ContinuousRecognitionSession_ResultGenerated(
      SpeechContinuousRecognitionSession sender,
      SpeechContinuousRecognitionResultGeneratedEventArgs args)
      {

        if (args.Result.Confidence == SpeechRecognitionConfidence.Medium ||
          args.Result.Confidence == SpeechRecognitionConfidence.High)
          {
            dictatedTextBuilder.Append(args.Result.Text + " ");

            await dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
              dictationTextBox.Text = dictatedTextBuilder.ToString();
              btnClearText.IsEnabled = true;
```

```
            });
          }
        else
        {
          await dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
              dictationTextBox.Text = dictatedTextBuilder.ToString();
            });
        }
      }
```

3. We then handle the **Completed** event, which indicates the end of continuous dictation.

   The session ends when you call the **StopAsync** or **CancelAsync** methods (described the next section). The session can also end when an error occurs, or when the user has stopped speaking. Check the **Status** property of the event argument to determine why the session ended (**SpeechRecognitionResultStatus**).

   Here, we register the handler for the **Completed** continuous recognition event in the **OnNavigatedTo** page event.

```
C#

speechRecognizer.ContinuousRecognitionSession.Completed +=
      ContinuousRecognitionSession_Completed;
```

4. The event handler checks the Status property to determine whether the recognition was successful. It also handles the case where the user has stopped speaking. Often, a **TimeoutExceeded** is considered successful recognition as it means the user has finished speaking. You should handle this case in your code for a good experience.

   Note the **ResultGenerated** event is raised on a background thread that cannot update the UI directly. If a handler needs to update the UI (as the [Speech and TTS sample] does), you must dispatch the updates to the UI thread through the **RunAsync** method of the dispatcher.

```
C#

private async void ContinuousRecognitionSession_Completed(
      SpeechContinuousRecognitionSession sender,
      SpeechContinuousRecognitionCompletedEventArgs args)
      {
        if (args.Status != SpeechRecognitionResultStatus.Success)
        {
          if (args.Status == SpeechRecognitionResultStatus.TimeoutExceeded)
```

```
            {
              await dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
              {
                rootPage.NotifyUser(
                  "Automatic Time Out of Dictation",
                  NotifyType.StatusMessage);

                DictationButtonText.Text = " Continuous Recognition";
                dictationTextBox.Text = dictatedTextBuilder.ToString();
              });
            }
            else
            {
              await dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
              {
                rootPage.NotifyUser(
                  "Continuous Recognition Completed: " +
args.Status.ToString(),
                  NotifyType.StatusMessage);

                DictationButtonText.Text = " Continuous Recognition";
              });
            }
          }
        }
```

Provide ongoing recognition feedback

When people converse, they often rely on context to fully understand what is being said. Similarly, the speech recognizer often needs context to provide high-confidence recognition results. For example, by themselves, the words "weight" and "wait" are indistinguishable until more context can be gleaned from surrounding words. Until the recognizer has some confidence that a word, or words, have been recognized correctly, it will not raise the **ResultGenerated** event.

This can result in a less than ideal experience for the user as they continue speaking and no results are provided until the recognizer has high enough confidence to raise the **ResultGenerated** event.

Handle the **HypothesisGenerated** event to improve this apparent lack of responsiveness. This event is raised whenever the recognizer generates a new set of potential matches for the word being processed. The event argument provides a **Hypothesis** property that contains the current matches. Show these to the user as they continue speaking and reassure them that processing is still active. Once confidence is high and a recognition result has been determined, replace the interim **Hypothesis** results with the final **Result** provided in the **ResultGenerated** event.

Here, we append the hypothetical text and an ellipsis ("...") to the current value of the output TextBox. The contents of the text box are updated as new hypotheses are generated and until the final results are obtained from the ResultGenerated event.

CSharp

```
private async void SpeechRecognizer_HypothesisGenerated(
  SpeechRecognizer sender,
  SpeechRecognitionHypothesisGeneratedEventArgs args)
  {

    string hypothesis = args.Hypothesis.Text;
    string textboxContent = dictatedTextBuilder.ToString() + " " +
hypothesis + " ...";

    await dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
      dictationTextBox.Text = textboxContent;
      btnClearText.IsEnabled = true;
    });
  }
```

Start and stop recognition

Before starting a recognition session, check the value of the speech recognizer State property. The speech recognizer must be in an Idle state.

After checking the state of the speech recognizer, we start the session by calling the StartAsync method of the speech recognizer's ContinuousRecognitionSession property.

CSharp

```
if (speechRecognizer.State == SpeechRecognizerState.Idle)
{
  await speechRecognizer.ContinuousRecognitionSession.StartAsync();
}
```

Recognition can be stopped in two ways:

- StopAsync lets any pending recognition events complete (ResultGenerated continues to be raised until all pending recognition operations are complete).
- CancelAsync terminates the recognition session immediately and discards any pending results.

After checking the state of the speech recognizer, we stop the session by calling the CancelAsync method of the speech recognizer's ContinuousRecognitionSession

property.

CSharp

```
if (speechRecognizer.State != SpeechRecognizerState.Idle)
{
  await speechRecognizer.ContinuousRecognitionSession.CancelAsync();
}
```

Note

A ResultGenerated event can occur after a call to CancelAsync.
Because of multithreading, a ResultGenerated event might still remain on the stack when CancelAsync is called. If so, the **ResultGenerated** event still fires.
If you set any private fields when canceling the recognition session, always confirm their values in the ResultGenerated handler. For example, don't assume a field is initialized in your handler if you set them to null when you cancel the session.

Related articles

- Speech interactions

Samples

- Speech recognition and speech synthesis sample

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Manage issues with audio input

Article • 06/24/2021

Learn how to manage issues with speech-recognition accuracy caused by audio-input quality.

> **Important APIs**: SpeechRecognizer, RecognitionQualityDegrading, SpeechRecognitionAudioProblem

Assess audio-input quality

When speech recognition is active, use the RecognitionQualityDegrading event of your speech recognizer to determine whether one or more audio issues might be interfering with speech input. The event argument (SpeechRecognitionQualityDegradingEventArgs) provides the Problem property, which describes the issues detected with the audio input.

Recognition can be affected by too much background noise, a muted microphone, and the volume or speed of the speaker.

Here, we configure a speech recognizer and start listening for the RecognitionQualityDegrading event.

CSharp

```
private async void WeatherSearch_Click(object sender, RoutedEventArgs e)
{
    // Create an instance of SpeechRecognizer.
    var speechRecognizer = new
Windows.Media.SpeechRecognition.SpeechRecognizer();

    // Listen for audio input issues.
    speechRecognizer.RecognitionQualityDegrading +=
speechRecognizer_RecognitionQualityDegrading;

    // Add a web search grammar to the recognizer.
    var webSearchGrammar = new
Windows.Media.SpeechRecognition.SpeechRecognitionTopicConstraint(Windows.Med
ia.SpeechRecognition.SpeechRecognitionScenario.WebSearch, "webSearch");


    speechRecognizer.UIOptions.AudiblePrompt = "Say what you want to search
for...";
    speechRecognizer.UIOptions.ExampleText = "Ex. 'weather for London'";
    speechRecognizer.Constraints.Add(webSearchGrammar);
```

```
    // Compile the constraint.
    await speechRecognizer.CompileConstraintsAsync();

    // Start recognition.
    Windows.Media.SpeechRecognition.SpeechRecognitionResult
speechRecognitionResult = await speechRecognizer.RecognizeWithUIAsync();
    //await speechRecognizer.RecognizeWithUIAsync();

    // Do something with the recognition result.
    var messageDialog = new
Windows.UI.Popups.MessageDialog(speechRecognitionResult.Text, "Text
spoken");
    await messageDialog.ShowAsync();
}
```

Manage the speech-recognition experience

Use the description provided by the **Problem** property to help the user improve conditions for recognition.

Here, we create a handler for the **RecognitionQualityDegrading** event that checks for a low volume level. We then use a **SpeechSynthesizer** object to suggest that the user try speaking louder.

CSharp

```
private async void speechRecognizer_RecognitionQualityDegrading(
    Windows.Media.SpeechRecognition.SpeechRecognizer sender,

Windows.Media.SpeechRecognition.SpeechRecognitionQualityDegradingEventArgs
args)
{
    // Create an instance of a speech synthesis engine (voice).
    var speechSynthesizer =
        new Windows.Media.SpeechSynthesis.SpeechSynthesizer();

    // If input speech is too quiet, prompt the user to speak louder.
    if (args.Problem ==
Windows.Media.SpeechRecognition.SpeechRecognitionAudioProblem.TooQuiet)
    {
        // Generate the audio stream from plain text.
        Windows.Media.SpeechSynthesis.SpeechSynthesisStream stream;
        try
        {
            stream = await
speechSynthesizer.SynthesizeTextToStreamAsync("Try speaking louder");
            stream.Seek(0);
        }
        catch (Exception)
        {
```

```
            stream = null;
        }

        // Send the stream to the MediaElement declared in XAML.
        await
CoreApplication.MainView.CoreWindow.Dispatcher.RunAsync(Windows.UI.Core.Core
DispatcherPriority.High, () =>
        {
            this.media.SetSource(stream, stream.ContentType);
        });
    }
}
```

Related articles

- Speech interactions

Samples

- Speech recognition and speech synthesis sample

Set speech recognition timeouts

Article • 06/24/2021

Set how long a speech recognizer ignores silence or unrecognizable sounds (babble) and continues listening for speech input.

> **Important APIs**: Timeouts, SpeechRecognizerTimeouts

Set a timeout

Here, we specify various **Timeouts** values:

- InitialSilenceTimeout - The length of time that a SpeechRecognizer detects silence (before any recognition results have been generated) and assumes speech input is not forthcoming.
- BabbleTimeout - The length of time that a SpeechRecognizer continues to listen to unrecognizable sounds (babble) before it assumes speech input has ended and finalizes the recognition operation.
- EndSilenceTimeout - The length of time that a SpeechRecognizer detects silence (after recognition results have been generated) and assumes speech input has ended.

Note Timeouts can be set on a per-recognizer basis.

CSharp

```
// Set timeout settings.
recognizer.Timeouts.InitialSilenceTimeout = TimeSpan.FromSeconds(6.0);
recognizer.Timeouts.BabbleTimeout = TimeSpan.FromSeconds(4.0);
recognizer.Timeouts.EndSilenceTimeout = TimeSpan.FromSeconds(1.2);
```

Related articles

- Speech interactions

Samples

- Speech recognition and speech synthesis sample

Cortana interactions in Windows apps

Article • 10/02/2024

> ⚠ **Warning**
>
> This feature is no longer supported as of the Windows 10 May 2020 Update (version 2004, codename "20H1").

Extend the basic functionality of **Cortana** with voice commands that launch and execute a single action in a Windows application.

The target app can be launched in the foreground (the app takes focus and **Cortana** is dismissed) or activated in the background (**Cortana** retains focus but provides results from the app), depending on the complexity of the interaction. Generally, voice commands that require additional context or user input are best handled in a foreground app, while basic commands can be handled in **Cortana** through a background app.

By integrating the basic functionality of your app, and providing a central entry point for the user to accomplish most of the tasks without opening your app directly, **Cortana** becomes a liaison between your app and the user. Providing this shortcut to app functionality and reducing the need to switch apps, can save the user significant time and effort.

> ⓘ **Note**
>
> A voice command is a single utterance with a specific intent, defined in a Voice Command Definition (VCD) file, directed at an installed app via **Cortana**.
>
> A VCD file defines one or more voice commands, each with a unique intent.
>
> Voice command definitions can vary in complexity. They can support anything from a single, constrained utterance to a collection of more flexible, natural language utterances, all denoting the same intent.

Other speech and conversation components

Speech, voice, and conversation in Windows 10

See Speech, voice, and conversation in Windows 10 for information on how the various Windows development frameworks provide speech recognition, speech synthesis, and conversation support for developers building Windows applications.

Related articles

- VCD elements and attributes v1.2
- Cortana design guidelines
- Cortana voice command sample

Cortana design guidelines

Article • 06/24/2021

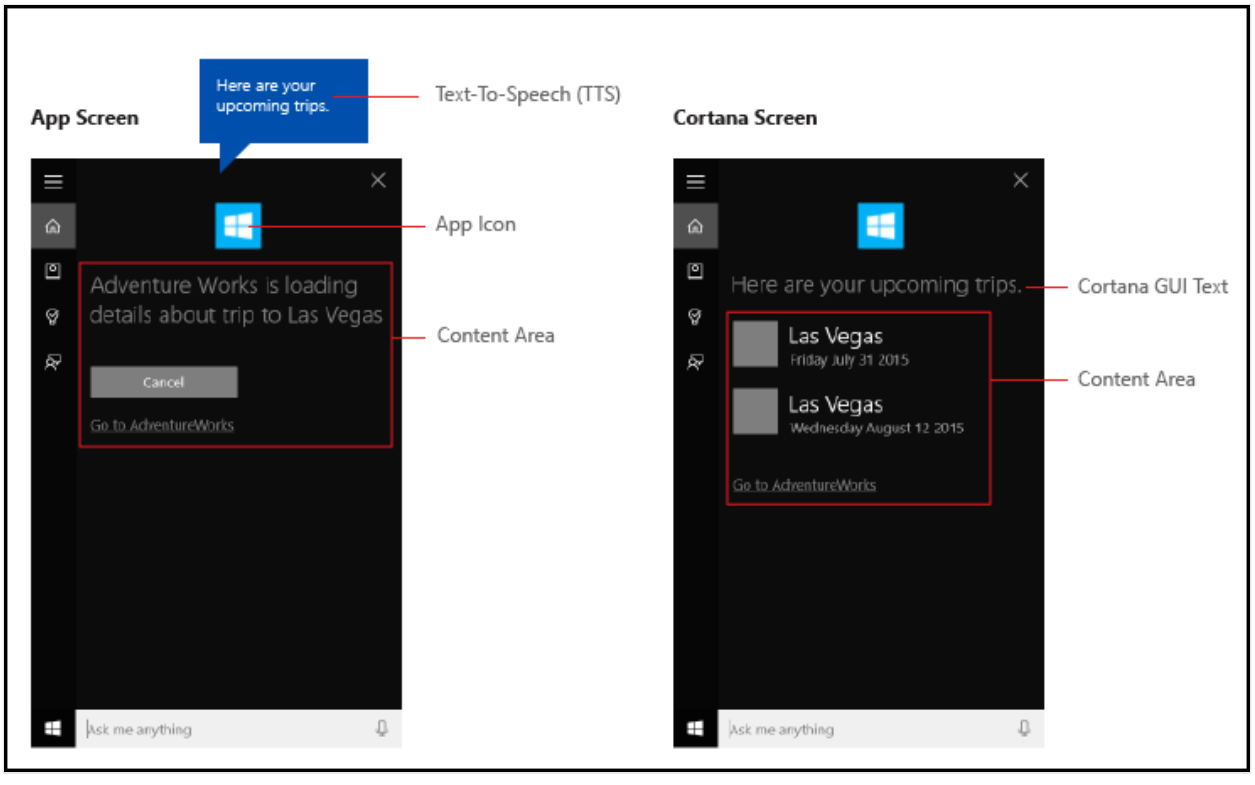
> **⚠ Warning**
>
> This feature is no longer supported as of the Windows 10 May 2020 Update (version 2004, codename "20H1").

These guidelines and recommendations describe how your app can best use **Cortana** to interact with the user, help them accomplish a task, and communicate clearly how it's all happening.

Cortana enables applications running in the background to prompt the user for confirmation or disambiguation, and in return provide the user with feedback on the status of the voice command. The process is lightweight, quick, and doesn't force the user to leave the **Cortana** experience or switch context to the application.

While the user should feel that **Cortana** is helping to make the process as light and easy as possible, you probably want **Cortana** to also be explicit that it's your app accomplishing the task.

We use a trip planning and management app named **Adventure Works** integrated into the **Cortana** UI, shown here, to demonstrate many of the concepts and features we discuss. For more info, see the Cortana voice command sample.



Conversational writing

Successful **Cortana** interactions require you to follow some fundamental principles when crafting text-to-speech (TTS) and GUI strings.

Expand table

Principle	Bad example	Good example
Efficient Use as few words as possible and put the most important information up front.	Sure can do, what movie would you like to search for today? We have a large collection.	Sure, what movie are you looking for?
Relevant Provide information pertinent only to the task, content, and context.	I've added this to your playlist. Just so you know, your battery is getting low.	I've added this to your playlist.
Clear Avoid ambiguity. Use everyday language instead of technical jargon.	No results for query "Trips to Las Vegas".	I couldn't find any trips to Las Vegas.
Trustworthy Be as accurate as possible. Be transparent about what's going on in the background—if a task hasn't finished yet, don't say that it has. Respect privacy—don't read private information out loud.	I couldn't find that movie, it must not have been released yet.	I couldn't find that movie in our catalogue.

Write how people speak. Don't emphasize grammatical accuracy over sounding natural. For example, ear-friendly verbal shortcuts like "wanna" or "gotta" are fine for TTS read out.

Use the implied first-person tense where possible and natural. For example, "Looking for your next Adventure Works trip" implies that someone is doing the looking, but does not use the word "I" to specify.

Use some variation to help make your app sound more natural. Provide different versions of your TTS and GUI strings to effectively say the same thing. For example, "What movie do you wanna see?" could have alternatives like "What movie would you like to watch?". People don't say the same thing the exact same way every time. Just make sure to keep your TTS and GUI versions in sync.

Use phrases like "OK" and "Alright" in your responses judiciously. While they can provide acknowledgment and a sense of progress, they can also get repetitive if used too often and without variation.

> ⓘ **Note**
>
> Use acknowledgment phrases in TTS only. Due to the limited space on the **Cortana** canvas, don't repeat them in the corresponding GUI strings.

Use contractions in your responses for more natural interactions and additional space saving on the **Cortana** canvas. For example," I can't find that movie" instead of "I was unable to find that movie". Write for the ear, not the eye.

Use language that the system understands. Users tend to repeat the terms they are presented with. Know what you display.

Use some variation in your responses by rotating, or randomly selecting, from a collection of alternative responses. For example, "What movie do you wanna see?" and "What film would you like to watch?". This makes your app sound more natural and unique.

Localization

To initiate an action using a voice command, your app must register voice commands in the language the user has selected on their device (Settings > System > Speech > Speech Language).

You should localize the voice commands your app responds to and all TTS and GUI strings.

You should avoid lengthy GUI strings. The **Cortana** canvas provides three lines for responses and will truncate strings longer than that.

For more info, see the Globalization and localization section.

Image resources and scaling

Universal Windows Platform (UWP) apps can automatically select the most appropriate app logo image based on specific settings and device capabilities (high contrast, effective pixels, locale, and so on). All you need to do is provide the images and ensure you use the appropriate naming convention and folder organization within the app project for the different resource versions. If you don't provide the recommended

resource versions, accessibility, localization, and image quality can suffer, depending on the user's preferences, abilities, device type, and location.

For more detail on image resources for high contrast and scale factors, see Guidelines for tile and icon assets.

You name resources using qualifiers. Resource qualifiers are folder and filename modifiers that identify the context in which a particular version of a resource should be used.

The standard naming convention is "foldername/qualifiername-value[_qualifiername-value]/filename.qualifiername-value[_qualifiername-value].ext". For example: images/logo.scale-100_contrast-white.png is simply referred to in code using the root folder and the filename: images/logo.png. See Manage language and region and How to name resources using qualifiers.

We recommend that you mark the default language on string resource files (such as "en-US\resources.resw") and the default scale factor on images (such as "logo.scale-100.png"), even if you do not currently plan to provide localized or multiple resolution resources. However, at a minimum, we recommend that you provide assets for 100, 200, and 400 scale factors.

> **ⓘ Important**
>
> The app icon used in the title area of the Cortana canvas is the Square44x44Logo icon specified in the "Package.appxmanifest" file.

You can also specify an icon for each result tile for a user query. Valid image sizes for results icons are:

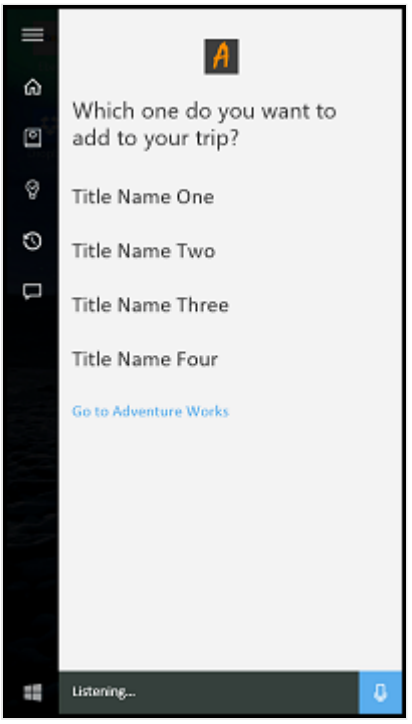
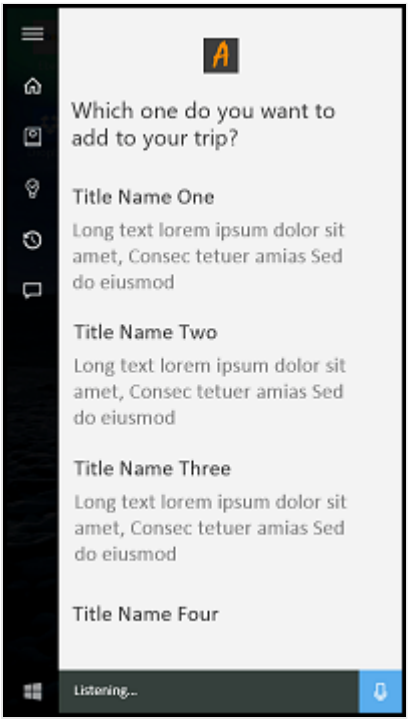
- 68w x 68h
- 68w x 92h
- 280w x 140h

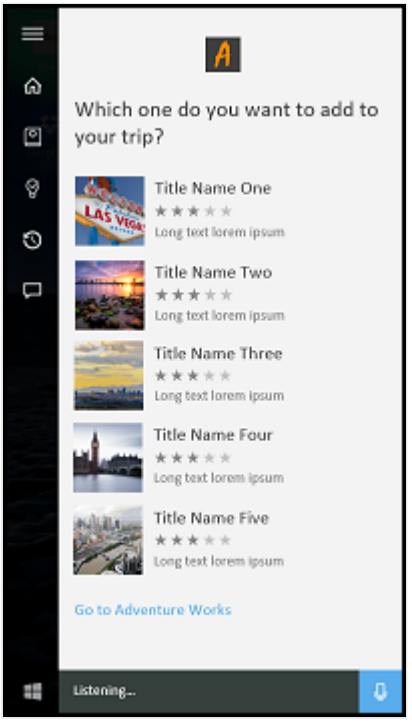
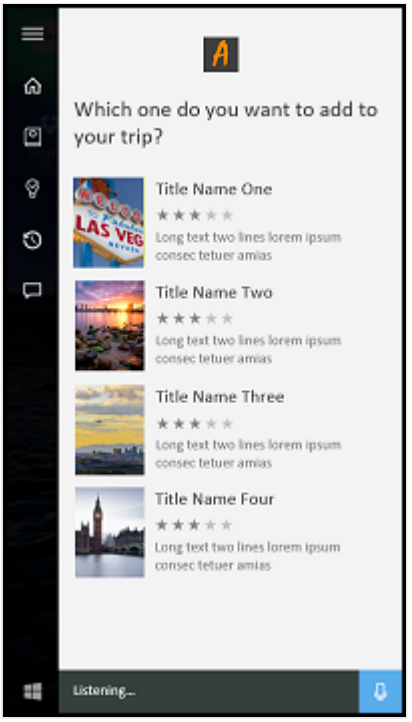
Result tile templates

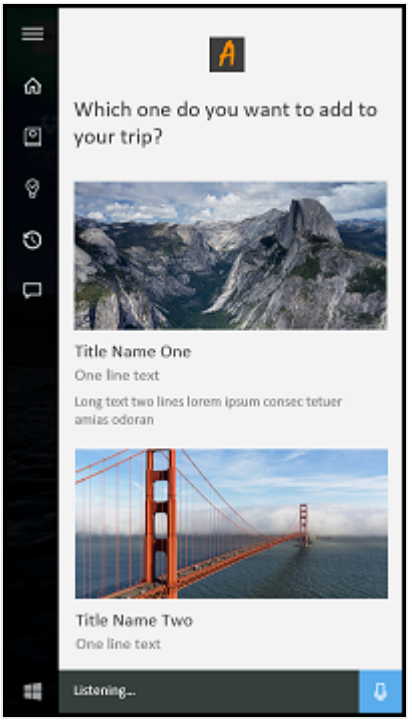
A set of templates are provided for the result tiles displayed on the Cortana canvas. Use these templates to specify the tile title and whether the tile includes text and a result icon image. Each tile can include up to three lines of text and one image, depending on the template specified.

Here are the supported templates (with examples):

Expand table

Name	Example
Title only	
Title with text	
Title with 68x68 icon	no image

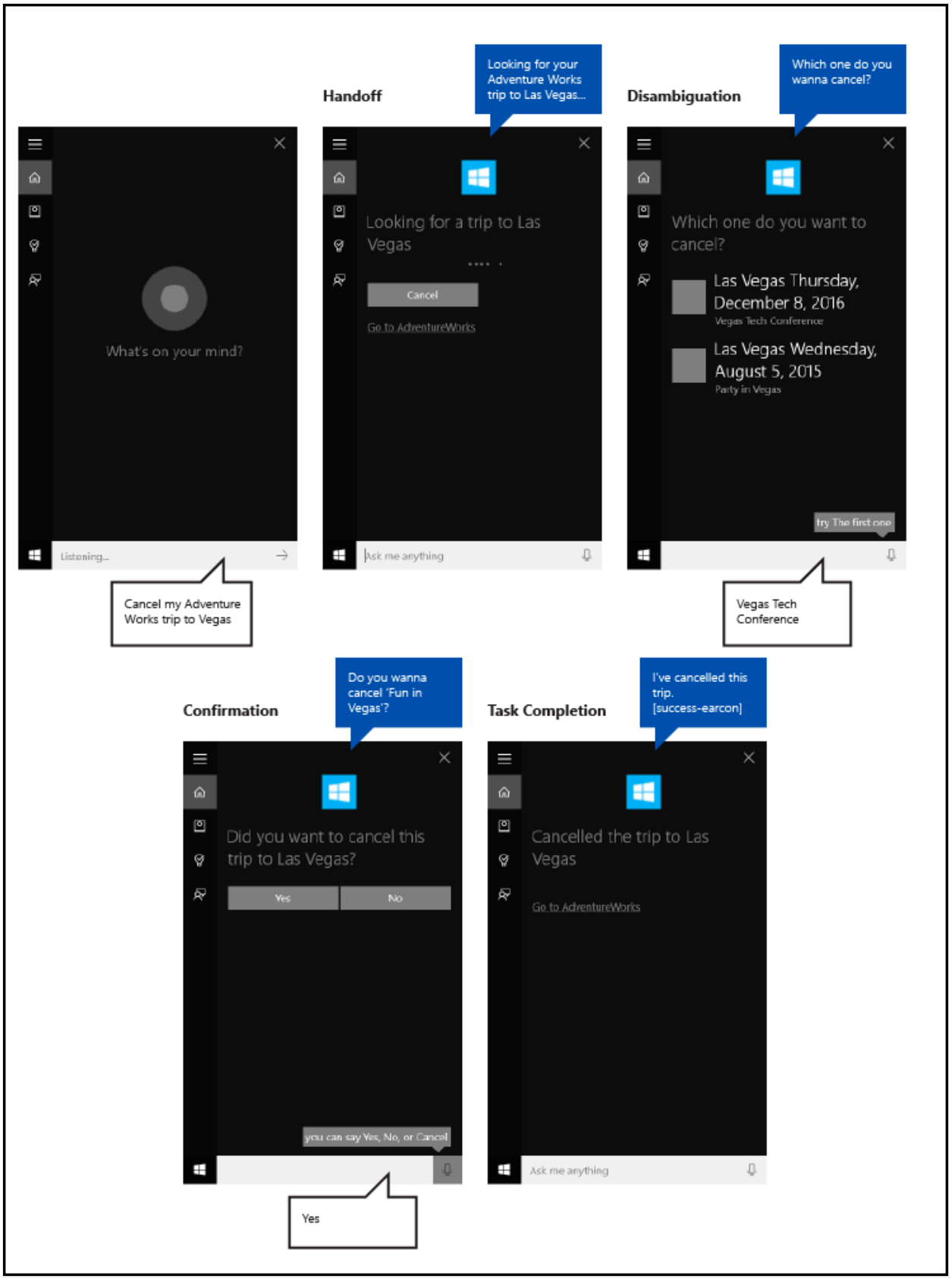
Name	Example
Title with 68x68 icon and text	
Title with 68x92 icon	no image
Title with 68x92 icon and text	
Title with 280x140 icon	no image

Name	Example
Title with 280x140 icon and text	

See VoiceCommandContentTileType for more info on Cortana templates.

Example

This example demonstrates an end-to-end task flow for a background app in **Cortana**. We're using the **Adventure Works** app to cancel a trip to Las Vegas. This example uses the "Title with 68x68 icon and text" template.



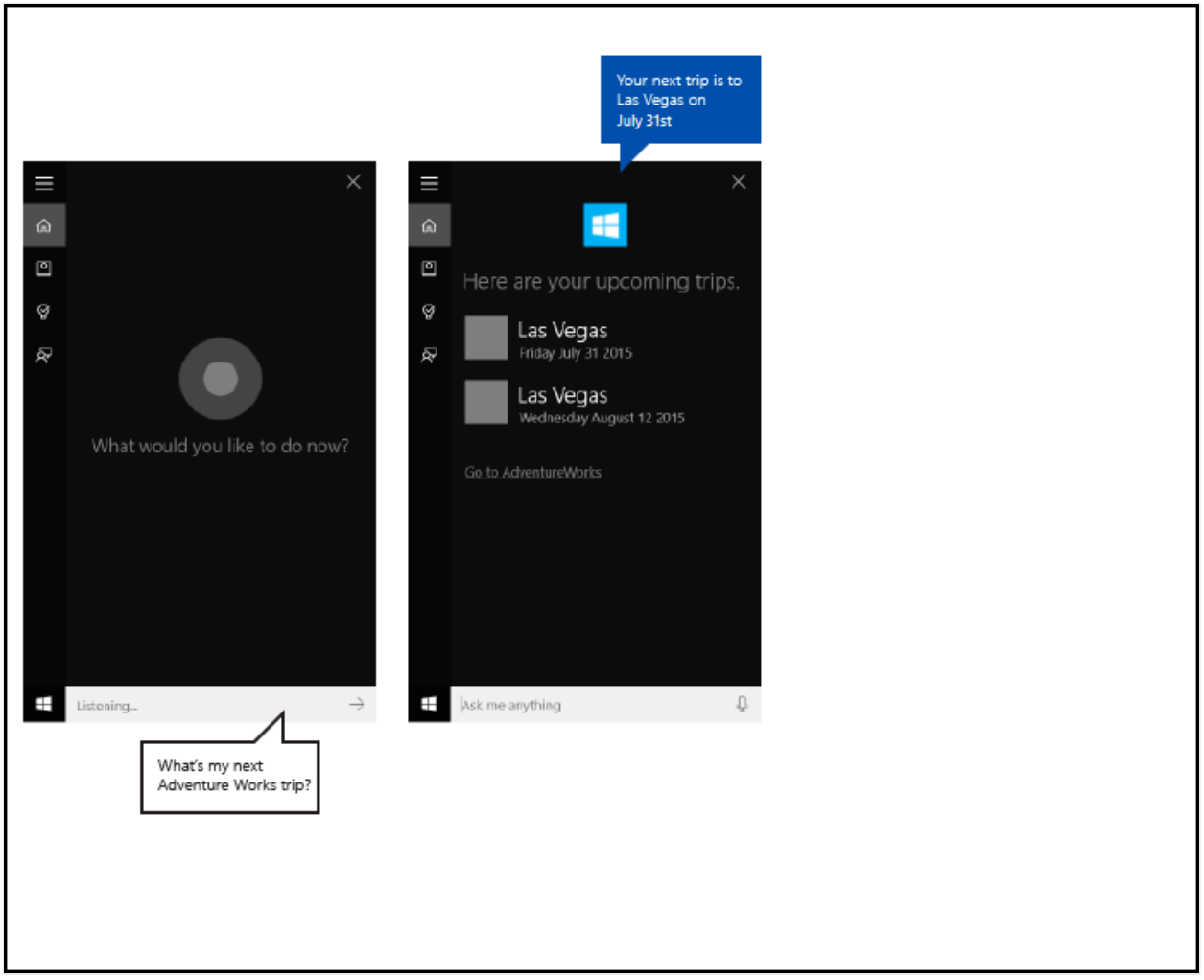
Here are the steps outlined in this image:

1. The user taps the microphone to initiate **Cortana**.
2. The user says "Cancel my Adventure Works trip to Vegas" to launch the **Adventure Works** app in the background. The app uses both **Cortana** speech and canvas to interact with the user.

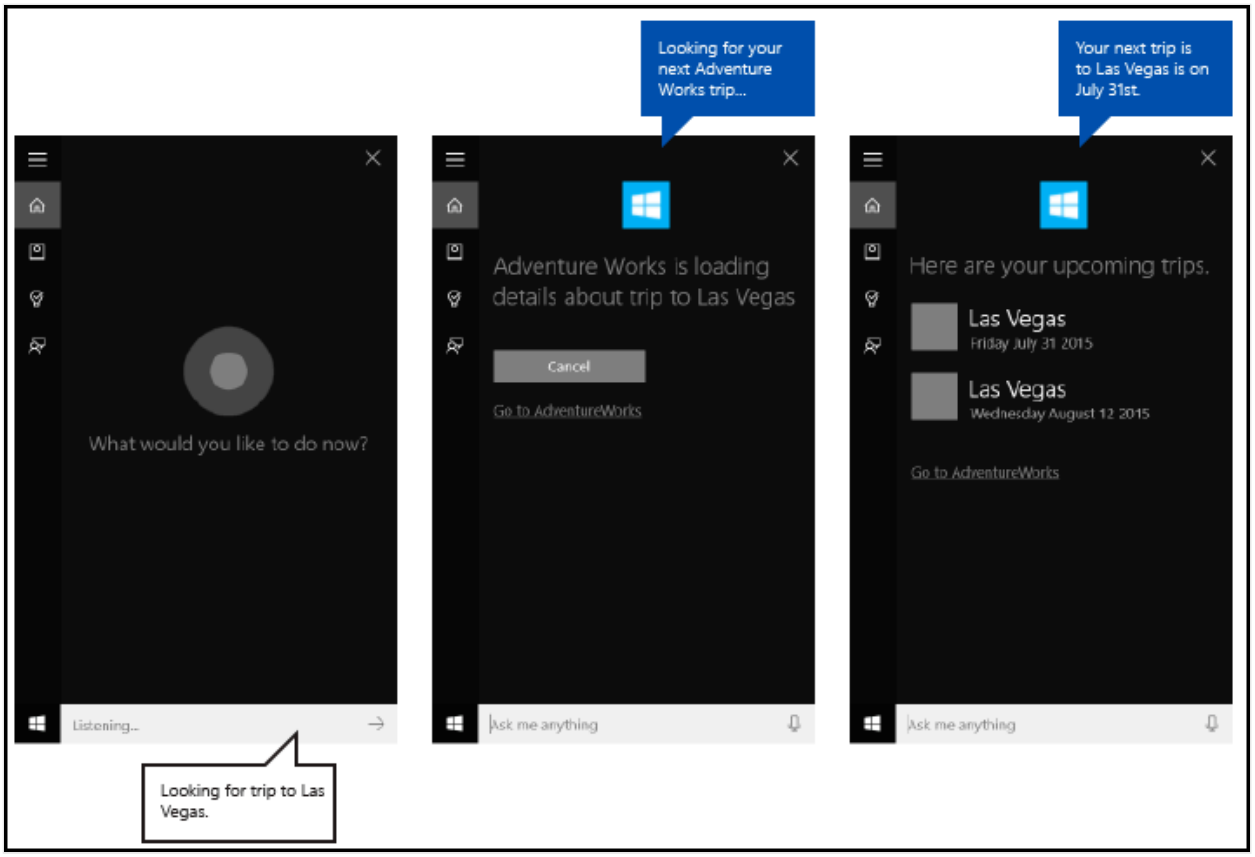
3. **Cortana** transitions to a handoff screen that gives the user acknowledgment feedback ("I'll get Adventure Works on that."), a status bar, and a cancel button.
4. In this case, the user has multiple trips that match the query, so the app provides a disambiguation screen that lists all the matching results and asks, "Which one do you wanna cancel?"
5. The user specifies the "Vegas Tech Conference" item.
6. As the cancellation cannot be undone, the app provides a confirmation screen that asks the user to confirm their intent.
7. The user says "Yes".
8. The app then provides a completion screen that shows the result of the operation.

We explore these steps in more detail here.

Handoff



AdventureWorks "Upcoming trip" with no handoff screen



AdventureWorks "Upcoming trip" with handoff screen

Tasks that take less than 500ms for your app to respond, and require no additional information from the user, can be completed without further participation from **Cortana**, other than displaying the completion screen.

If your application requires more than 500ms to respond, **Cortana** provides a handoff screen. The app icon and name are displayed, and you must provide both GUI and TTS handoff strings to indicate that the voice command was correctly understood. The handoff screen will be shown for up to 5 seconds; if your app doesn't respond within this time, **Cortana** presents a generic error screen.

GUI and TTS guidelines for handoff screens

Clearly indicate that the task is in progress.

Use present tense.

Use an action verb that confirms what task is initiating and reference the specific entity.

Use a generic verb that doesn't commit to the requested, incomplete action. For example, "Looking for your trip" instead of "Canceling your trip". In this case, if no results are returned the user doesn't hear something like "Cancelling your trip to Las Vegas... I couldn't find a trip to Las Vegas".

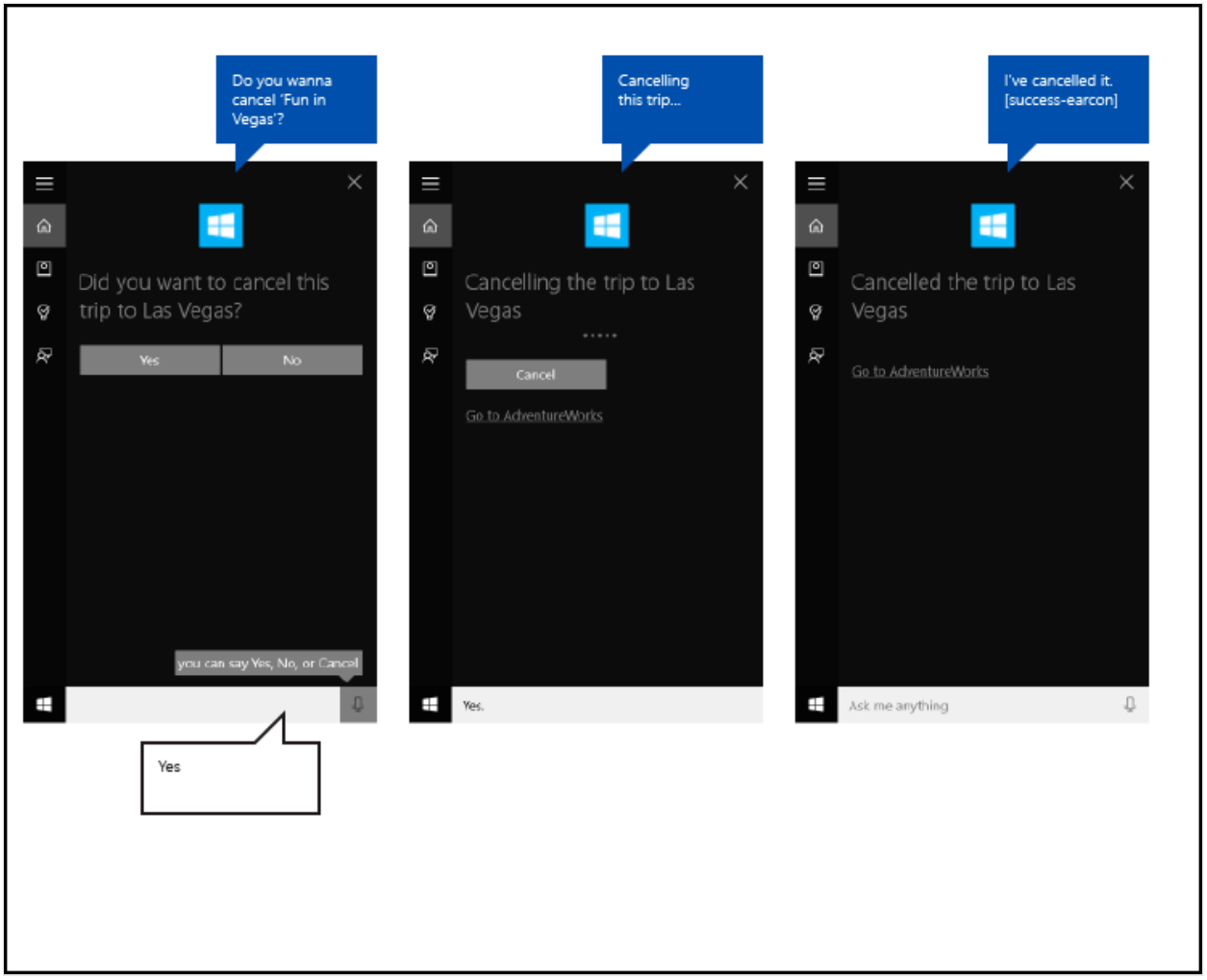
Be clear that the task hasn't already taken place if the app still needs to resolve the entity requested. For example, notice how we say "Looking for your trip" instead of "Cancelling your trip" because zero or more trips can be matched, and we don't know the result yet.

The GUI and TTS strings can be the same, but don't need to be. Try to keep the GUI string short to avoid truncation and duplication of other visual assets.

Expand table

TTS	GUI
Looking for your next Adventure Works trip.	Looking for your next trip...
Searching for your Adventure Works trip to Falls City.	Searching for trip to Falls City...

Progress



AdventureWorks "Cancel trip" progress

When a task takes a while between steps, your app needs to step in and update the user on what's happening on a progress screen. The app icon is displayed, and you must

provide both GUI and TTS progress strings to indicate that the task is underway.

You should provide a link to your app with launch parameters to start the app in the appropriate state. This lets the user view or complete the task themselves. **Cortana** provides the link text (such as, "Go to Adventure Works").

Progress screens will show for 5 seconds each, after which they must be followed by another screen or the task will time out.

These screens can follow a progress screen:

- Progress
- Confirmation (explicit, described later)
- Disambiguation
- Completion

GUI and TTS guidelines for progress screens

Use present tense.

Use an action verb that confirms the task is underway.

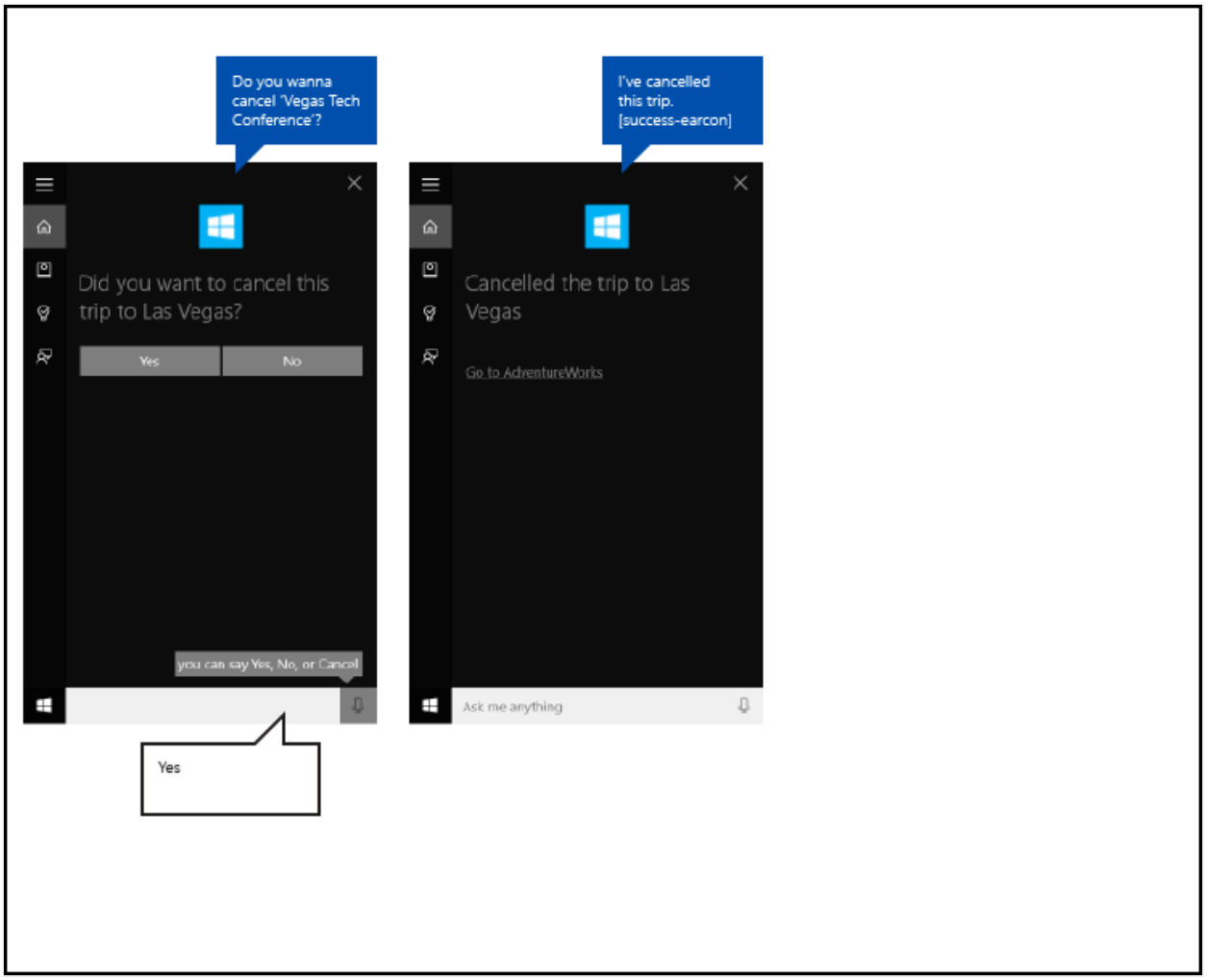
GUI: If the entity is shown, use a reference to it ("Cancelling this trip..."); if no entity is shown, explicitly call out the entity ("Cancelling 'Vegas Tech Conference'").

TTS: You should only include a TTS string on the first progress screen. If further progress screens are required, send an empty string, {}, as your TTS string, and provide a GUI string only.

Expand table

Conditions	TTS	GUI
ENTITY READ ON PRIOR TURN / ENTITY SHOWN ON DISPLAY	Cancelling this trip...	Cancelling this trip...
ENTITY NOT READ ON PRIOR TURN / ENTITY SHOWN ON DISPLAY	Cancelling your trip to Vegas...	Cancelling this trip...
ENTITY NOT READ ON PRIOR TURN / ENTITY NOT SHOWN	Cancelling your trip to Vegas...	Cancelling your trip to Vegas...

Confirmation



AdventureWorks "Cancel trip" confirmation

Some tasks can be implicitly confirmed by the nature of the user's command; others are potentially more sensitive and require explicit confirmation. Here are some guidelines for when to use explicit vs. implicit confirmation.

Both GUI and TTS strings on the confirmation screen are specified by your app, and the app icon, if provided, is shown instead of the **Cortana** avatar.

After the customer responds to the confirmation, your application must provide the next screen within 500 ms to avoid going to a progress screen.

Use explicit when...

- Content is leaving the user (such as, a text message, email, or social post)
- An action can't be undone (such as, making a purchase or deleting something)
- The result could be embarrassing (such as, calling the wrong person)
- More complex recognition is required (such as, open-ended transcription)

Use implicit when...

- Content is saved for the user only (such as, a note-to-self)
- There's an easy way to back out (such as, turning an alarm on or off)

- The task needs to be quick (such as, quickly capturing an idea before forgetting)
- Accuracy is high (such as, a simple menu)

GUI and TTS guidelines for confirmation screens

Use present tense.

Ask the user an unambiguous question that can be answered with "Yes" or "No". The question should explicitly confirm what the user is trying to do and there should be no other obvious options.

Provide a variation of the question for a re-prompt, in case the voice command is not understood the first time.

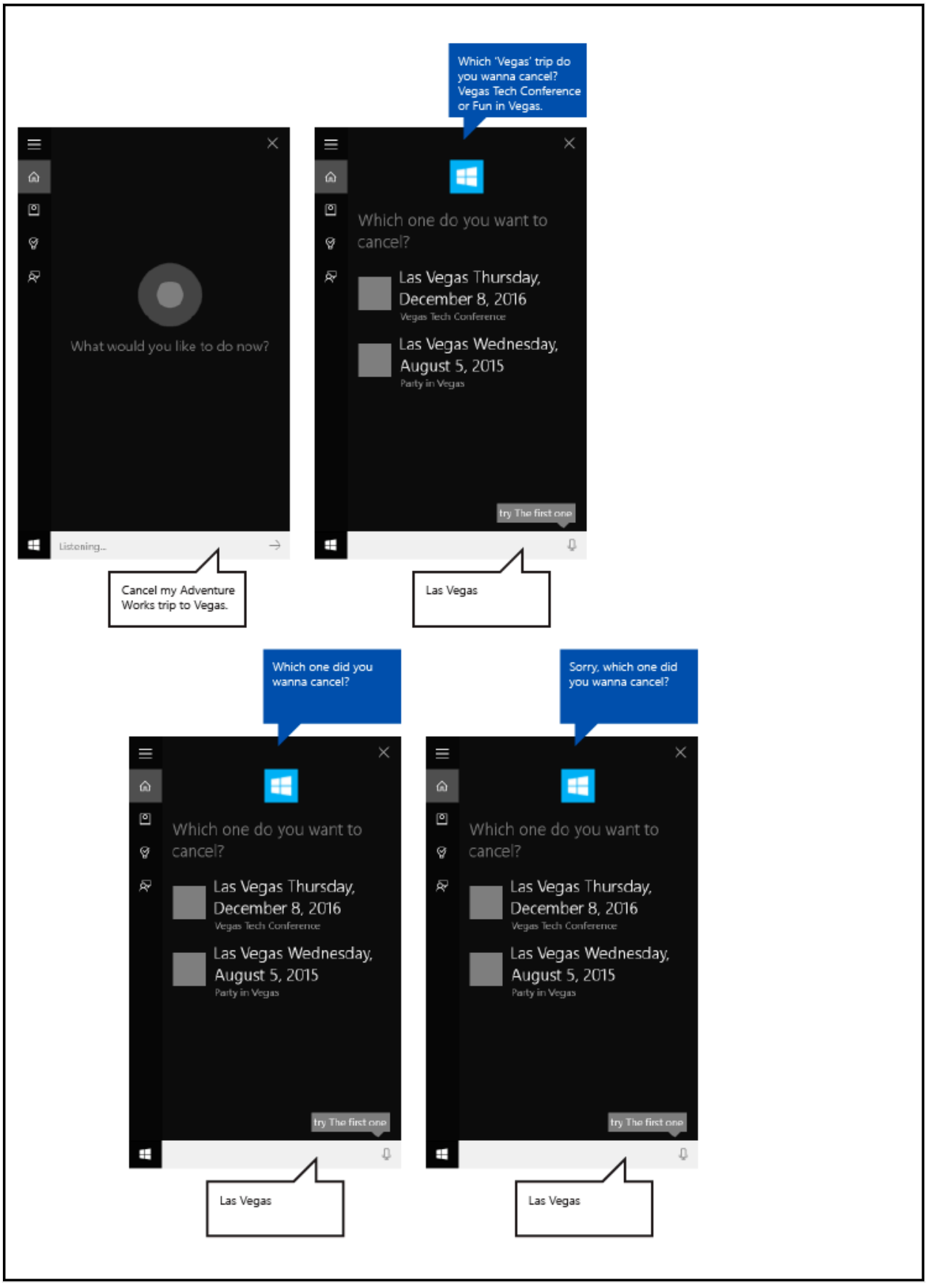
GUI: If the entity is shown, use a reference to it. If no entity is shown, explicitly call out the entity.

TTS: For clarity, always reference the specific item or entity, unless it was read out by the system on the previous turn.

Expand table

Conditions	TTS	GUI
ENTITY NOT READ ON PRIOR TURN / ENTITY SHOWN ON DISPLAY	Do you wanna cancel Vegas Tech Conference?	Cancel this trip?
ENTITY NOT READ ON PRIOR TURN / ENTITY NOT SHOWN	Do you wanna cancel Vegas Tech Conference?	Cancel Vegas Tech Conference?
ENTITY READ ON PRIOR TURN / ENTITY NOT SHOWN	Do you wanna cancel this trip?	Cancel this trip?
REPROMPT WITH ENTITY SHOWN	Did you wanna cancel this trip?	Did you want to cancel this trip?
REPROMPT WITH ENTITY NOT SHOWN	Did you wanna cancel this trip?	Did you want to cancel Vegas Tech Conference?

Disambiguation



AdventureWorks "Cancel trip" disambiguation

Some tasks might require the user to select from a list of entities to complete the task.

Both GUI and TTS strings on the disambiguation screen are specified by your app, and the app icon, if provided, is shown instead of the **Cortana** avatar.

After the customer responds to the disambiguation question, your application must provide the next screen within 500 ms to avoid going to a progress screen.

GUI and TTS guidelines for disambiguation screens

Use present tense.

Ask the user an unambiguous question that can be answered with the title or text line of any entity displayed.

Up to 10 entities can be displayed.

Each entity should have a unique title.

Provide a variation of the question for a re-prompt, in case the voice command is not understood the first time.

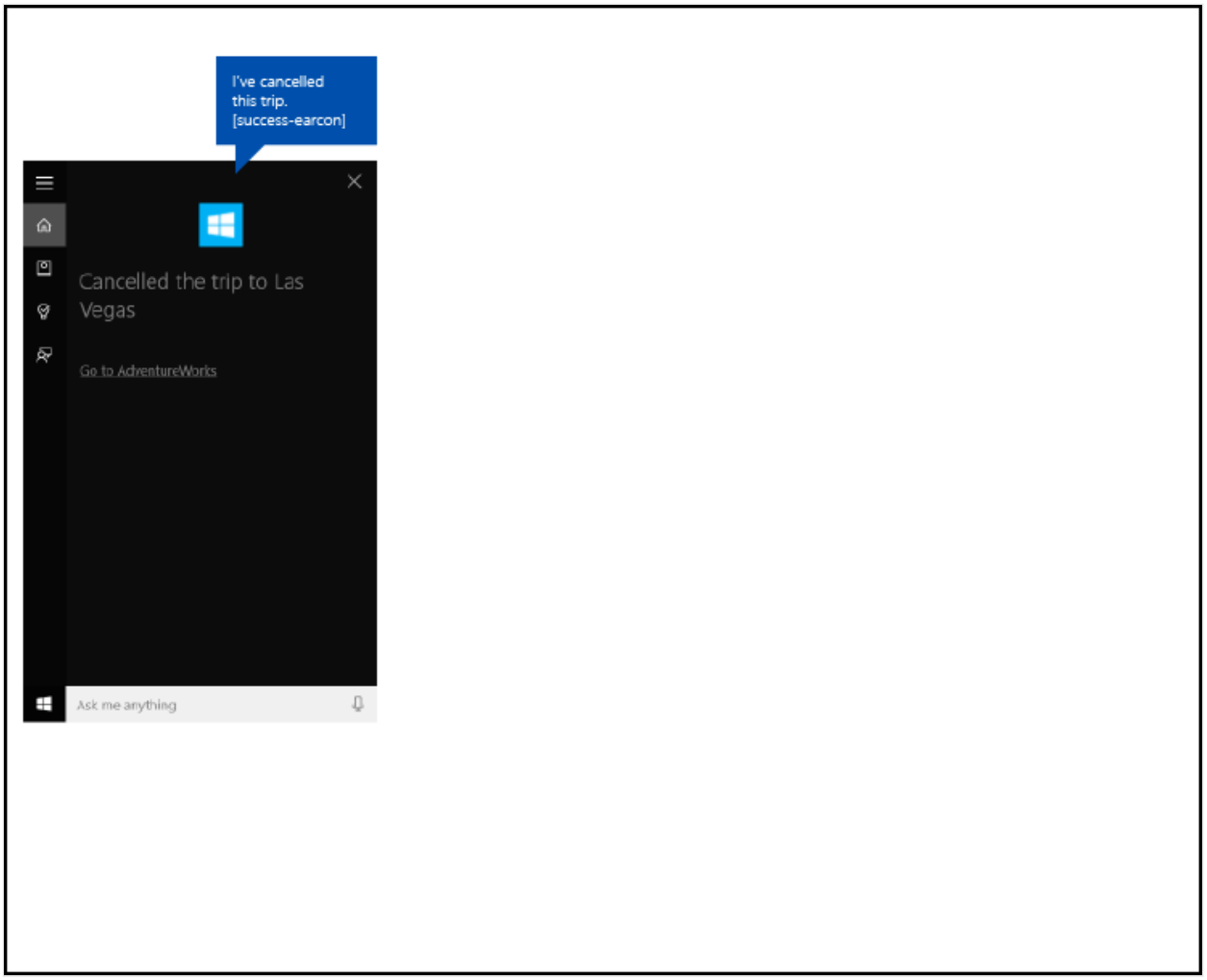
TTS: For clarity, always reference the specific item or entity, unless it was spoken on the previous turn.

TTS: Don't read out the entity list, unless there are three or fewer and they are short.

Expand table

Conditions	TTS	GUI
PROMPT - 3 OR FEWER ITEMS	Which Vegas trip do you wanna cancel? Vegas Tech Conference or Party in Vegas?	Which one do you want to cancel?
PROMPT - MORE THAN 3 ITEMS	Which Vegas trip do you wanna cancel?	Which one do you want to cancel?
REPROMPT	Which Vegas trip did you wanna cancel?	Which one do you want to cancel?

Completion



AdventureWorks "Cancel trip" completion

On successful task completion, your app should inform the user that the requested task was completed successfully.

Both GUI and TTS strings on the completion screen are specified by your app, and the app icon, if provided, is shown instead of the **Cortana** avatar.

You should provide a link to your app with launch parameters to start the app in the appropriate state. This lets the user view or complete the task themselves. **Cortana** provides the link text (such as, "Go to Adventure Works").

GUI and TTS guidelines for completion screens

Use past tense.

Use an action verb to explicitly state that the task has completed.

If the entity is shown, or it has been referenced on prior turn, only reference it.

Expand table

Conditions	TTS	GUI
ENTITY SHOWN / ENTITY READ ON PRIOR TURN	I've cancelled this trip.	Cancelled this trip.
ENTITY NOT SHOWN / ENTITY NOT READ ON PRIOR TURN	I've cancelled your Vegas Tech Conference trip.	Cancelled "Vegas Tech Conference."

Error

AdventureWorks "Cancel trip" error

When one of the following errors occur, **Cortana** displays the same generic error message.

- The app service terminates unexpectedly.
- **Cortana** fails to communicate with the app service.
- The app fails to provide a screen after **Cortana** shows a handoff screen or a progress screen for 5 seconds.

Related articles

- Cortana interactions in Windows apps
- VCD elements and attributes v1.2
- Cortana voice command sample

Activate a foreground app with voice commands through Cortana

Article • 06/24/2021

> ⚠ **Warning**
>
> This feature is no longer supported as of the Windows 10 May 2020 Update (version 2004, codename "20H1").

In addition to using voice commands within **Cortana** to access system features, you can also extend **Cortana** with features and functionality from your app. Using voice commands, your app can be activated to the foreground and an action or command executed within the app.

> ⓘ **Note**
>
> **Important APIs**
>
> - Windows.ApplicationModel.VoiceCommands
> - VCD elements and attributes v1.2

When an app handles a voice command in the foreground, it takes focus and Cortana is dismissed. If you prefer, you can activate your app and execute a command as a background task. In this case, Cortana retains focus and your app returns all feedback and results through the **Cortana** canvas and the **Cortana** voice.

Voice commands that require additional context or user input (such as sending a message to a specific contact) are best handled in a foreground app, while basic commands (such as listing upcoming trips) can be handled in **Cortana** through a background app.

If you want to activate an app in the background using voice commands, see Activate a background app in Cortana using voice commands.

> ⓘ **Note**
>
> A voice command is a single utterance with a specific intent, defined in a Voice Command Definition (VCD) file, directed at an installed app via **Cortana**.

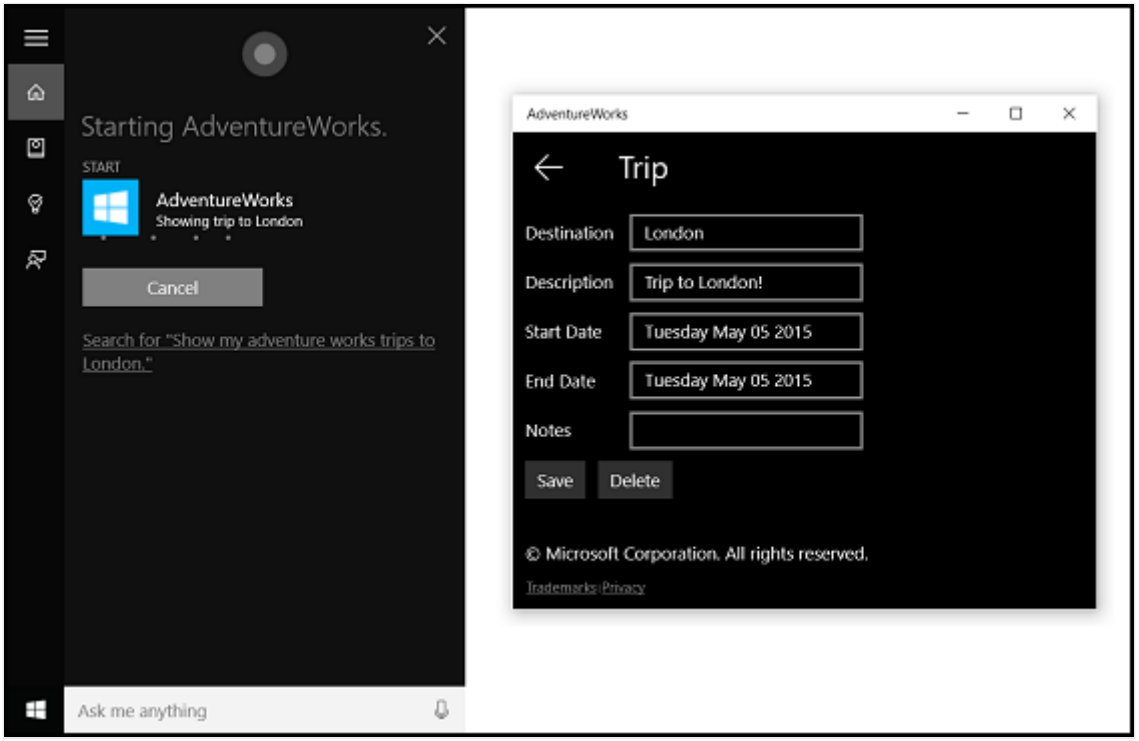
A VCD file defines one or more voice commands, each with a unique intent.

Voice command definitions can vary in complexity. They can support anything from a single, constrained utterance to a collection of more flexible, natural language utterances, all denoting the same intent.

To demonstrate foreground app features, we'll use a trip planning and management app named **Adventure Works** from the Cortana voice command sample .

To create a new **Adventure Works** trip without **Cortana**, a user would launch the app and navigate to the **New trip** page. To view an existing trip, a user would launch the app, navigate to the **Upcoming trips** page, and select the trip.

Using voice commands through **Cortana**, the user can instead just say, "Adventure Works add a trip" or "Add a trip on Adventure Works" to launch the app and navigate to the **New trip** page. In turn, saying "Adventure Works, show my trip to London" will launch the app and navigate to the **Trip** detail page, shown here.



These are the basic steps to add voice-command functionality and integrate Cortana with your app using speech or keyboard input:

1. Create a VCD file. This is an XML document that defines all the spoken commands that the user can say to initiate actions or invoke commands when activating your app. See **VCD elements and attributes v1.2**.
2. Register the command sets in the VCD file when the app is launched.
3. Handle the activation-by-voice-command, navigation within the app, and execution of the command.

> **Tip**
>
> **Prerequisites**
>
> If you're new to developing Universal Windows Platform (UWP) apps, have a look through these topics to get familiar with the technologies discussed here.
>
> - Create your first app
> - Learn about events with Events and routed events overview
>
> **User experience guidelines**
>
> See Cortana design guidelines for info about how to integrate your app with **Cortana** and Speech interactions for helpful tips on designing a useful and engaging speech-enabled app.

Create a new solution with project in Visual Studio

1. Launch Microsoft Visual Studio 2015.

   The Visual Studio 2015 Start page appears.

2. On the **File** menu, select **New** > **Project**.

   The **New Project** dialog appears. The left pane of the dialog lets you select the type of templates to display.

3. In the left pane, expand **Installed** > **Templates** > **Visual C#** > **Windows**, then pick the **Universal** template group. The dialog's center pane displays a list of project templates for Universal Windows Platform (UWP) apps.

4. In the center pane, select the **Blank App (Universal Windows)** template.

   The **Blank App** template creates a minimal UWP app that compiles and runs, but contains no user-interface controls or data. You add controls to the app over the course of this tutorial.

5. In the **Name** text box, type your project name. For this example, we use "AdventureWorks".

6. Click **OK** to create the project.

Microsoft Visual Studio creates your project and displays it in the **Solution Explorer**.

Add image assets to project and specify them in the app manifest

UWP apps can automatically select the most appropriate images based on specific settings and device capabilities (high contrast, effective pixels, locale, and so on). All you need to do is provide the images and ensure you use the appropriate naming convention and folder organization within the app project for the different resource versions. If you don't provide the recommended resource versions, accessibility, localization, and image quality can suffer, depending on the user's preferences, abilities, device type, and location.

For more detail on image resources for high contrast and scale factors, see Guidelines for tile and icon assets.

You name resources using qualifiers. Resource qualifiers are folder and filename modifiers that identify the context in which a particular version of a resource should be used.

The standard naming convention is `foldername/qualifiername-value[_qualifiername-value]/filename.qualifiername-value[_qualifiername-value].ext`. For example, `images/logo.scale-100_contrast-white.png`, which can be referred to in code using just the root folder and the filename: `images/logo.png`. See How to name resources using qualifiers.

We recommend that you mark the default language on string resource files (such as `en-US\resources.resw`) and the default scale factor on images (such as `logo.scale-100.png`), even if you do not currently plan to provide localized or multiple resolution resources. However, at a minimum, we recommend that you provide assets for 100, 200, and 400 scale factors.

> **Important**
>
> The app icon used in the title area of the **Cortana** canvas is the Square44x44Logo icon specified in the "Package.appxmanifest" file.

Create a VCD file

1. In Visual Studio, right-click your primary project name, select **Add** > **New Item**. Add an **XML File**.
2. Type a name for the VCD file (for this example, "AdventureWorksCommands.xml"), and click Add.
3. In **Solution Explorer**, select the VCD file.
4. In the **Properties** window, set **Build action** to **Content**, and then set **Copy to output directory** to **Copy if newer**.

Edit the VCD file

Add a **VoiceCommands** element with an **xmlns** attribute pointing to `https://schemas.microsoft.com/voicecommands/1.2`.

1. For each language supported by your app, create a CommandSet element that contains the voice commands supported by your app.

   You can declare multiple CommandSet elements, each with a different xml:lang attribute so your app to be used in different markets. For example, an app for the United States might have a CommandSet for English and a CommandSet for Spanish.

   > ⊗ **Caution**
   >
   > To activate an app and initiate an action using a voice command, the app must register a VCD file that contains a CommandSet with a language that matches the speech language selected by the user for their device. The speech language is located in **Settings** > **System** > **Speech** > **Speech Language**.

2. Add a **Command** element for each command you want to support. Each **Command** declared in a VCD file must include the following information:

   - An **AppName** attribute that your application uses to identify the voice command at runtime.
   - An **Example** element that contains a phrase describing how a user can invoke the command. **Cortana** shows this example when the user says "What can I say?", "Help", or they tap **See more**.
   - A **ListenFor** element that contains the words or phrases that your app recognizes as a command. Each **ListenFor** element can contain references to one or more **PhraseList** elements that contain specific words relevant to the command.

> **Note**
>
> **ListenFor** elements cannot be programmatically modified. However, **PhraseList** elements associated with **ListenFor** elements can be programmatically modified. Applications should modify the content of the **PhraseList** at runtime based on the data set generated as the user uses the app. See Dynamically modify Cortana VCD phrase lists.

A **Feedback** element that contains the text for **Cortana** to display and speak as the application is launched.

A **Navigate** element indicates that the voice command activates the app to the foreground. In this example, the `showTripToDestination` command is a foreground task.

A **VoiceCommandService** element indicates that the voice command activates the app in the background. The value of the **Target** attribute of this element should match the value of the **Name** attribute of the **uap:AppService** element in the package.appxmanifest file. In this example, the `whenIsTripToDestination` and `cancelTripToDestination` commands are background tasks that specify the name of the app service as "AdventureWorksVoiceCommandService".

For more detail, see the **VCD elements and attributes v1.2** reference.

Here's a portion of the **VCD** file that defines the en-us voice commands for the **Adventure Works** app.

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<VoiceCommands xmlns="https://schemas.microsoft.com/voicecommands/1.2">
  <CommandSet xml:lang="en-us" Name="AdventureWorksCommandSet_en-us">
    <AppName> Adventure Works </AppName>
    <Example> Show trip to London </Example>

    <Command Name="showTripToDestination">
      <Example> Show trip to London </Example>
      <ListenFor RequireAppName="BeforeOrAfterPhrase"> show [my] trip to
{destination} </ListenFor>
      <ListenFor RequireAppName="ExplicitlySpecified"> show [my]
{builtin:AppName} trip to {destination} </ListenFor>
      <Feedback> Showing trip to {destination} </Feedback>
      <Navigate />
    </Command>

    <Command Name="whenIsTripToDestination">
      <Example> When is my trip to Las Vegas?</Example>
      <ListenFor RequireAppName="BeforeOrAfterPhrase"> when is [my] trip to
```

```
{destination}</ListenFor>
      <ListenFor RequireAppName="ExplicitlySpecified"> when is [my]
{builtin:AppName} trip to {destination} </ListenFor>
      <Feedback> Looking for trip to {destination}</Feedback>
      <VoiceCommandService Target="AdventureWorksVoiceCommandService"/>
    </Command>

    <Command Name="cancelTripToDestination">
      <Example> Cancel my trip to Las Vegas </Example>
      <ListenFor RequireAppName="BeforeOrAfterPhrase"> cancel [my] trip to
{destination}</ListenFor>
      <ListenFor RequireAppName="ExplicitlySpecified"> cancel [my]
{builtin:AppName} trip to {destination} </ListenFor>
      <Feedback> Cancelling trip to {destination}</Feedback>
      <VoiceCommandService Target="AdventureWorksVoiceCommandService"/>
    </Command>

    <PhraseList Label="destination">
      <Item>London</Item>
      <Item>Las Vegas</Item>
      <Item>Melbourne</Item>
      <Item>Yosemite National Park</Item>
    </PhraseList>
  </CommandSet>
```

Install the VCD commands

Your app must run once to install the VCD.

> **Note**
>
> Voice command data is not preserved across app installations. To ensure the voice command data for your app remains intact, consider initializing your VCD file each time your app is launched or activated, or maintain a setting that indicates if the VCD is currently installed.

In the "app.xaml.cs" file:

1. Add the following using directive:

```
C#
```

```
using Windows.Storage;
```

2. Mark the "OnLaunched" method with the async modifier.

```
C#

protected async override void OnLaunched(LaunchActivatedEventArgs e)
```

3. Call InstallCommandDefinitionsFromStorageFileAsync in the OnLaunched handler to register the voice commands that the system should recognize.

In the Adventure Works sample, we first define a StorageFile object.

We then call GetFileAsync to initialize it with our "AdventureWorksCommands.xml" file.

This StorageFile object is then passed to InstallCommandDefinitionsFromStorageFileAsync.

```
C#

try
{
  // Install the main VCD.
  StorageFile vcdStorageFile =
  await Package.Current.InstalledLocation.GetFileAsync(
  @"AdventureWorksCommands.xml");

  await
Windows.ApplicationModel.VoiceCommands.VoiceCommandDefinitionManager.
InstallCommandDefinitionsFromStorageFileAsync(vcdStorageFile);

  // Update phrase list.
  ViewModel.ViewModelLocator locator =
App.Current.Resources["ViewModelLocator"] as ViewModel.ViewModelLocator;
  if(locator != null)
  {
     await locator.TripViewModel.UpdateDestinationPhraseList();
  }
}
catch (Exception ex)
{
  System.Diagnostics.Debug.WriteLine("Installing Voice Commands Failed: " +
ex.ToString());
}
```

Handle activation and execute voice commands

Specify how your app responds to subsequent voice command activations (after it has been launched at least once and the voice command sets have been installed).

1. Confirm that your app was activated by a voice command.

Override the Application.OnActivated event and check whether IActivatedEventArgs.Kind is VoiceCommand.

2. Determine the name of the command and what was spoken.

   Get a reference to a VoiceCommandActivatedEventArgs object from the IActivatedEventArgs and query the Result property for a SpeechRecognitionResult object.

   To determine what the user said, check the value of Text or the semantic properties of the recognized phrase in the SpeechRecognitionSemanticInterpretation dictionary.

3. Take the appropriate action in your app, such as navigating to the desired page.

For this example, we refer back to the VCD in Step 3: Edit the VCD file.

Once we get the speech-recognition result for the voice command, we get the command name from the first value in the RulePath array. As the VCD file defined more than one possible voice command, we need to compare the value against the command names in the VCD and take the appropriate action.

The most common action an application can take is to navigate to a page with content relevant to the context of the voice command. For this example, we navigate to a **TripPage** page and pass in the value of the voice command, how the command was input, and the recognized "destination" phrase (if applicable). Alternatively, the app could send a navigation parameter to the SpeechRecognitionResult when navigating to the page.

You can find out whether the voice command that launched your app was actually spoken, or whether it was typed in as text, from the SpeechRecognitionSemanticInterpretation.Properties dictionary using the **commandMode** key. The value of that key will be either "voice" or "text". If the value of the key is "voice", consider using speech synthesis (Windows.Media.SpeechSynthesis) in your app to provide the user with spoken feedback.

Use the SpeechRecognitionSemanticInterpretation.Properties to find out the content spoken in the **PhraseList** or **PhraseTopic** constraints of a **ListenFor** element. The dictionary key is the value of the **Label** attribute of the **PhraseList** or **PhraseTopic** element. Here, we show how to access the value of **{destination}** phrase.

C#

```
/// <summary>
/// Entry point for an application activated by some means other than normal
```

```
launching.
/// This includes voice commands, URI, share target from another app, and so
on.
///
/// NOTE:
/// A previous version of the VCD file might remain in place
/// if you modify it and update the app through the store.
/// Activations might include commands from older versions of your VCD.
/// Try to handle these commands gracefully.
/// </summary>
/// <param name="args">Details about the activation method.</param>
protected override void OnActivated(IActivatedEventArgs args)
{
    base.OnActivated(args);

    Type navigationToPageType;
    ViewModel.TripVoiceCommand? navigationCommand = null;

    // Voice command activation.
    if (args.Kind == ActivationKind.VoiceCommand)
    {
        // Event args can represent many different activation types.
        // Cast it so we can get the parameters we care about out.
        var commandArgs = args as VoiceCommandActivatedEventArgs;

        Windows.Media.SpeechRecognition.SpeechRecognitionResult
speechRecognitionResult = commandArgs.Result;

        // Get the name of the voice command and the text spoken.
        // See VoiceCommands.xml for supported voice commands.
        string voiceCommandName = speechRecognitionResult.RulePath[0];
        string textSpoken = speechRecognitionResult.Text;

        // commandMode indicates whether the command was entered using
speech or text.
        // Apps should respect text mode by providing silent (text)
feedback.
        string commandMode = this.SemanticInterpretation("commandMode",
speechRecognitionResult);

        switch (voiceCommandName)
        {
            case "showTripToDestination":
                // Access the value of {destination} in the voice command.
                string destination =
this.SemanticInterpretation("destination", speechRecognitionResult);

                // Create a navigation command object to pass to the page.
                navigationCommand = new ViewModel.TripVoiceCommand(
                    voiceCommandName,
                    commandMode,
                    textSpoken,
                    destination);

                // Set the page to navigate to for this voice command.
```

```
                navigationToPageType = typeof(View.TripDetails);
                break;
            default:
                // If we can't determine what page to launch, go to the
default entry point.
                navigationToPageType = typeof(View.TripListView);
                break;
        }
    }
    // Protocol activation occurs when a card is clicked within Cortana
(using a background task).
    else if (args.Kind == ActivationKind.Protocol)
    {
        // Extract the launch context. In this case, we're just using the
destination from the phrase set (passed
        // along in the background task inside Cortana), which makes no
attempt to be unique. A unique id or
        // identifier is ideal for more complex scenarios. We let the
destination page check if the
        // destination trip still exists, and navigate back to the trip list
if it doesn't.
        var commandArgs = args as ProtocolActivatedEventArgs;
        Windows.Foundation.WwwFormUrlDecoder decoder = new
Windows.Foundation.WwwFormUrlDecoder(commandArgs.Uri.Query);
        var destination = decoder.GetFirstValueByName("LaunchContext");

        navigationCommand = new ViewModel.TripVoiceCommand(
                                "protocolLaunch",
                                "text",
                                "destination",
                                destination);

        navigationToPageType = typeof(View.TripDetails);
    }
    else
    {
        // If we were launched via any other mechanism, fall back to the
main page view.
        // Otherwise, we'll hang at a splash screen.
        navigationToPageType = typeof(View.TripListView);
    }

    // Repeat the same basic initialization as OnLaunched() above, taking
into account whether
    // or not the app is already active.
    Frame rootFrame = Window.Current.Content as Frame;

    // Do not repeat app initialization when the Window already has content,
    // just ensure that the window is active.
    if (rootFrame == null)
    {
        // Create a frame to act as the navigation context and navigate to
the first page.
        rootFrame = new Frame();
        App.NavigationService = new NavigationService(rootFrame);
```

```
        rootFrame.NavigationFailed += OnNavigationFailed;

        // Place the frame in the current window.
        Window.Current.Content = rootFrame;
    }

    // Since we're expecting to always show a details page, navigate even if
    // a content frame is in place (unlike OnLaunched).
    // Navigate to either the main trip list page, or if a valid voice
command
    // was provided, to the details page for that trip.
    rootFrame.Navigate(navigationToPageType, navigationCommand);

    // Ensure the current window is active
    Window.Current.Activate();
}

/// <summary>
/// Returns the semantic interpretation of a speech result.
/// Returns null if there is no interpretation for that key.
/// </summary>
/// <param name="interpretationKey">The interpretation key.</param>
/// <param name="speechRecognitionResult">The speech recognition result to
get the semantic interpretation from.</param>
/// <returns></returns>
private string SemanticInterpretation(string interpretationKey,
SpeechRecognitionResult speechRecognitionResult)
{
  return
speechRecognitionResult.SemanticInterpretation.Properties[interpretationKey]
.FirstOrDefault();
}
```

Related articles

- Cortana interactions in Windows apps
- Activate a background app in Cortana using voice commands
- VCD elements and attributes v1.2
- Cortana design guidelines
- Cortana voice command sample

Dynamically modify Cortana VCD phrase lists

Article • 06/24/2021

> ⚠ **Warning**
>
> This feature is no longer supported as of the Windows 10 May 2020 Update (version 2004, codename "20H1").

Access and update the list of supported phrases (**PhraseList** elements) in a Voice Command Definition (VCD) file at run time using the speech recognition result.

> ⓘ **Note**
>
> A voice command is a single utterance with a specific intent, defined in a Voice Command Definition (VCD) file, directed at an installed app via **Cortana**.
>
> A VCD file defines one or more voice commands, each with a unique intent.
>
> Voice command definitions can vary in complexity. They can support anything from a single, constrained utterance to a collection of more flexible, natural language utterances, all denoting the same intent.

Dynamically modifying a phrase list at run time is useful if the voice command is specific to a task involving some kind of user-defined, or transient, app data.

> ⓘ **Note**
>
> **Important APIs**
>
> - Windows.ApplicationModel.VoiceCommands
> - VCD elements and attributes v1.2

As an example, let's say you have a travel app where users can enter destinations, and you want users to be able to start the app by saying the app name followed by "Show trip to <destination>". In the **ListenFor** element itself, you would specify something like: `<ListenFor> Show trip to {destination} </ListenFor>`, where "destination" is the value of the **Label** attribute for the **PhraseList**.

Updating the phrase list at run time eliminates the need to create a separate **ListenFor** element for each possible destination. Instead, you can dynamically populate **PhraseList** with destinations specified by the user as they enter their itineraries.

For more info about **PhraseList** and other VCD elements, see the **VCD elements and attributes v1.2** reference.

> **Tip**
>
> **Prerequisites**
>
> If you're new to developing Universal Windows Platform (UWP) apps, have a look through these topics to get familiar with the technologies discussed here.
>
> - Create your first app
> - Learn about events with Events and routed events overview
>
> **User experience guidelines**
>
> See Cortana design guidelines for info about how to integrate your app with **Cortana** and Speech interactions for helpful tips on designing a useful and engaging speech-enabled app.

Identify the command and update the phrase list

Here's an example VCD file that defines a **Command** "showTripToDestination" and a **PhraseList** that defines three options for destination in our **Adventure Works** travel app. As the user saves and deletes destinations in the app, the app updates the options in the **PhraseList**.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<VoiceCommands xmlns="https://schemas.microsoft.com/voicecommands/1.1">
  <CommandSet xml:lang="en-us" Name="AdventureWorksCommandSet_en-us">
    <AppName> Adventure Works, </AppName>
    <Example> Show trip to London </Example>

    <Command Name="showTripToDestination">
      <Example> show trip to London  </Example>
      <ListenFor> show trip to {destination} </ListenFor>
      <Feedback> Showing trip to {destination} </Feedback>
      <Navigate/>
```

```
    </Command>

    <PhraseList Label="destination">
      <Item> London </Item>
      <Item> Dallas </Item>
      <Item> New York </Item>
    </PhraseList>

  </CommandSet>

<!-- Other CommandSets for other languages -->

</VoiceCommands>
```

To update a **PhraseList** element in the VCD file, get the **CommandSet** element that contains the phrase list. Use the **Name** attribute of that **CommandSet** element (**Name** must be unique in the VCD file) as a key to access the VoiceCommandManager.InstalledCommandSets property and get the VoiceCommandSet reference.

After you've identified the command set, get a reference to the phrase list that you want to modify and call the SetPhraseListAsync method; use the **Label** attribute of the **PhraseList** element and an array of strings as the new content of the phrase list.

> **Note**
>
> If you modify a phrase list, the entire phrase list is replaced. If you want to insert new items into a phrase list, you must specify both the existing items and the new items in the call to SetPhraseListAsync.

In this example, we update the **PhraseList** shown in the previous example with an additional destination to Phoenix.

CSharp

```
Windows.ApplicationModel.VoiceCommands.VoiceCommandDefinition.VoiceCommandSe
t commandSetEnUs;

if (Windows.ApplicationModel.VoiceCommands.VoiceCommandDefinitionManager.
      InstalledCommandSets.TryGetValue(
        "AdventureWorksCommandSet_en-us", out commandSetEnUs))
{
  await commandSetEnUs.SetPhraseListAsync(
    "destination", new string[] {"London", "Dallas", "New York",
"Phoenix"});
}
```

Remarks

Using a **PhraseList** to constrain the recognition is appropriate for a relatively small set or words. When the set of words is too large (hundreds of words, for example), or shouldn't be constrained at all, use the **PhraseTopic** element and a **Subject** element to refine the relevance of speech-recognition results to improve scalability.

In our example, we have a **PhraseTopic** with a **Scenario** of "Search", further refined by a **Subject** of "City\State".

XML

```
<?xml version="1.0" encoding="utf-8"?>
<VoiceCommands xmlns="https://schemas.microsoft.com/voicecommands/1.1">
  <CommandSet xml:lang="en-us" Name="AdventureWorksCommandSet_en-us">
    <AppName> Adventure Works, </AppName>
    <Example> Show trip to London </Example>

    <Command Name="showTripToDestination">
      <Example> show trip to London  </Example>
      <ListenFor> show trip to {destination} </ListenFor>
      <Feedback> Showing trip to {destination} </Feedback>
      <Navigate/>
    </Command>

    <PhraseList Label="destination">
      <Item> London </Item>
      <Item> Dallas </Item>
      <Item> New York </Item>
    </PhraseList>

    <PhraseTopic Label="destination" Scenario="Search">
      <Subject>City/State</Subject>
    </PhraseTopic>

  </CommandSet>
```

Related articles

- Cortana interactions in Windows apps
- VCD elements and attributes v1.2
- Activate a foreground app with voice commands through Cortana
- Activate a background app in Cortana using voice commands
- Cortana design guidelines
- Cortana voice command sample

Activate a background app in Cortana using voice commands

Article • 06/24/2021

> ⚠ **Warning**
>
> This feature is no longer supported as of the Windows 10 May 2020 Update (version 2004, codename "20H1").

In addition to using voice commands within **Cortana** to access system features, you may also extend **Cortana** with features and functionality from your app (as a background task) using voice commands that specify an action or command to run. When an app handles a voice command in the background, it does not take focus. Instead, it returns all feedback and results through the **Cortana** canvas and the **Cortana** voice.

> ⓘ **Note**
>
> **Important APIs**
>
> - Windows.ApplicationModel.VoiceCommands
> - VCD elements and attributes v1.2

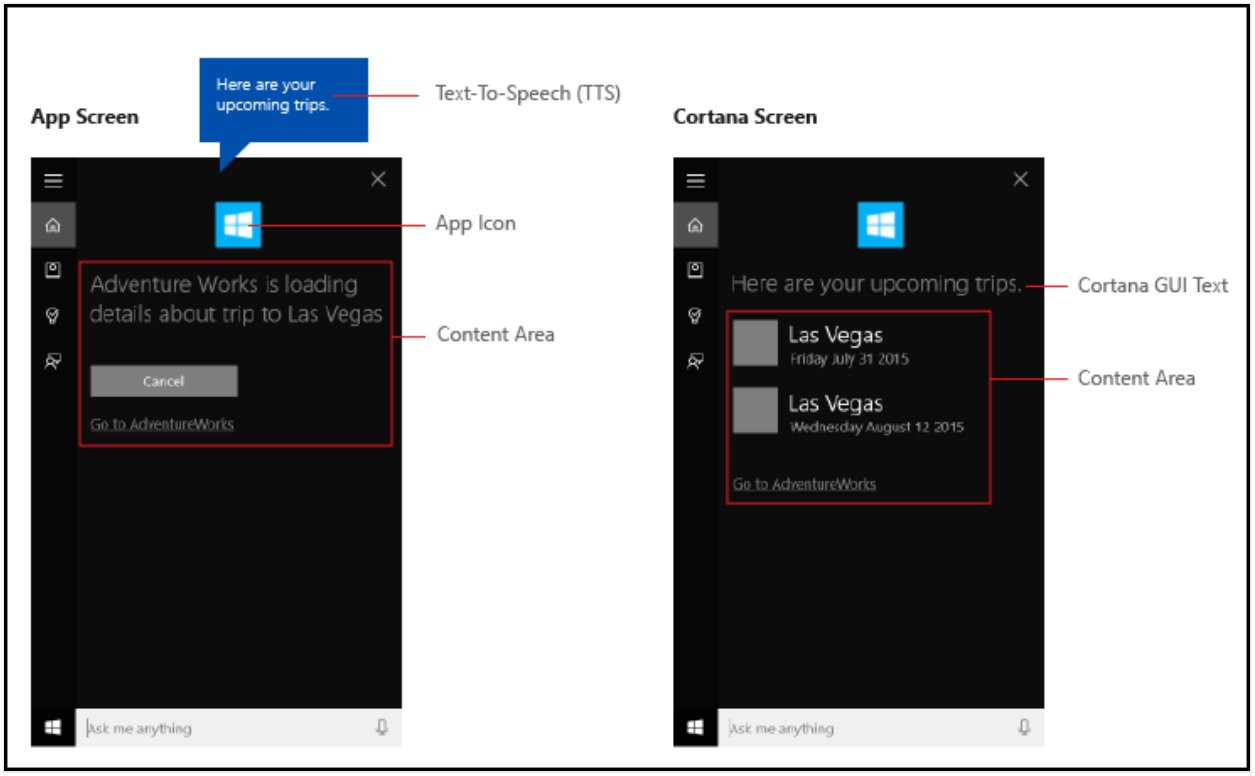
Apps may be activated to the foreground (the app takes focus) or activated in the background (**Cortana** retains focus), depending on the complexity of the interaction. For example, voice commands that require additional context or user input (such as sending a message to a specific contact) are best handled in a foreground app, while basic commands (such as listing upcoming trips) may be handled in **Cortana** through a background app.

If you want to activate an app to the foreground using voice commands, see Activate a foreground app with voice commands through Cortana.

> ⓘ **Note**
>
> A voice command is a single utterance with a specific intent, defined in a Voice Command Definition (VCD) file, directed at an installed app via **Cortana**.
>
> A VCD file defines one or more voice commands, each with a unique intent.

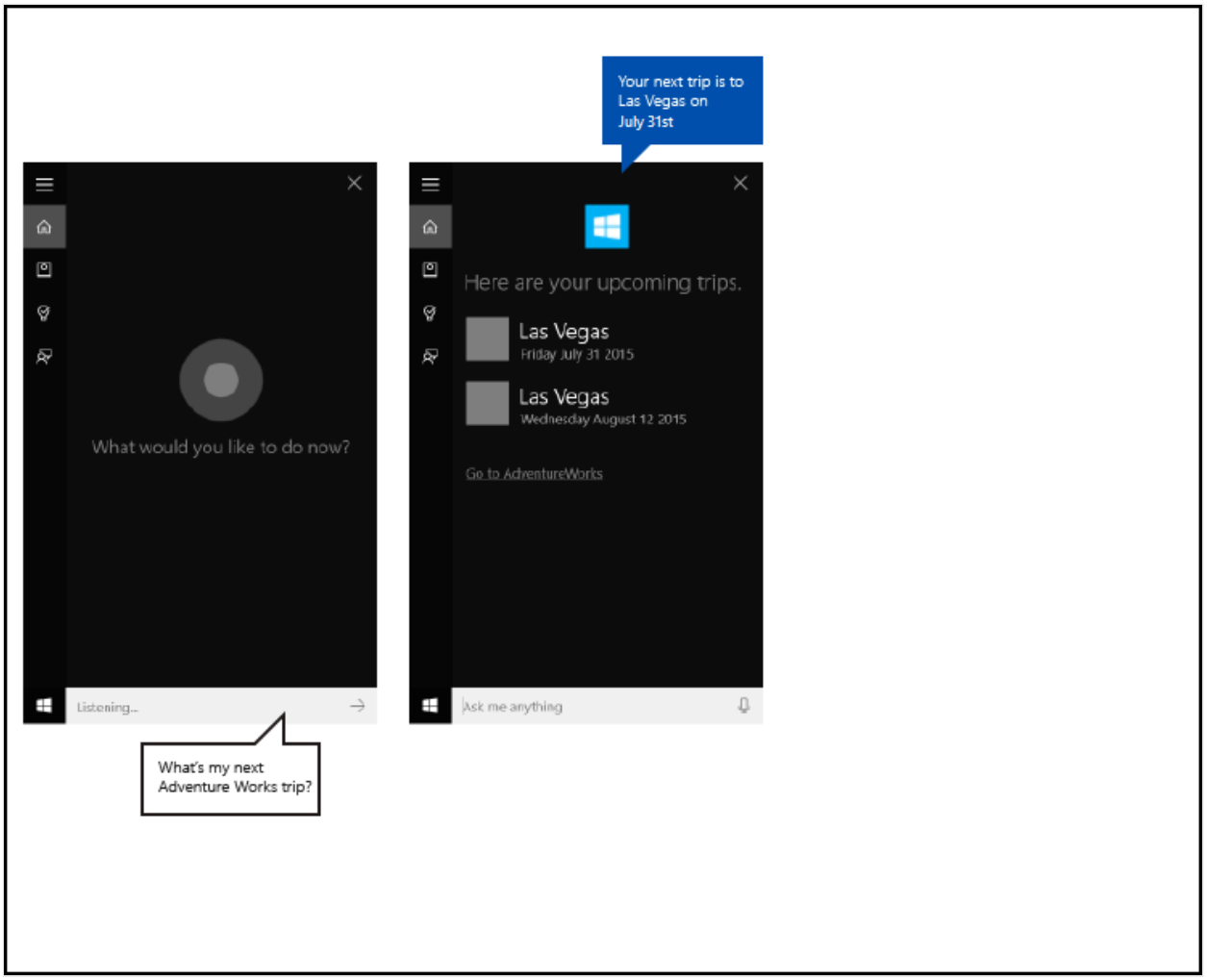
Voice command definitions can vary in complexity. They can support anything from a single, constrained utterance to a collection of more flexible, natural language utterances, all denoting the same intent.

We use a trip planning and management app named **Adventure Works** integrated into the **Cortana** UI, shown here, to demonstrate many of the concepts and features we discuss. For more info, see the Cortana voice command sample.



To view an **Adventure Works** trip without **Cortana**, a user would launch the app and navigate to the **Upcoming trips** page.

Using voice commands through **Cortana** to launch your app in the background, the user may instead just say, `Adventure Works, when is my trip to Las Vegas?`. Your app handles the command and **Cortana** displays results along with your app icon and other app info, if provided.



The following basic steps add voice-command functionality and extend **Cortana** with background functionality from your app using speech or keyboard input.

1. Create an app service (see **Windows.ApplicationModel.AppService**) that **Cortana** invokes in the background.
2. Create a VCD file. The VCD file is an XML document that defines all the spoken commands that the user may say to initiate actions or invoke commands when activating your app. See **VCD elements and attributes v1.2**.
3. Register the command sets in the VCD file when the app is launched.
4. Handle the background activation of the app service and the running of the voice command.
5. Display and speak the appropriate feedback to the voice command within **Cortana**.

Tip

Prerequisites

If you're new to developing Universal Windows Platform (UWP) apps, have a look through these topics to get familiar with the technologies discussed here.

- Create your first app

- Learn about events with **Events and routed events overview**

User experience guidelines

See **Cortana design guidelines** for info about how to integrate your app with **Cortana** and **Speech interactions** for helpful tips on designing a useful and engaging speech-enabled app.

Create a New Solution with a Primary Project in Visual Studio

1. Launch Microsoft Visual Studio 2015.
   The Visual Studio 2015 Start page appears.

2. On the **File** menu, select **New** > **Project**.
   The **New Project** dialog appears. The left pane of the dialog lets you select the type of templates to display.

3. In the left pane, expand **Installed** > **Templates** > **Visual C#** > **Windows**, then pick the **Universal** template group. The center pane of the dialog displays a list of project templates for Universal Windows Platform (UWP) apps.

4. In the center pane, select the **Blank App (Universal Windows)** template.
   The **Blank App** template creates a minimal UWP app that compiles and runs. The **Blank App** template includes no user-interface controls or data. You add controls to the app using this page as a guide.

5. In the **Name** text box, type your project name. Example: Use `AdventureWorks`.

6. Click on the **OK** button to create the project.
   Microsoft Visual Studio creates your project and displays it in the **Solution Explorer**.

Add Image Assets to Primary Project and Specify them in the App Manifest

UWP apps should automatically select the most appropriate images. The selection is based upon specific settings and device capabilities (high contrast, effective pixels, locale, and so on). You must provide the images and ensure that you use the appropriate naming convention and folder organization within your app project for the different resource versions.

If you do not provide the recommended resource versions, then the user experience may suffer in the following ways.

- Accessibility
- Localization
- Image quality
  The resource versions are used to adapt the following changes in the user experience.
- User preferences
- Abilities
- Device type
- Location

For more detail about image resources for high contrast and scale factors, visit the Guidelines for tile and icon assets page located at msdn.microsoft.com/windows/uwp/controls-and-patterns/tiles-and-notifications-app-assets.

You must name resources using qualifiers. Resource qualifiers are folder and filename modifiers that identify the context in which a particular version of a resource should be used.

The standard naming convention is `foldername/qualifiername-value[_qualifiername-value]/filename.qualifiername-value[_qualifiername-value].ext`.
Example: `images/logo.scale-100_contrast-white.png`, which may refer to code using just the root folder and the filename: `images/logo.png`.
For more information, visit the How to name resources using qualifiers page located at msdn.microsoft.com/library/windows/apps/xaml/hh965324.aspx.

Microsoft recommends that you mark the default language on string resource files (such as `en-US\resources.resw`) and the default scale factor on images (such as `logo.scale-100.png`), even if you do not currently plan to provide localized or multiple resolution resources. However, at a minimum, Microsoft recommends that you provide assets for 100, 200, and 400 scale factors.

> **ⓘ Important**
>
> The app icon used in the title area of the **Cortana** canvas is the Square44x44Logo icon specified in the `Package.appxmanifest` file.
> You may also specify an icon for each entry in the content area of the **Cortana** canvas. Valid image sizes for the results icons are:

- 68w x 68h
- 68w x 92h
- 280w x 140h

The content tile is not validated until a VoiceCommandResponse object is passed to the VoiceCommandServiceConnection class. If you pass a VoiceCommandResponse object to **Cortana** that includes a content tile with an image that does not adhere to these size ratios, then an exception may occur.

Example: The **Adventure Works** app (`VoiceCommandService\\AdventureWorksVoiceCommandService.cs`) specifies a simple, grey square (`GreyTile.png`) on the VoiceCommandContentTile class using the TitleWith68x68IconAndText tile template. The logo variants are located in `VoiceCommandService\\Images`, and are retrieved using the GetFileFromApplicationUriAsync method.

C#

```
var destinationTile = new VoiceCommandContentTile();

destinationTile.ContentTileType =
VoiceCommandContentTileType.TitleWith68x68IconAndText;
destinationTile.Image = await StorageFile.GetFileFromApplicationUriAsync(
    new Uri("ms-appx:///AdventureWorks.VoiceCommands/Images/GreyTile.png")
);
```

Create an App Service Project

1. Right-click on your Solution name, select **New** > **Project**.
2. Under **Installed** > **Templates** > **Visual C#** > **Windows** > **Universal**, select **Windows Runtime Component**. The **Windows Runtime Component** is the component that implements the app service (Windows.ApplicationModel.AppService).
3. Type a name for the project and click on the **OK** button.
   Example: `VoiceCommandService`.
4. In **Solution Explorer**, select the `VoiceCommandService` project and rename the `Class1.cs` file generated by Visual Studio. Example: The **Adventure Works** uses `AdventureWorksVoiceCommandService.cs`.

5. Click on the **Yes** button; when asked if you want to rename all occurrences of `Class1.cs`.

6. In the `AdventureWorksVoiceCommandService.cs` file:
   a. Add the following using directive.
   `using Windows.ApplicationModel.Background;`
   b. When you create a new project, the project name is used as the default root namespace in all files. Rename the namespace to nest the app service code under the primary project. Example: `namespace AdventureWorks.VoiceCommands`.
   c. Right-click on the app service project name in Solution Explorer and select **Properties**.
   d. On the **Library** tab, update the **Default namespace** field with this same value. Example: `AdventureWorks.VoiceCommands`).
   e. Create a new class that implements the **IBackgroundTask** interface. This class requires a **Run** method, which is the entry point when Cortana recognizes the voice command.

Example: A basic background task class from the **Adventure Works** app.

> **Note**
>
> The background task class itself, as well as all classes in the background task project, must be sealed public classes.

C#

```
namespace AdventureWorks.VoiceCommands
{
    ...

    /// <summary>
    /// The VoiceCommandService implements the entry point for all
voice commands.
    /// The individual commands supported are described in the VCD xml
file.
    /// The service entry point is defined in the appxmanifest.
    /// </summary>
    public sealed class AdventureWorksVoiceCommandService :
IBackgroundTask
    {
        ...

        /// <summary>
        /// The background task entrypoint.
        ///
        /// Background tasks must respond to activation by Cortana
```

```
within 0.5 second, and must
        /// report progress to Cortana every 5 seconds (unless Cortana
is waiting for user
        /// input). There is no running time limit on the background
task managed by Cortana,
        /// but developers should use plmdebug
(https://msdn.microsoft.com/library/windows/hardware/jj680085%28v=vs.85
%29.aspx)
        /// on the Cortana app package in order to prevent Cortana
timing out the task during
        /// debugging.
        ///
        /// The Cortana UI is dismissed if Cortana loses focus.
        /// The background task is also dismissed even if being
debugged.
        /// Use of Remote Debugging is recommended in order to debug
background task behaviors.
        /// Open the project properties for the app package (not the
background task project),
        /// and enable Debug -> "Do not launch, but debug my code when
it starts".
        /// Alternatively, add a long initial progress screen, and
attach to the background task process while it runs.
        /// </summary>
        /// <param name="taskInstance">Connection to the hosting
background service process.</param>
        public void Run(IBackgroundTaskInstance taskInstance)
        {
          //
          // TODO: Insert code
          //
          //
    }
  }
}
```

7. Declare your background task as an **AppService** in the app manifest.
   a. In **Solution Explorer**, right-click on the `Package.appxmanifest` file and select **View Code**.
   b. Find the Application element.
   c. Add an Extensions element to the Application element.
   d. Add a uap:Extension element to the Extensions element.
   e. Add a `Category` attribute to the `uap:Extension` element and set the value of the `Category` attribute to `windows.appService`.
   f. Add an `EntryPoint` attribute to the `uap: Extension` element and set the value of the `EntryPoint` attribute to the name of the class that implements IBackgroundTask.
   Example: `AdventureWorks.VoiceCommands.AdventureWorksVoiceCommandService`.
   g. Add a uap:AppService element to the `uap:Extension` element.

h. Add a `Name` attribute to the uap:AppService element and set the value of the `Name` attribute to a name for the app service, in this case `AdventureWorksVoiceCommandService`.
i. Add a second uap:Extension element to the Extensions element.
j. Add a `Category` attribute to this uap:Extension element and set the value of the `Category` attribute to `windows.personalAssistantLaunch`.

Example: A manifest from the Adventure Works app.

XML

```
<Package>
    <Applications>
        <Application>

            <Extensions>
                <uap:Extension Category="windows.appService"
EntryPoint="CortanaBack1.VoiceCommands.AdventureWorksVoiceCommandServic
e">
                    <uap:AppService
Name="AdventureWorksVoiceCommandService"/>
                </uap:Extension>
                <uap:Extension
Category="windows.personalAssistantLaunch"/>
            </Extensions>

        <Application>
    <Applications>
</Package>
```

8. Add this app service project as a reference in the primary project.
   a. Right-click on the **References**.
   b. Select **Add Reference...**.
   c. In the **Reference Manager** dialog, expand **Projects** and select the app service project.
   d. Click on the **OK** button.

Create a VCD File

1. In Visual Studio, right-click on your primary project name, select **Add** > **New Item**. Add an **XML File**.
2. Type a name for the VCD file.
   Example: `AdventureWorksCommands.xml`.
3. Click on the **Add** button.
4. In **Solution Explorer**, select the VCD file.

5. In the **Properties** window, set **Build action** to **Content**, and then set **Copy to output directory** to **Copy if newer**.

Edit the VCD File

1. Add a `VoiceCommands` element with an `xmlns` attribute pointing to `https://schemas.microsoft.com/voicecommands/1.2`.

2. For each language supported by your app, create a CommandSet element that includes the voice commands supported by your app.
   You are able to declare multiple CommandSet elements, each with a different xml:lang attribute so your app to be used in different markets. For example, an app for the United States might have a CommandSet for English and a CommandSet for Spanish.

   > **ⓘ Important**
   >
   > To activate an app and initiate an action using a voice command, the app must register a VCD file that includes a **CommandSet** element with a language that matches the speech language indicated in the device of the user. The speech language is located in **Settings** > **System** > **Speech** > **Speech Language**.

3. Add a `Command` element for each command you want to support.
   Each `Command` declared in a **VCD** file must include this information:

   - A `Name` attribute that your application uses to identify the voice command at runtime.
   - An `Example` element that includes a phrase describing how a user invokes the command. **Cortana** shows the example when the user says `What can I say?`, `Help`, or taps **See more**.
   - A `ListenFor` element that includes the words or phrases that your app recognizes as a command. Each `ListenFor` element may contain references to one or more `PhraseList` elements that contain specific words relevant to the command.

     > **ⓘ Note**

> `ListenFor` elements must not be programmatically modified. However, `PhraseList` elements associated with `ListenFor` elements may be programmatically modified. Applications should modify the content of the `PhraseList` element at runtime based on the data set generated as the user uses the app.
>
> For more information, see Dynamically modify Cortana VCD phrase lists.

- A `Feedback` element that includes the text for **Cortana** to display and speak as the application is launched.

A `Navigate` element indicates that the voice command activates the app to the foreground. In this example, the `showTripToDestination` command is a foreground task.

A `VoiceCommandService` element indicates that the voice command activates the app in the background. The value of the `Target` attribute of this element should match the value of the `Name` attribute of the uap:AppService element in the package.appxmanifest file. In this example, the `whenIsTripToDestination` and `cancelTripToDestination` commands are background tasks that specify the name of the app service as `AdventureWorksVoiceCommandService`.

For more detail, see the **VCD elements and attributes v1.2** reference.

Example: A portion of the **VCD** file that defines the `en-us` voice commands for the **Adventure Works** app.

XML

```
<?xml version="1.0" encoding="utf-8" ?>
<VoiceCommands xmlns="https://schemas.microsoft.com/voicecommands/1.2">
<CommandSet xml:lang="en-us" Name="AdventureWorksCommandSet_en-us">
    <AppName> Adventure Works </AppName>
    <Example> Show trip to London </Example>

    <Command Name="showTripToDestination">
        <Example> Show trip to London </Example>
        <ListenFor RequireAppName="BeforeOrAfterPhrase"> show [my] trip to
{destination} </ListenFor>
        <ListenFor RequireAppName="ExplicitlySpecified"> show [my]
{builtin:AppName} trip to {destination} </ListenFor>
        <Feedback> Showing trip to {destination} </Feedback>
        <Navigate />
    </Command>

    <Command Name="whenIsTripToDestination">
```

```
        <Example> When is my trip to Las Vegas?</Example>
        <ListenFor RequireAppName="BeforeOrAfterPhrase"> when is [my] trip
to {destination}</ListenFor>
        <ListenFor RequireAppName="ExplicitlySpecified"> when is [my]
{builtin:AppName} trip to {destination} </ListenFor>
        <Feedback> Looking for trip to {destination}</Feedback>
        <VoiceCommandService Target="AdventureWorksVoiceCommandService"/>
    </Command>

    <Command Name="cancelTripToDestination">
        <Example> Cancel my trip to Las Vegas </Example>
        <ListenFor RequireAppName="BeforeOrAfterPhrase"> cancel [my] trip to
{destination}</ListenFor>
        <ListenFor RequireAppName="ExplicitlySpecified"> cancel [my]
{builtin:AppName} trip to {destination} </ListenFor>
        <Feedback> Cancelling trip to {destination}</Feedback>
        <VoiceCommandService Target="AdventureWorksVoiceCommandService"/>
    </Command>

    <PhraseList Label="destination">
        <Item>London</Item>
        <Item>Las Vegas</Item>
        <Item>Melbourne</Item>
        <Item>Yosemite National Park</Item>
    </PhraseList>
</CommandSet>
```

Install the VCD Commands

Your app must run once to install the VCD.

> ⓘ **Note**
>
> Voice command data is not preserved across app installations. To ensure the voice command data for your app remains intact, consider initializing your VCD file each time your app is launched or activated, or maintain a setting that indicates if the VCD is currently installed.

In the `app.xaml.cs` file:

1. Add the following using directive:

   C#

   ```
   using Windows.Storage;
   ```

2. Mark the `OnLaunched` method with the async modifier.

C#

```
protected async override void OnLaunched(LaunchActivatedEventArgs e)
```

3. Call the InstallCommandDefinitionsFromStorageFileAsync method in the OnLaunched handler to register the voice commands that should be recognized.
   Example: The Adventure Works app defines a StorageFile object.
   Example: Call the GetFileAsync method to initialize the StorageFile object with the `AdventureWorksCommands.xml` file.
   The StorageFile object is then passed to InstallCommandDefinitionsFromStorageFileAsync method.

C#

```
try {
   // Install the main VCD.
   StorageFile vcdStorageFile = await
Package.Current.InstalledLocation.GetFileAsync(
         @"AdventureWorksCommands.xml"
   );

   await
Windows.ApplicationModel.VoiceCommands.VoiceCommandDefinitionManager.In
stallCommandDefinitionsFromStorageFileAsync(vcdStorageFile);

   // Update phrase list.
   ViewModel.ViewModelLocator locator =
App.Current.Resources["ViewModelLocator"] as
ViewModel.ViewModelLocator;
   if(locator != null) {
         await locator.TripViewModel.UpdateDestinationPhraseList();
     }
 }
 catch (Exception ex) {
     System.Diagnostics.Debug.WriteLine("Installing Voice Commands
Failed: " + ex.ToString());
 }
```

Handle Activation

Specify how your app responds to subsequent voice command activations.

> **Note**

You must launch your app at least once after the voice command sets have been installed.

1. Confirm that your app was activated by a voice command.

   Override the Application.OnActivated event and check whether **IActivatedEventArgs**.**Kind** is **VoiceCommand**.

2. Determine the name of the command and what was spoken.

   Get a reference to a VoiceCommandActivatedEventArgs object from the **IActivatedEventArgs** and query the Result property for a SpeechRecognitionResult object.

   To determine what the user said, check the value of **Text** or the semantic properties of the recognized phrase in the SpeechRecognitionSemanticInterpretation dictionary.

3. Take the appropriate action in your app, such as navigating to the desired page.

   > **Note**
   >
   > If you need to refer to your VCD, visit the **Edit the VCD File** section.

   After receiving the speech-recognition result for the voice command, you get the command name from the first value in the RulePath array. Since the VCD file defines more than one possible voice command, you must verify that the value matches the command names in the VCD and take the appropriate action.

   The most common action for an application is to navigate to a page with content relevant to the context of the voice command.
   Example: Open the **TripPage** page and pass in the value of the voice command, how the command was input, and the recognized destination phrase (if applicable). Alternatively, the app may send a navigation parameter to the **SpeechRecognitionResult** when navigating to the **TripPage** page.

   You are able to find out whether the voice command that launched your app was actually spoken, or whether it was typed in as text, from the SpeechRecognitionSemanticInterpretation.Properties dictionary using the **commandMode** key. The value of that key will be either `voice` or `text`. If the value of the key is `voice`, consider using speech synthesis (**Windows.Media.SpeechSynthesis**) in your app to provide the user with spoken feedback.

Use the SpeechRecognitionSemanticInterpretation.Properties to find out the content spoken in the `PhraseList` or `PhraseTopic` constraints of a `ListenFor` element. The dictionary key is the value of the `Label` attribute of the `PhraseList` or `PhraseTopic` element. Example: The following code for How to access the value of **{destination}** phrase.

C#

```
/// <summary>
/// Entry point for an application activated by some means other than
normal launching.
/// This includes voice commands, URI, share target from another app,
and so on.
///
/// NOTE:
/// A previous version of the VCD file might remain in place
/// if you modify it and update the app through the store.
/// Activations might include commands from older versions of your VCD.
/// Try to handle these commands gracefully.
/// </summary>
/// <param name="args">Details about the activation method.</param>
protected override void OnActivated(IActivatedEventArgs args) {
    base.OnActivated(args);

    Type navigationToPageType;
    ViewModel.TripVoiceCommand? navigationCommand = null;

    // Voice command activation.
    if (args.Kind == ActivationKind.VoiceCommand) {
        // Event args may represent many different activation types.
        // Cast the args so that you only get useful parameters out.
        var commandArgs = args as VoiceCommandActivatedEventArgs;

        Windows.Media.SpeechRecognition.SpeechRecognitionResult
speechRecognitionResult = commandArgs.Result;

        // Get the name of the voice command and the text spoken.
        // See VoiceCommands.xml for supported voice commands.
        string voiceCommandName = speechRecognitionResult.RulePath[0];
        string textSpoken = speechRecognitionResult.Text;

        // commandMode indicates whether the command was entered using
speech or text.
        // Apps should respect text mode by providing silent (text)
feedback.
        string commandMode = this.SemanticInterpretation("commandMode",
speechRecognitionResult);

        switch (voiceCommandName) {
            case "showTripToDestination":
                // Access the value of {destination} in the voice
command.
```

```
                string destination =
this.SemanticInterpretation("destination", speechRecognitionResult);

                // Create a navigation command object to pass to the
page.
                navigationCommand = new ViewModel.TripVoiceCommand(
                    voiceCommandName,
                    commandMode,
                    textSpoken,
                    destination
                );

                // Set the page to navigate to for this voice command.
                navigationToPageType = typeof(View.TripDetails);
                break;
            default:
                // If not able to determine what page to launch, then
go to the default entry point.
                navigationToPageType = typeof(View.TripListView);
                break;
        }
    }
    // Protocol activation occurs when a card is selected within
Cortana (using a background task).
    else if (args.Kind == ActivationKind.Protocol) {
        // Extract the launch context. In this case, use the
destination from the phrase set (passed
        // along in the background task inside Cortana), which makes no
attempt to be unique. A unique id or
        // identifier is ideal for more complex scenarios. The
destination page is left to check if the
        // destination trip still exists, and navigate back to the trip
list if it does not.
        var commandArgs = args as ProtocolActivatedEventArgs;
        Windows.Foundation.WwwFormUrlDecoder decoder = new
Windows.Foundation.WwwFormUrlDecoder(commandArgs.Uri.Query);
        var destination = decoder.GetFirstValueByName("LaunchContext");

        navigationCommand = new ViewModel.TripVoiceCommand(
            "protocolLaunch",
            "text",
            "destination",
            destination
        );

        navigationToPageType = typeof(View.TripDetails);
    }
    else {
        // If launched using any other mechanism, fall back to the main
page view.
        // Otherwise, the app will freeze at a splash screen.
        navigationToPageType = typeof(View.TripListView);
    }

    // Repeat the same basic initialization as OnLaunched() above,
```

```
taking into account whether
    // or not the app is already active.
    Frame rootFrame = Window.Current.Content as Frame;

    // Do not repeat app initialization when the Window already has
content,
    // just ensure that the window is active.
    if (rootFrame == null) {
        // Create a frame to act as the navigation context and navigate
to the first page.
        rootFrame = new Frame();
        App.NavigationService = new NavigationService(rootFrame);

        rootFrame.NavigationFailed += OnNavigationFailed;

        // Place the frame in the current window.
        Window.Current.Content = rootFrame;
    }

    // Since the expectation is to always show a details page, navigate
even if
    // a content frame is in place (unlike OnLaunched).
    // Navigate to either the main trip list page, or if a valid voice
command
    // was provided, to the details page for that trip.
    rootFrame.Navigate(navigationToPageType, navigationCommand);

    // Ensure the current window is active
    Window.Current.Activate();
}

/// <summary>
/// Returns the semantic interpretation of a speech result.
/// Returns null if there is no interpretation for that key.
/// </summary>
/// <param name="interpretationKey">The interpretation key.</param>
/// <param name="speechRecognitionResult">The speech recognition result
to get the semantic interpretation from.</param>
/// <returns></returns>
private string SemanticInterpretation(string interpretationKey,
SpeechRecognitionResult speechRecognitionResult) {
    return
speechRecognitionResult.SemanticInterpretation.Properties[interpretatio
nKey].FirstOrDefault();
}
```

Handle the Voice Command in the App Service

Process the voice command in the app service.

1. Add the following using directives to your voice command service file.
   Example: `AdventureWorksVoiceCommandService.cs`.

   C#

   ```
   using Windows.ApplicationModel.VoiceCommands;
   using Windows.ApplicationModel.Resources.Core;
   using Windows.ApplicationModel.AppService;
   ```

2. Take a service deferral so your app service is not terminated while handling the voice command.

3. Confirm that your background task is running as an app service activated by a voice command.
   a. Cast the **IBackgroundTaskInstance.TriggerDetails** to **Windows.ApplicationModel.AppService.AppServiceTriggerDetails**.
   b. Check that **IBackgroundTaskInstance.TriggerDetails.Name** is the name of the app service in the `Package.appxmanifest` file.

4. Use **IBackgroundTaskInstance.TriggerDetails** to create a **VoiceCommandServiceConnection** to **Cortana** to retrieve the voice command.

5. Register an event handler for **VoiceCommandServiceConnection**. **VoiceCommandCompleted** to receive notification when the app service is closed due to a user cancellation.

6. Register an event handler for the **IBackgroundTaskInstance.Canceled** to receive notification when the app service is closed due to an unexpected failure.

7. Determine the name of the command and what was spoken.
   a. Use the **VoiceCommand.CommandName** property to determine the name of the voice command.
   b. To determine what the user said, check the value of **Text** or the semantic properties of the recognized phrase in the SpeechRecognitionSemanticInterpretation dictionary.

8. Take the appropriate action in your app service.

9. Display and speak the feedback to the voice command using **Cortana**.
   a. Determine the strings that you want **Cortana** to display and speak to the user in response to the voice command and create a VoiceCommandResponse object. For guidance on how to select the feedback strings that **Cortana** shows and speaks, see Cortana design guidelines.

b. Use the VoiceCommandServiceConnection instance to report progress or completion to **Cortana** by calling ReportProgressAsync or ReportSuccessAsync with the `VoiceCommandServiceConnection` object.

> **Note**
>
> If you need to refer to your VCD, visit the Edit the VCD File section.

C#

```
public sealed class VoiceCommandService : IBackgroundTask {
    private BackgroundTaskDeferral serviceDeferral;
    VoiceCommandServiceConnection voiceServiceConnection;

    public async void Run(IBackgroundTaskInstance taskInstance) {
        //Take a service deferral so the service isn&#39;t terminated.
        this.serviceDeferral = taskInstance.GetDeferral();

        taskInstance.Canceled += OnTaskCanceled;

        var triggerDetails = taskInstance.TriggerDetails as
AppServiceTriggerDetails;

        if (triggerDetails != null &amp;&amp;
            triggerDetails.Name ==
"AdventureWorksVoiceServiceEndpoint") {
            try {
                voiceServiceConnection =

VoiceCommandServiceConnection.FromAppServiceTriggerDetails(
                    triggerDetails);
                voiceServiceConnection.VoiceCommandCompleted +=
                VoiceCommandCompleted;

                VoiceCommand voiceCommand = await
                voiceServiceConnection.GetVoiceCommandAsync();

                switch (voiceCommand.CommandName) {
                    case "whenIsTripToDestination":
                        {
                            var destination =
                            voiceCommand.Properties["destination"][0];

SendCompletionMessageForDestination(destination);
                            break;
                        }

                        // As a last resort, launch the app in the
foreground.
                    default:
                        LaunchAppInForeground();
```

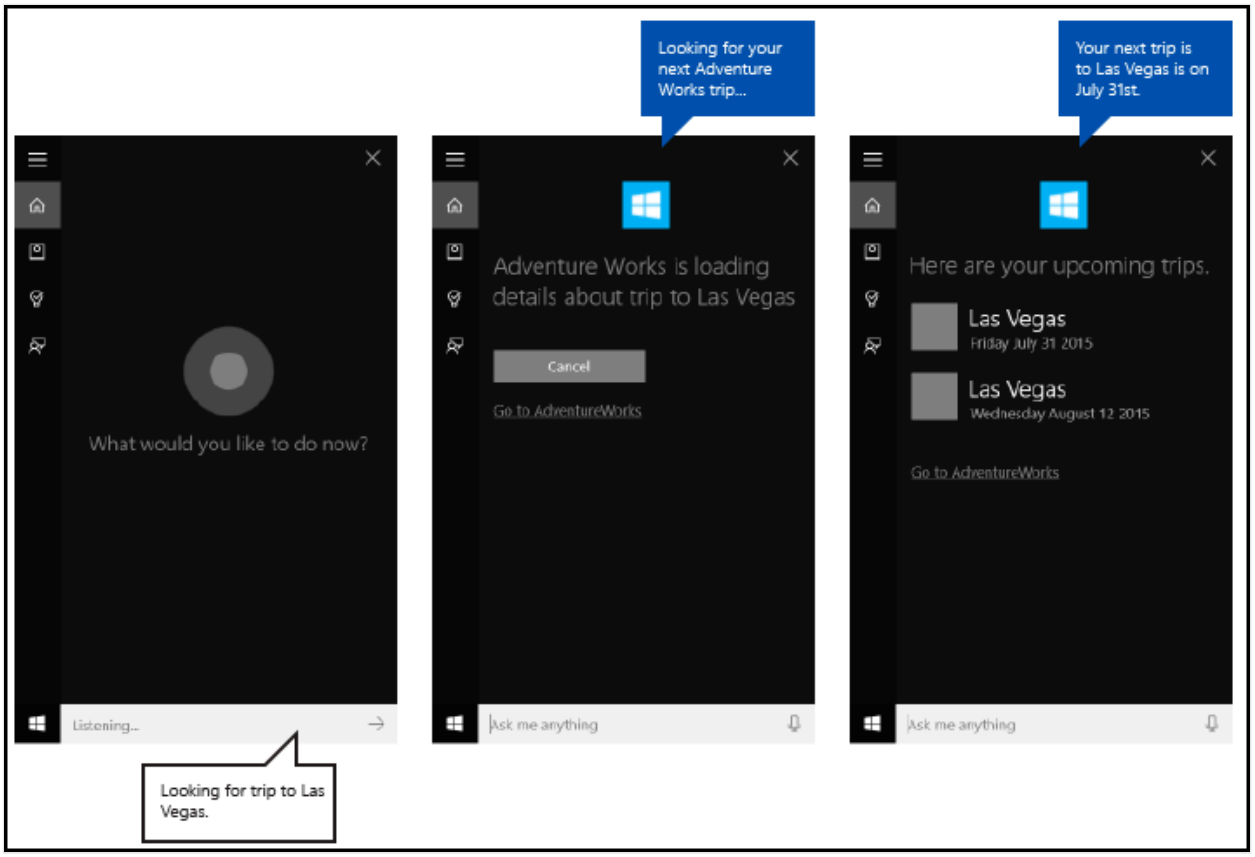
```
                    break;
                }
            }
            finally {
                if (this.serviceDeferral != null) {
                    // Complete the service deferral.
                    this.serviceDeferral.Complete();
                }
            }
        }
    }

    private void VoiceCommandCompleted(VoiceCommandServiceConnection
sender,
        VoiceCommandCompletedEventArgs args) {
        if (this.serviceDeferral != null) {
            // Insert your code here.
            // Complete the service deferral.
            this.serviceDeferral.Complete();
        }
    }

    private async void SendCompletionMessageForDestination(
        string destination) {
        // Take action and determine when the next trip to destination
        // Insert code here.

        // Replace the hardcoded strings used here with strings
        // appropriate for your application.

        // First, create the VoiceCommandUserMessage with the strings
        // that Cortana will show and speak.
        var userMessage = new VoiceCommandUserMessage();
        userMessage.DisplayMessage = "Here's your trip.";
        userMessage.SpokenMessage = "Your trip to Vegas is on August
3rd.";

        // Optionally, present visual information about the answer.
        // For this example, create a VoiceCommandContentTile with an
        // icon and a string.
        var destinationsContentTiles = new
List<VoiceCommandContentTile>();

        var destinationTile = new VoiceCommandContentTile();
        destinationTile.ContentTileType =
            VoiceCommandContentTileType.TitleWith68x68IconAndText;
        // The user taps on the visual content to launch the app.
        // Pass in a launch argument to enable the app to deep link to
a
        // page relevant to the item displayed on the content tile.
        destinationTile.AppLaunchArgument =
            string.Format("destination={0}", "Las Vegas");
        destinationTile.Title = "Las Vegas";
        destinationTile.TextLine1 = "August 3rd 2015";
        destinationsContentTiles.Add(destinationTile);
```

```
        // Create the VoiceCommandResponse from the userMessage and
list
        // of content tiles.
        var response = VoiceCommandResponse.CreateResponse(
            userMessage, destinationsContentTiles);

        // Cortana displays a "Go to app_name" link that the user
        // taps to launch the app.
        // Pass in a launch to enable the app to deep link to a page
        // relevant to the voice command.
        response.AppLaunchArgument = string.Format(
            "destination={0}", "Las Vegas");

        // Ask Cortana to display the user message and content tile and
        // also speak the user message.
        await voiceServiceConnection.ReportSuccessAsync(response);
    }

    private async void LaunchAppInForeground() {
        var userMessage = new VoiceCommandUserMessage();
        userMessage.SpokenMessage = "Launching Adventure Works";

        var response =
VoiceCommandResponse.CreateResponse(userMessage);

        // When launching the app in the foreground, pass an app
        // specific launch parameter to indicate what page to show.
        response.AppLaunchArgument = "showAllTrips=true";

        await voiceServiceConnection.RequestAppLaunchAsync(response);
    }
}
```

Once activated, the app service has 0.5 second to call ReportSuccessAsync. **Cortana** shows and says a feedback string.

> **Note**
>
> You are able to declare a **Feedback** string in the VCD file. The string does not affect the UI text displayed on the Cortana canvas, it only affects the text spoken by **Cortana**.

If the app takes longer than 0.5 second to make the call, **Cortana** inserts a hand-off screen, as shown here. **Cortana** displays the hand-off screen until the application calls **ReportSuccessAsync**, or for up to 5 seconds. If the app service does not call **ReportSuccessAsync**, or any of the VoiceCommandServiceConnection methods that provide **Cortana** with information, the user receives an error message and the app service is canceled.



Related articles

- Cortana interactions in Windows apps
- Activate a foreground app with voice commands through Cortana
- VCD elements and attributes v1.2
- Cortana design guidelines
- Cortana voice command sample

Interact with a background app in Cortana

Article • 06/24/2021

> ⚠ **Warning**
>
> This feature is no longer supported as of the Windows 10 May 2020 Update (version 2004, codename "20H1").

Enable user interaction with a background app, through speech and text input in the **Cortana** canvas, while executing a voice command.

> ⓘ **Note**
>
> **Important APIs**
>
> - Windows.ApplicationModel.VoiceCommands
> - VCD elements and attributes v1.2

Cortana supports a complete turn-by-turn workflow with your app. This workflow is defined by your app, and can support functionality such as:

- Successful completion
- Hand-off
- Progress
- Confirmation
- Disambiguation
- Error

Composing feedback strings

> 💡 **Tip**
>
> **Prerequisites**
>
> If you're new to developing Universal Windows Platform (UWP) apps, have a look through these topics to get familiar with the technologies discussed here.
>
> - Create your first app

- Learn about events with **Events and routed events overview**

User experience guidelines

See **Cortana design guidelines** for info about how to integrate your app with **Cortana** and **Speech interactions** for helpful tips on designing a useful and engaging speech-enabled app.

Compose the feedback strings that are both displayed and spoken by **Cortana**.

The Cortana design guidelines provides recommendations on composing strings for **Cortana**.

Content cards can provide additional context for the user and help you keep feedback strings concise.

Cortana supports the following content card templates (only one template can be used on the completion screen):

- Title only
- Title with up to three lines of text
- Title with image
- Title with image and up to three lines of text

The image can be:

- 68w x 68h
- 68w x 92h
- 280w x 140h

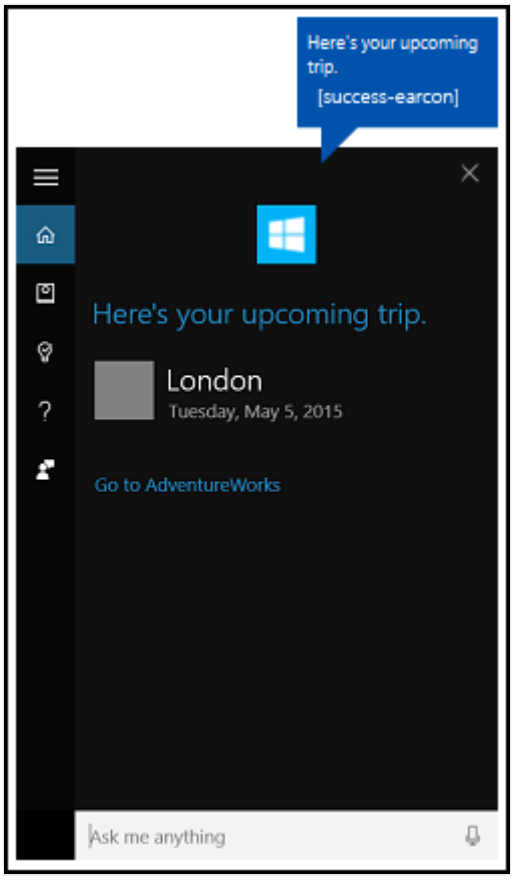
You can also let users launch your app in the foreground by clicking either a card or the text link to your app.

Completion screen

A completion screen provides the user with information about the completed voice command task.

Tasks that take less than 500 milliseconds for your app to respond, and require no additional information from the user, are completed without further interaction with **Cortana**. Cortana simply displays the completion screen.

Here, we use the **Adventure Works** app to show the completion screen for a voice command request to display upcoming trips to London.



The voice command is defined in AdventureWorksCommands.xml:

XML

```
<Command Name="whenIsTripToDestination">
  <Example> When is my trip to Las Vegas?</Example>
  <ListenFor RequireAppName="BeforeOrAfterPhrase"> when is [my] trip to
{destination}</ListenFor>
  <ListenFor RequireAppName="ExplicitlySpecified"> when is [my]
{builtin:AppName} trip to {destination} </ListenFor>
  <Feedback> Looking for trip to {destination}</Feedback>
  <VoiceCommandService Target="AdventureWorksVoiceCommandService"/>
</Command>
```

AdventureWorksVoiceCommandService.cs contains the completion message method:

C#

```
/// <summary>
/// Show details for a single trip, if the trip can be found.
/// This demonstrates a simple response flow in Cortana.
/// </summary>
/// <param name="destination">The destination, expected to be in the phrase
list.</param>
private async Task SendCompletionMessageForDestination(string destination)
{
    // If this operation is expected to take longer than 0.5 seconds, the
task must
```

```
    // supply a progress response to Cortana before starting the operation,
and
    // updates must be provided at least every 5 seconds.
    string loadingTripToDestination = string.Format(
                cortanaResourceMap.GetValue("LoadingTripToDestination",
cortanaContext).ValueAsString,
                destination);
    await ShowProgressScreen(loadingTripToDestination);
    Model.TripStore store = new Model.TripStore();
    await store.LoadTrips();

    // Query for the specified trip.
    // The destination should be in the phrase list. However, there might be
    // multiple trips to the destination. We pick the first.
    IEnumerable<Model.Trip> trips = store.Trips.Where(p => p.Destination ==
destination);

    var userMessage = new VoiceCommandUserMessage();
    var destinationsContentTiles = new List<VoiceCommandContentTile>();
    if (trips.Count() == 0)
    {
        string foundNoTripToDestination = string.Format(
                cortanaResourceMap.GetValue("FoundNoTripToDestination",
cortanaContext).ValueAsString,
                destination);
        userMessage.DisplayMessage = foundNoTripToDestination;
        userMessage.SpokenMessage = foundNoTripToDestination;
    }
    else
    {
        // Set plural or singular title.
        string message = "";
        if (trips.Count() > 1)
        {
            message = cortanaResourceMap.GetValue("PluralUpcomingTrips",
cortanaContext).ValueAsString;
        }
        else
        {
            message = cortanaResourceMap.GetValue("SingularUpcomingTrip",
cortanaContext).ValueAsString;
        }
        userMessage.DisplayMessage = message;
        userMessage.SpokenMessage = message;

        // Define a tile for each destination.
        foreach (Model.Trip trip in trips)
        {
            int i = 1;

            var destinationTile = new VoiceCommandContentTile();

            destinationTile.ContentTileType =
VoiceCommandContentTileType.TitleWith68x68IconAndText;
            destinationTile.Image = await
```

```
StorageFile.GetFileFromApplicationUriAsync(new Uri("ms-
appx:///AdventureWorks.VoiceCommands/Images/GreyTile.png"));

            destinationTile.AppLaunchArgument = trip.Destination;
            destinationTile.Title = trip.Destination;
            if (trip.StartDate != null)
            {
                destinationTile.TextLine1 =
trip.StartDate.Value.ToString(dateFormatInfo.LongDatePattern);
            }
            else
            {
                destinationTile.TextLine1 = trip.Destination + " " + i;
            }

            destinationsContentTiles.Add(destinationTile);
            i++;
        }
    }

    var response = VoiceCommandResponse.CreateResponse(userMessage,
destinationsContentTiles);

    if (trips.Count() > 0)
    {
        response.AppLaunchArgument = destination;
    }

    await voiceServiceConnection.ReportSuccessAsync(response);
}
```

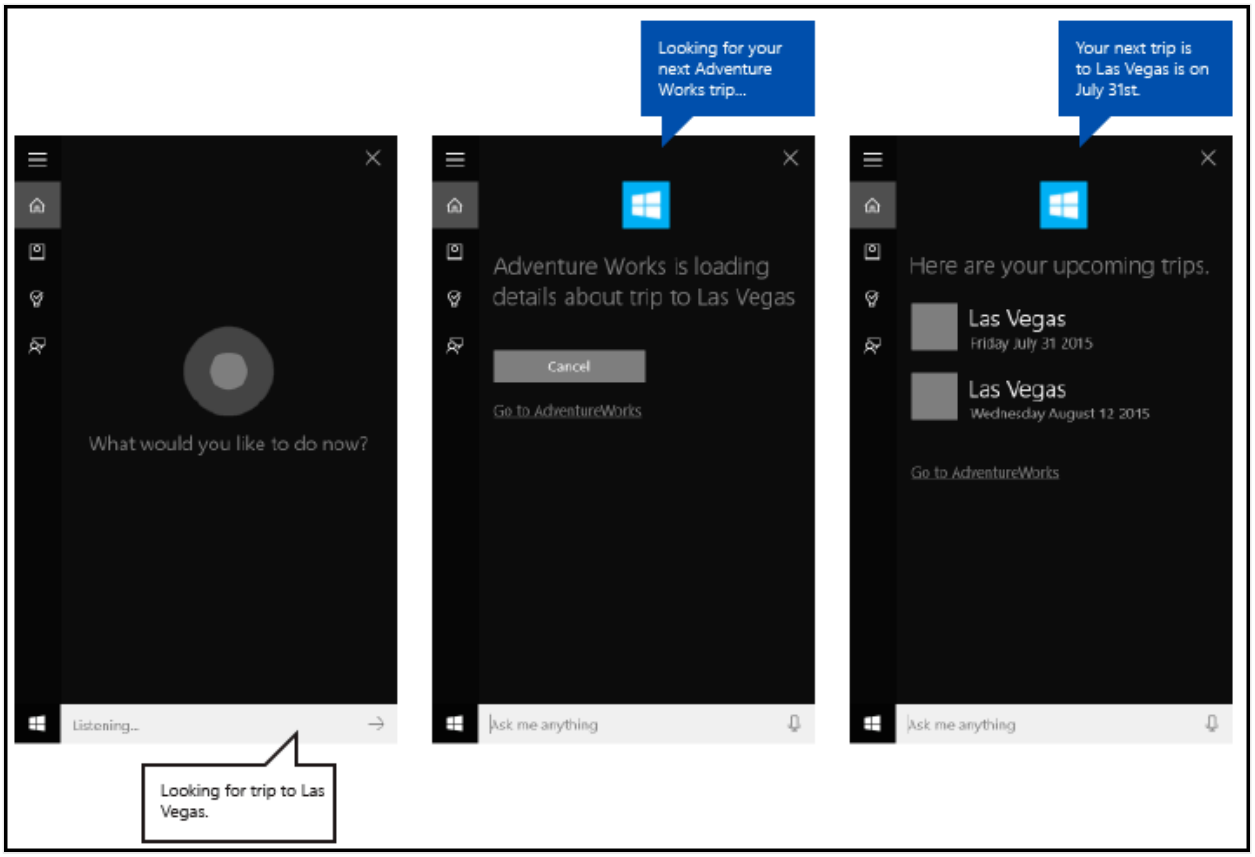
Hand-off screen

Once a voice command is recognized, **Cortana** must call ReportSuccessAsync and present feedback within approximately 500If the app service cannot complete the action specified by the voice command within 500ms, **Cortana** presents a hand-off screen that is shown until your app calls ReportSuccessAsync, or for up to 5 seconds.

If the app service doesn't call ReportSuccessAsync, or any other VoiceCommandServiceConnection method, the user receives an error message and the app service call is cancelled.

Here's an example of a hand-off screen for the **Adventure Works** app. In this example, a user has queried **Cortana** for upcoming trips. The hand-off screen includes a message customized with the app service name, an icon, and a **Feedback** string.

[!NOTE] You can declare a **Feedback** string in the VCD file. This string does not affect the UI text displayed on the Cortana canvas, it only affects the text spoken by **Cortana**.

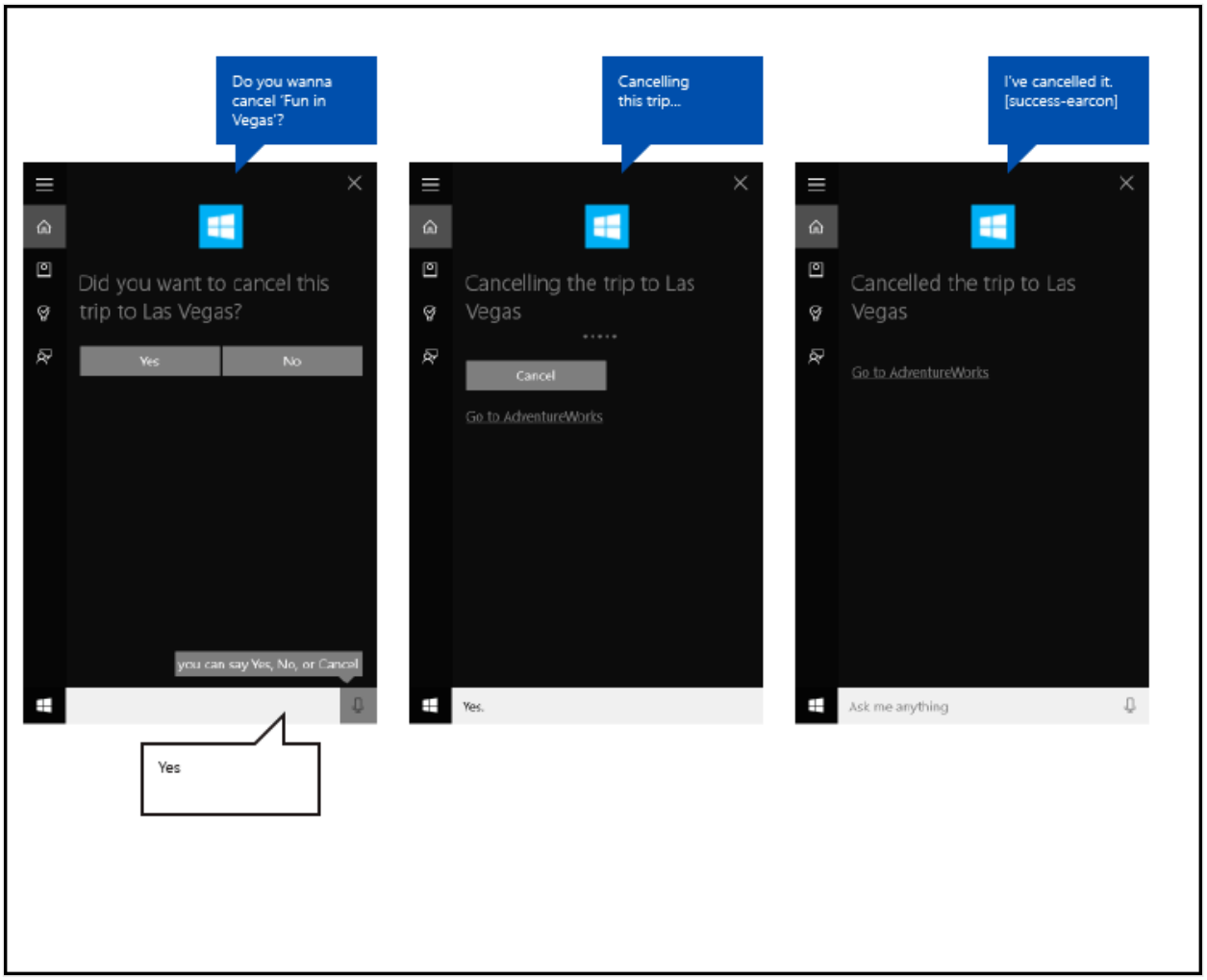


Progress screen

If the app service takes more than 500ms to call ReportSuccessAsync, **Cortana** provides the user with a progress screen. The app icon is displayed, and you must provide both GUI and TTS progress strings to indicate that the task is being actively handled.

Cortana shows a progress screen for a maximum of 5 seconds. After 5 seconds, **Cortana** presents the user with an error message and the app service is closed. If the app service needs more than 5 seconds to complete the action, it can continue to update **Cortana** with progress screens.

Here's an example of a progress screen for the **Adventure Works** app. In this example, a user has canceled a trip to Las Vegas. The progress screen includes a message customized for the action, an icon, and a content tile with information about the trip being canceled.



AdventureWorksVoiceCommandService.cs contains the following progress message method, which calls **ReportProgressAsync** to show the progress screen in **Cortana**.

CSharp

```
/// <summary>
/// Show a progress screen. These should be posted at least every 5 seconds
for a
/// long-running operation.
/// </summary>
/// <param name="message">The message to display, relating to the task being
performed.</param>
/// <returns></returns>
private async Task ShowProgressScreen(string message)
{
    var userProgressMessage = new VoiceCommandUserMessage();
    userProgressMessage.DisplayMessage = userProgressMessage.SpokenMessage =
message;

    VoiceCommandResponse response =
VoiceCommandResponse.CreateResponse(userProgressMessage);
    await voiceServiceConnection.ReportProgressAsync(response);
}
```

Confirmation screen

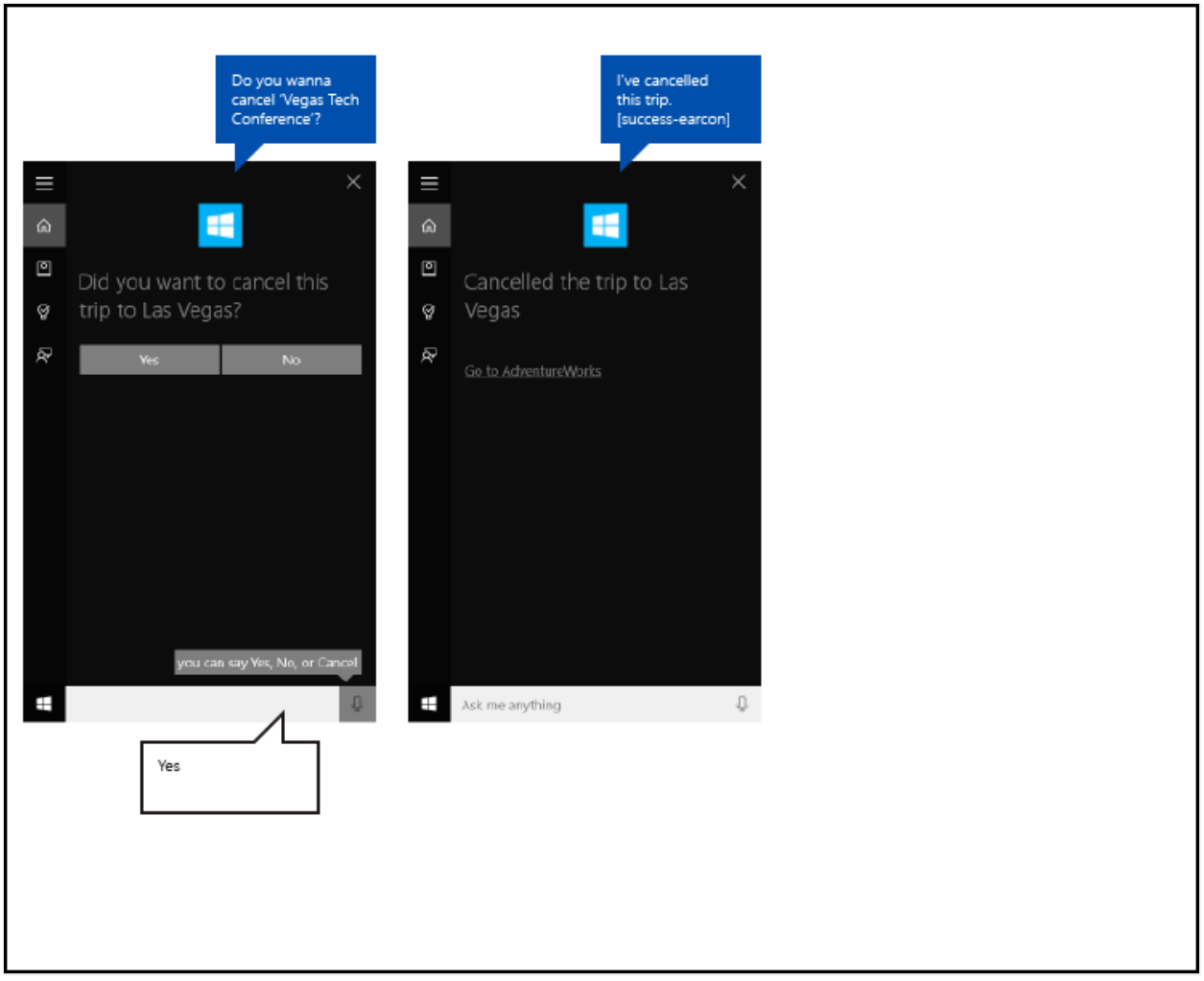
When an action specified by a voice command is irreversible, has a significant impact, or the recognition confidence is not high, an app service can request confirmation.

Here's an example of a confirmation screen for the **Adventure Works** app. In this example, a user has instructed the app service to cancel a trip to Las Vegas through **Cortana**. The app service has provided **Cortana** with a confirmation screen that prompts the user for a yes or no answer before canceling the trip.

If the user says something other than "Yes" or "No", **Cortana** cannot determine the answer to the question. In this case, **Cortana** prompts the user with a similar question provided by the app service.

On the second prompt, if the user still doesn't say "Yes" or "No", **Cortana** prompts the user a third time with the same question prefixed with an apology. If the user still doesn't say "Yes" or "No", **Cortana** stops listening for voice input and asks the user to tap one of the buttons instead.

The confirmation screen includes a message customized for the action, an icon, and a content tile with information about the trip being canceled.



AdventureWorksVoiceCommandService.cs contains the following trip cancellation method, which calls **RequestConfirmationAsync** to show a confirmation screen in **Cortana**.

CSharp

```
/// <summary>
/// Handle the Trip Cancellation task. This task demonstrates how to prompt
a user
/// for confirmation of an operation, show users a progress screen while
performing
/// a long-running task, and show a completion screen.
/// </summary>
/// <param name="destination">The name of a destination.</param>
/// <returns></returns>
private async Task SendCompletionMessageForCancellation(string destination)
{
    // Begin loading data to search for the target store.
    // Consider inserting a progress screen here, in order to prevent
Cortana from timing out.
    string progressScreenString = string.Format(
        cortanaResourceMap.GetValue("ProgressLookingForTripToDest",
cortanaContext).ValueAsString,
        destination);
    await ShowProgressScreen(progressScreenString);

    Model.TripStore store = new Model.TripStore();
    await store.LoadTrips();

    IEnumerable<Model.Trip> trips = store.Trips.Where(p => p.Destination ==
destination);
    Model.Trip trip = null;
    if (trips.Count() > 1)
    {
        // If there is more than one trip, provide a disambiguation screen.
        // However, if a significant number of items are returned, you might
want to
        // just display a link to your app and provide a deeper search
experience.
        string disambiguationDestinationString = string.Format(
            cortanaResourceMap.GetValue("DisambiguationWhichTripToDest",
cortanaContext).ValueAsString,
            destination);
        string disambiguationRepeatString =
cortanaResourceMap.GetValue("DisambiguationRepeat",
cortanaContext).ValueAsString;
        trip = await DisambiguateTrips(trips,
disambiguationDestinationString, disambiguationRepeatString);
    }
    else
    {
        trip = trips.FirstOrDefault();
    }
```

```
    var userPrompt = new VoiceCommandUserMessage();

    VoiceCommandResponse response;
    if (trip == null)
    {
        var userMessage = new VoiceCommandUserMessage();
        string noSuchTripToDestination = string.Format(
            cortanaResourceMap.GetValue("NoSuchTripToDestination",
cortanaContext).ValueAsString,
            destination);
        userMessage.DisplayMessage = userMessage.SpokenMessage =
noSuchTripToDestination;

        response = VoiceCommandResponse.CreateResponse(userMessage);
        await voiceServiceConnection.ReportSuccessAsync(response);
    }
    else
    {
        // Prompt the user for confirmation that this is the correct trip to
cancel.
        string cancelTripToDestination = string.Format(
            cortanaResourceMap.GetValue("CancelTripToDestination",
cortanaContext).ValueAsString,
            destination);
        userPrompt.DisplayMessage = userPrompt.SpokenMessage =
cancelTripToDestination;
        var userReprompt = new VoiceCommandUserMessage();
        string confirmCancelTripToDestination = string.Format(
            cortanaResourceMap.GetValue("ConfirmCancelTripToDestination",
cortanaContext).ValueAsString,
            destination);
        userReprompt.DisplayMessage = userReprompt.SpokenMessage =
confirmCancelTripToDestination;

        response = VoiceCommandResponse.CreateResponseForPrompt(userPrompt,
userReprompt);

        var voiceCommandConfirmation = await
voiceServiceConnection.RequestConfirmationAsync(response);

        // If RequestConfirmationAsync returns null, Cortana has likely been
dismissed.
        if (voiceCommandConfirmation != null)
        {
            if (voiceCommandConfirmation.Confirmed == true)
            {
                string cancellingTripToDestination = string.Format(
               cortanaResourceMap.GetValue("CancellingTripToDestination",
cortanaContext).ValueAsString,
               destination);
                await ShowProgressScreen(cancellingTripToDestination);

                // Perform the operation to remove the trip from app data.
                // As the background task runs within the app package of the
```

```
installed app,
                // we can access local files belonging to the app without
issue.
                await store.DeleteTrip(trip);

                // Provide a completion message to the user.
                var userMessage = new VoiceCommandUserMessage();
                string cancelledTripToDestination = string.Format(

cortanaResourceMap.GetValue("CancelledTripToDestination",
cortanaContext).ValueAsString,
                    destination);
                userMessage.DisplayMessage = userMessage.SpokenMessage =
cancelledTripToDestination;
                response = VoiceCommandResponse.CreateResponse(userMessage);
                await voiceServiceConnection.ReportSuccessAsync(response);
            }
            else
            {
                // Confirm no action for the user.
                var userMessage = new VoiceCommandUserMessage();
                string keepingTripToDestination = string.Format(
                    cortanaResourceMap.GetValue("KeepingTripToDestination",
cortanaContext).ValueAsString,
                    destination);
                userMessage.DisplayMessage = userMessage.SpokenMessage =
keepingTripToDestination;

                response = VoiceCommandResponse.CreateResponse(userMessage);
                await voiceServiceConnection.ReportSuccessAsync(response);
            }
        }
    }
}
```

Disambiguation screen

When an action specified by a voice command has more than one possible outcome, an app service can request more info from the user.

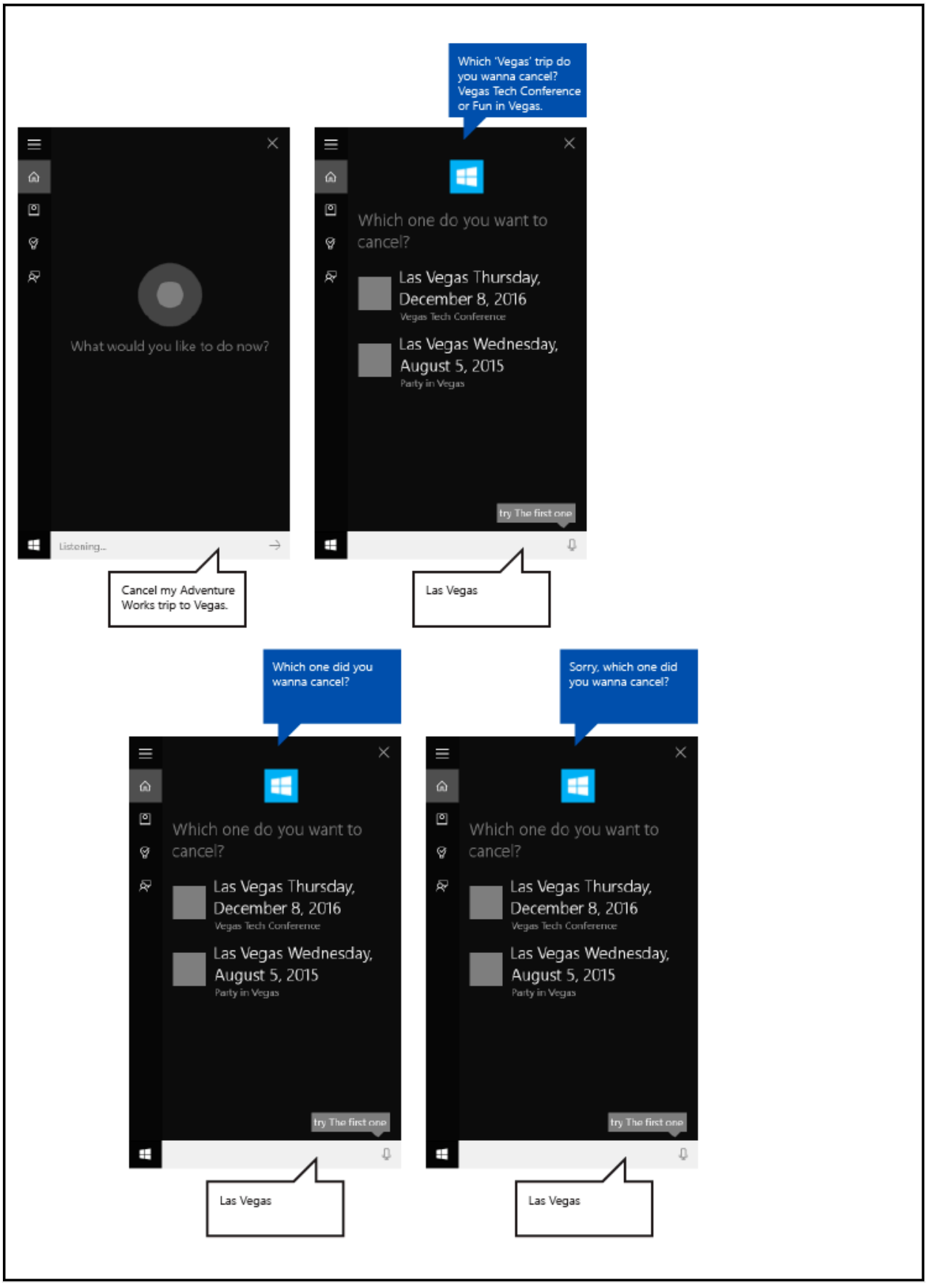
Here's an example of a disambiguation screen for the **Adventure Works** app. In this example, a user has instructed the app service to cancel a trip to Las Vegas through **Cortana**. However, the user has two trips to Las Vegas on different dates and the app service cannot complete the action without the user selecting the intended trip.

The app service provides **Cortana** with a disambiguation screen that prompts the user to make a selection from a list of matching trips, before it cancels any.

In this case, **Cortana** prompts the user with a similar question provided by the app service.

On the second prompt, if the user still doesn't say something that can be used to identify the selection, **Cortana** prompts the user a third time with the same question prefixed with an apology. If the user still doesn't say something that can be used to identify the selection, **Cortana** stops listening for voice input and asks the user to tap one of the buttons instead.

The disambiguation screen includes a message customized for the action, an icon, and a content tile with information about the trip being canceled.



AdventureWorksVoiceCommandService.cs contains the following trip cancellation method, which calls **RequestDisambiguationAsync** to show the disambiguation screen in **Cortana**.

```
C#
```

```
/// <summary>
/// Provide the user with a way to identify which trip to cancel.
/// </summary>
/// <param name="trips">The set of trips</param>
/// <param name="disambiguationMessage">The initial disambiguation
message</param>
/// <param name="secondDisambiguationMessage">Repeat prompt retry
message</param>
private async Task<Model.Trip> DisambiguateTrips(IEnumerable<Model.Trip>
trips, string disambiguationMessage, string secondDisambiguationMessage)
{
    // Create the first prompt message.
    var userPrompt = new VoiceCommandUserMessage();
    userPrompt.DisplayMessage =
        userPrompt.SpokenMessage = disambiguationMessage;

    // Create a re-prompt message if the user responds with an out-of-
grammar response.
    var userReprompt = new VoiceCommandUserMessage();
    userReprompt.DisplayMessage =
        userReprompt.SpokenMessage = secondDisambiguationMessage;

    // Create card for each item.
    var destinationContentTiles = new List<VoiceCommandContentTile>();
    int i = 1;
    foreach (Model.Trip trip in trips)
    {
        var destinationTile = new VoiceCommandContentTile();

        destinationTile.ContentTileType =
VoiceCommandContentTileType.TitleWith68x68IconAndText;
        destinationTile.Image = await
StorageFile.GetFileFromApplicationUriAsync(new Uri("ms-
appx:///AdventureWorks.VoiceCommands/Images/GreyTile.png"));

        // The AppContext can be any arbitrary object.
        destinationTile.AppContext = trip;
        string dateFormat = "";
        if (trip.StartDate != null)
        {
            dateFormat =
trip.StartDate.Value.ToString(dateFormatInfo.LongDatePattern);
        }
        else
        {
            // The app allows a trip to have no date.
            // However, the choices must be unique so they can be
distinguished.
            // Here, we add a number to identify them.
            dateFormat = string.Format("{0}", i);
        }

        destinationTile.Title = trip.Destination + " " + dateFormat;
        destinationTile.TextLine1 = trip.Description;
```

```
        destinationContentTiles.Add(destinationTile);
        i++;
    }

    // Cortana handles re-prompting if no valid response.
    var response = VoiceCommandResponse.CreateResponseForPrompt(userPrompt,
userReprompt, destinationContentTiles);

    // If cortana is dismissed in this operation, null is returned.
    var voiceCommandDisambiguationResult = await
        voiceServiceConnection.RequestDisambiguationAsync(response);
    if (voiceCommandDisambiguationResult != null)
    {
        return
(Model.Trip)voiceCommandDisambiguationResult.SelectedItem.AppContext;
    }

    return null;
}
```

Error screen

When an action specified by a voice command cannot be completed, an app service can provide an error screen.

Here's an example of an error screen for the **Adventure Works** app. In this example, a user has instructed the app service to cancel a trip to Las Vegas through **Cortana**. However, the user does not have any trips scheduled to Las Vegas.

The app service provides **Cortana** with an error screen that includes a message customized for the action, an icon, and the specific error message.

Call ReportFailureAsync to show the error screen in **Cortana**.

C#

```
var userMessage = new VoiceCommandUserMessage();
    userMessage.DisplayMessage = userMessage.SpokenMessage =
      "Sorry, you don't have any trips to Las Vegas";

    var response = VoiceCommandResponse.CreateResponse(userMessage);

    response.AppLaunchArgument = "showUpcomingTrips";
    await voiceServiceConnection.ReportFailureAsync(response);
```

Related articles

- Cortana interactions in Windows apps
- VCD elements and attributes v1.2
- Cortana design guidelines
- Cortana voice command sample

Deep link from a background app in Cortana to a foreground app

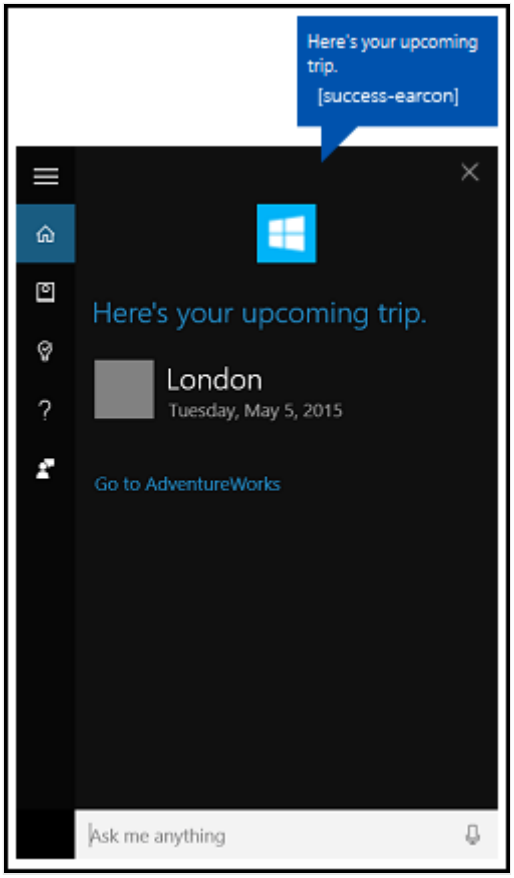
Article • 06/24/2021

> ⚠ **Warning**
>
> This feature is no longer supported as of the Windows 10 May 2020 Update (version 2004, codename "20H1").

Provide deep links from a background app in **Cortana** that launch the app to the foreground in a specific state or context.

> ⓘ **Note**
>
> Both **Cortana** and the background app service are terminated when the foreground app is launched.

A deep link is displayed by default on the **Cortana** completion screen as shown here ("Go to AdventureWorks"), but you can display deep links on various other screens.



> **Note**
>
> **Important APIs**
>
> - Windows.ApplicationModel.VoiceCommands
> - VCD elements and attributes v1.2

Overview

Users can access your app through **Cortana** by:

- Activating it as a foreground app (see Activate a foreground app with voice commands through Cortana).
- Exposing specific functionality as a background app service (see Activate a background app in Cortana using voice commands).
- Deep linking to specific pages, content, and state or context.

We discuss deep linking here.

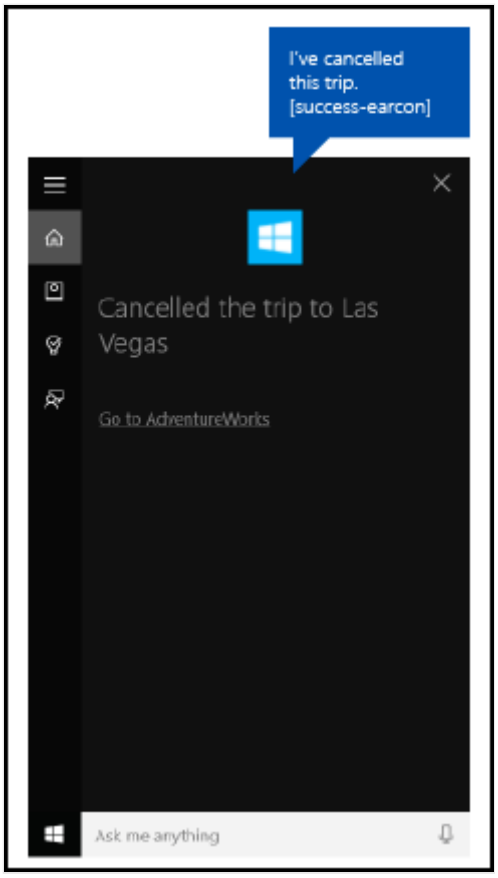
Deep linking is useful when Cortana and your app service act as a gateway to your full-featured app (instead of requiring the user to launch the app through the Start menu), or for providing access to richer detail and functionality within your app that is not possible through Cortana. Deep linking is another way to increase usability and promote your app.

There are three ways to provide deep links:

- A "Go to <app>" link on various **Cortana** screens.
- A link embedded in a content tile on various **Cortana** screens.
- Programmatically launching the foreground app from the background app service.

"Go to <app>" deep link

Cortana displays a "Go to <app>" deep link below the content card on most screens.



You can provide a launch argument for this link that opens your app in similar context as the app service. If you don't provide a launch argument, the app is launched to the main screen.

In this example from AdventureWorksVoiceCommandService.cs of the **AdventureWorks** sample, we pass the specified destination (`destination`) string to the SendCompletionMessageForDestination method, which retrieves all matching trips and provides a deep link to the app.

First, we create a **VoiceCommandUserMessage** (`userMessage`) that is spoken by **Cortana** and shown on the **Cortana** canvas. A **VoiceCommandContentTile** list object is then created for displaying the collection of result cards on the canvas.

These two objects are then passed to the CreateResponse method of the **VoiceCommandResponse** object (`response`). We then set the **AppLaunchArgument** property value of the response object to the value of `destination` passed to this function. When a user taps a content tile on the Cortana canvas, the parameter values are passed to the app through the response object.

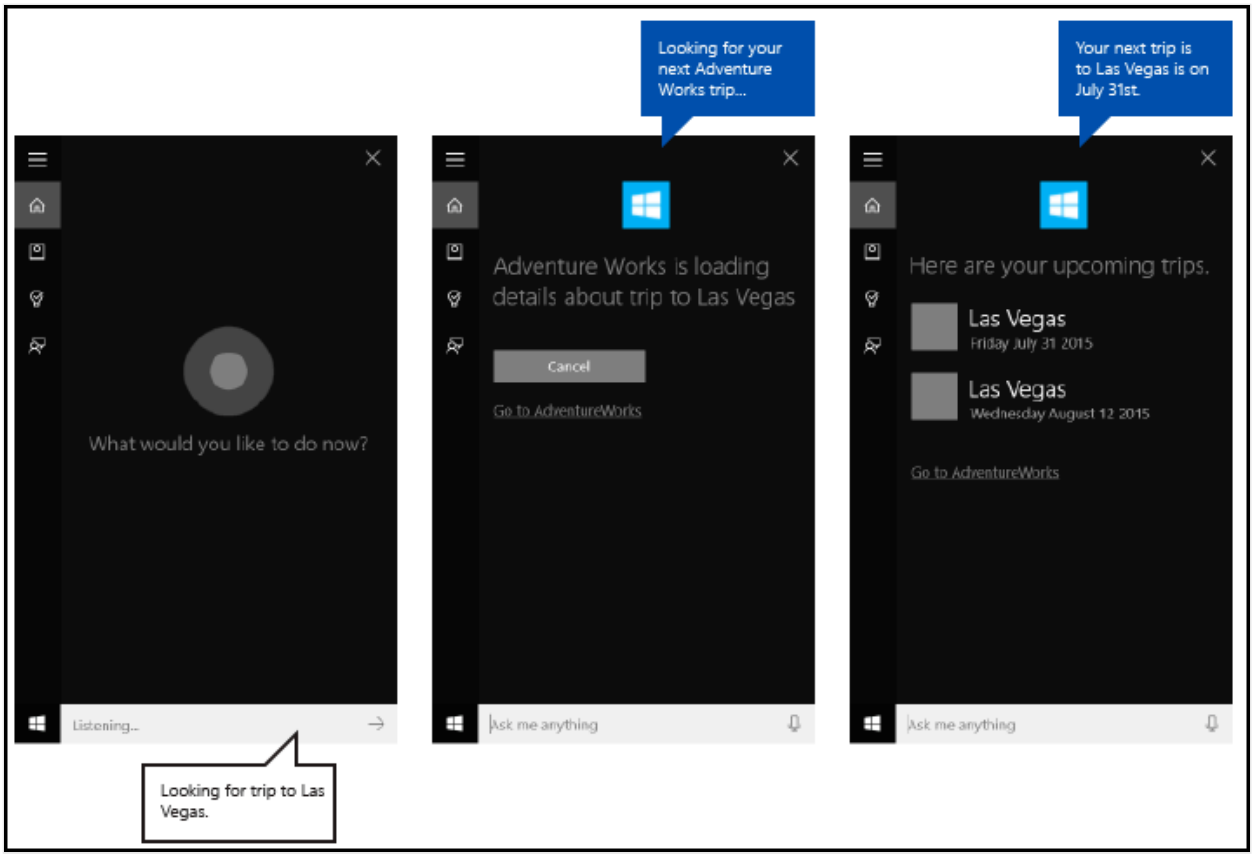
Finally, we call the **ReportSuccessAsync** method of the **VoiceCommandServiceConnection**.

```
C#
```

```
/// <summary>
/// Show details for a single trip, if the trip can be found.
/// This demonstrates a simple response flow in Cortana.
/// </summary>
/// <param name="destination">The destination specified in the voice
command.</param>
private async Task SendCompletionMessageForDestination(string destination)
{
...
    IEnumerable<Model.Trip> trips = store.Trips.Where(p => p.Destination ==
destination);

    var userMessage = new VoiceCommandUserMessage();
    var destinationsContentTiles = new List<VoiceCommandContentTile>();
...
    var response = VoiceCommandResponse.CreateResponse(userMessage,
destinationsContentTiles);

    if (trips.Count() > 0)
    {
        response.AppLaunchArgument = destination;
    }

    await voiceServiceConnection.ReportSuccessAsync(response);
}
```

Content tile deep link

You can add deep links to content cards on various **Cortana** screens.



AdventureWorks "Upcoming trip" with handoff screen

Like the "Go to <app>" links, you can provide a launch argument to open your app with similar context as the app service. If you don't provide a launch argument, the content tile does not link to your app.

In this example from AdventureWorksVoiceCommandService.cs of the **AdventureWorks** sample, we pass the specified destination to the SendCompletionMessageForDestination method, which retrieves all matching trips and provides content cards with deep links to the app.

First, we create a **VoiceCommandUserMessage** (`userMessage`) that is spoken by **Cortana** and shown on the **Cortana** canvas. A **VoiceCommandContentTile** list object is then created for displaying the collection of result cards on the canvas.

These two objects are then passed to the CreateResponse method of the **VoiceCommandResponse** object (`response`). We then set the **AppLaunchArgument** property value to the value of the destination in the voice command.

Finally, we call the **ReportSuccessAsync** method of the **VoiceCommandServiceConnection**. Here, we add two content tiles with different **AppLaunchArgument** parameter values to a **VoiceCommandContentTile** list used in the **ReportSuccessAsync** call of the **VoiceCommandServiceConnection** object.

```
C#
```

```
/// <summary>
/// Show details for a single trip, if the trip can be found.
/// This demonstrates a simple response flow in Cortana.
/// </summary>
/// <param name="destination">The destination specified in the voice
command.</param>
private async Task SendCompletionMessageForDestination(string destination)
{
    // If this operation is expected to take longer than 0.5 seconds, the
task must
    // supply a progress response to Cortana before starting the operation,
and
    // updates must be provided at least every 5 seconds.
    string loadingTripToDestination = string.Format(
                cortanaResourceMap.GetValue("LoadingTripToDestination",
cortanaContext).ValueAsString,
                destination);
    await ShowProgressScreen(loadingTripToDestination);
    Model.TripStore store = new Model.TripStore();
    await store.LoadTrips();

    // Query for the specified trip.
    // The destination should be in the phrase list. However, there might be
    // multiple trips to the destination. We pick the first.
    IEnumerable<Model.Trip> trips = store.Trips.Where(p => p.Destination ==
destination);

    var userMessage = new VoiceCommandUserMessage();
    var destinationsContentTiles = new List<VoiceCommandContentTile>();
    if (trips.Count() == 0)
    {
        string foundNoTripToDestination = string.Format(
                cortanaResourceMap.GetValue("FoundNoTripToDestination",
cortanaContext).ValueAsString,
                destination);
        userMessage.DisplayMessage = foundNoTripToDestination;
        userMessage.SpokenMessage = foundNoTripToDestination;
    }
    else
    {
        // Set plural or singular title.
        string message = "";
        if (trips.Count() > 1)
        {
            message = cortanaResourceMap.GetValue("PluralUpcomingTrips",
cortanaContext).ValueAsString;
        }
        else
        {
            message = cortanaResourceMap.GetValue("SingularUpcomingTrip",
cortanaContext).ValueAsString;
        }
        userMessage.DisplayMessage = message;
        userMessage.SpokenMessage = message;
```

```
        // Define a tile for each destination.
        foreach (Model.Trip trip in trips)
        {
            int i = 1;

            var destinationTile = new VoiceCommandContentTile();

            destinationTile.ContentTileType =
VoiceCommandContentTileType.TitleWith68x68IconAndText;
            destinationTile.Image = await
StorageFile.GetFileFromApplicationUriAsync(new Uri("ms-
appx:///AdventureWorks.VoiceCommands/Images/GreyTile.png"));

            destinationTile.AppLaunchArgument = trip.Destination;
            destinationTile.Title = trip.Destination;
            if (trip.StartDate != null)
            {
                destinationTile.TextLine1 =
trip.StartDate.Value.ToString(dateFormatInfo.LongDatePattern);
            }
            else
            {
                destinationTile.TextLine1 = trip.Destination + " " + i;
            }

            destinationsContentTiles.Add(destinationTile);
            i++;
        }
    }

    var response = VoiceCommandResponse.CreateResponse(userMessage,
destinationsContentTiles);

    if (trips.Count() > 0)
    {
        response.AppLaunchArgument = destination;
    }

    await voiceServiceConnection.ReportSuccessAsync(response);
}
```

Programmatic deep link

You can also programmatically launch your app with a launch argument to open your app with similar context as the app service. If you don't provide a launch argument, the app is launched to the main screen.

Here, we add an **AppLaunchArgument** parameter with a value of "Las Vegas" to a **VoiceCommandResponse** object used in the **RequestAppLaunchAsync** call of the

VoiceCommandServiceConnection object.

```CSharp
var userMessage = new VoiceCommandUserMessage();
userMessage.DisplayMessage = "Here are your trips.";
userMessage.SpokenMessage =
  "You have one trip to Vegas coming up.";

response = VoiceCommandResponse.CreateResponse(userMessage);
response.AppLaunchArgument = "Las Vegas";
await  VoiceCommandServiceConnection.RequestAppLaunchAsync(response);
```

App manifest

To enable deep linking to your app, you must declare the `windows.personalAssistantLaunch` extension in the Package.appxmanifest file of your app project.

Here, we declare the `windows.personalAssistantLaunch` extension for the **Adventure Works** app.

```XML
<Extensions>
  <uap:Extension Category="windows.appService"

EntryPoint="AdventureWorks.VoiceCommands.AdventureWorksVoiceCommandService">
    <uap:AppService Name="AdventureWorksVoiceCommandService"/>
  </uap:Extension>
  <uap:Extension Category="windows.personalAssistantLaunch"/>
</Extensions>
```

Protocol contract

Your app is launched to the foreground through Uniform Resource Identifier (URI) activation using a **Protocol** contract. Your app must override your app's **OnActivated** event and check for an **ActivationKind** of **Protocol**. For more info, see Handle URI activation.

Here, we decode the URI provided by the **ProtocolActivatedEventArgs** to access the launch argument. For this example, the **Uri** is set to "windows.personalassistantlaunch:?LaunchContext=Las Vegas".

```
```

CSharp

```
if (args.Kind == ActivationKind.Protocol)
  {
    var commandArgs = args as ProtocolActivatedEventArgs;
    Windows.Foundation.WwwFormUrlDecoder decoder =
      new Windows.Foundation.WwwFormUrlDecoder(commandArgs.Uri.Query);
    var destination = decoder.GetFirstValueByName("LaunchContext");

    navigationCommand = new ViewModel.TripVoiceCommand(
      "protocolLaunch",
      "text",
      "destination",
      destination);

    navigationToPageType = typeof(View.TripDetails);

    rootFrame.Navigate(navigationToPageType, navigationCommand);

    // Ensure the current window is active.
    Window.Current.Activate();
  }
```

Related articles

- Cortana interactions in Windows apps
- Cortana design guidelines
- VCD elements and attributes v1.2
- Cortana voice command sample

Support natural language voice commands in Cortana

Article • 06/24/2021

> ⚠ **Warning**
>
> This feature is no longer supported as of the Windows 10 May 2020 Update (version 2004, codename "20H1").

Extend **Cortana** with more flexible and natural voice commands that let a user say your app's name anywhere in the command.

> ⓘ **Note**
>
> **Important APIs**
>
> - Windows.ApplicationModel.VoiceCommands
> - VCD elements and attributes v1.2

Using voice commands to extend Cortana with functionality from your app requires the user to specify both the app and the command or function to execute. This is typically accomplished by announcing the app name at the beginning or the end of the voice command. For example, "Adventure Works, add a new trip to Las Vegas."

However, specifying the application name in this way can sound awkward, stilted, or not even make sense. In many cases, being able to say the app name elsewhere in the command is more comfortable and natural, and helps make the interaction much more intuitive and engaging for the user. Our previous example, "Adventure Works, add a new trip to Las Vegas." could be rephrased as "Add a new Adventure Works trip to Las Vegas." or "Using Adventure Works, add a new trip to Las Vegas."

You can set up your voice commands to support the app name as a:

- Prefix - before the command phrase
- Infix - within the command phrase
- Suffix - after the command phrase

> 💡 **Tip**

Prerequisites

This topic builds on Activate a background app in Cortana using voice commands. We continue here to demonstrate features with a trip planning and management app named **Adventure Works**.

If you're new to developing Universal Windows Platform (UWP) apps, have a look through these topics to get familiar with the technologies discussed here.

- Create your first app
- Learn about events with Events and routed events overview

User experience guidelines

See Cortana design guidelines for info about how to integrate your app with **Cortana** and Speech interactions for helpful tips on designing a useful and engaging speech-enabled app.

Specify an AppName element in the VCD

The **AppName** element is used to specify a user-friendly name for an app in a voice command.

```
XML
<AppName>Adventure Works</AppName>
```

Specify where the app name can be spoken in the voice command

The **ListenFor** element has a **RequireAppName** attribute that specifies where the app name can appear in the voice command. This attribute supports four values.

1. **BeforePhrase**

   Default.

   Indicates that users must say your app name before the command phrase.

   Here, Cortana listens for "Adventure Works when is my trip to Las Vegas".

   ```
   XML
   ```

```
<ListenFor RequireAppName="BeforePhrase"> show [my] trip to
{destination} </ListenFor>
```

2. **AfterPhrase**

   Indicates that users must say your app name after the command phrase.

   A localized phrase list of prepositional conjunctions is provided by the system. This includes phrases such as, "using", "with "and "on".

   Here, Cortana listens for commands like "Show my next trip to Las Vegas on Adventure Works" and "Show my next trip to Las Vegas using Adventure Works".

   XML

   ```
   <ListenFor RequireAppName="AfterPhrase">show [my] next trip to
   {destination} </ListenFor>
   ```

3. **BeforeOrAfterPhrase**

   Indicates that users must say your app name either before or after the command phrase.

   For the suffix version, a localized phrase list of prepositional conjunctions is provided by the system. This includes phrases such as, "using", "with "and "on".

   Here, Cortana listens for commands like "Adventure Works, show my next trip to Las Vegas" or "Show my next trip to Last Vegas on Adventure works".

   XML

   ```
   <ListenFor RequireAppName="BeforeOrAfterPhrase">show [my] next trip to
   {destination}</ListenFor>
   ```

4. **ExplicitlySpecified**

   Indicates that users must say your app name exactly where you specify in the command phrase. The user is not required to say the app name either before or after the phrase.

   You must explicitly reference your app name using the **{builtin:AppName}** tag.

   Here, Cortana listens for commands like "Adventure Works, show my next trip to Las Vegas" or "Show my next Adventure Works trip to Las Vegas".

XML

```
<ListenFor RequireAppName="ExplicitlySpecified">show [my] next
{builtin:AppName} trip to {destination} </ListenFor>
```

Special cases

When you declare a **ListenFor** element where **RequireAppName** is either "AfterPhrase" or "ExplicitlySpecified", you must ensure certain requirements are met:

1. **{builtin:AppName}** must appear once and only once when **RequireAppName** is "ExplicitlySpecified".

   With this value, the system cannot infer where the app name can appear in the voice command. You must explicitly specify the location.

2. You cannot have a voice command begin with a **PhraseTopic** element, which is typically used for large vocabulary speech recognition. At least one word must precede it.

   This helps to minimize the chance that **Cortana** launches your app if a command contains your app name, or part of it, anywhere in the utterance.

   Here is an invalid declaration that could lead to **Cortana** launching the **Adventure Works** app if the user says something like "Show me reviews for Kinect Adventure works".

   XML

   ```
   <ListenFor RequireAppName="ExplicitlySpecified">{searchPhrase}
   {builtin:AppName}</ListenFor>
   ```

3. There must be at least two words in the **ListenFor** string besides your app name and references to **PhraseTopic** elements.

   Similar to case 2, you need to ensure your commands contain sufficient phonetic content to minimize the chances your app is launched unintentionally.

   This helps you set up your application for best possible success so your application does not get incorrectly launched when user says for example "Find Kinect Adventure works".

   Here are invalid declarations that could lead to **Cortana** launching the **Adventure Works** app if the user says something like "Hey adventure works" or "Find Kinect

adventure works".

XML

```
<ListenFor RequireAppName="ExplicitlySpecified">Hey {builtin:AppName}
</ListenFor>
<ListenFor RequireAppName="ExplicitlySpecified">Find {searchPhrase}
{builtin:AppName}</ListenFor>
```

Remarks

Supporting more variation in how a voice command can be uttered by a user in **Cortana** also increases the general usability of your app.

Avoid having "Hey [app name]" as your **AppName**. Users are much more likely to say "Hey Cortana" to invoke Cortana through voice activation, and having "Hey [app name]" in the utterance does not sound natural. For example, "Hey Cortana, show my next trip to Las Vegas on Hey Adventure Works".

Consider adding infix/suffix variations to your existing voice commands. As we've shown here, it doesn't require a lot of effort to add an additional attribute to your existing **ListenFor** elements and support suffix variants. It feels a lot more natural to say "Hey Cortana, show my next trip to Las Vegas on Adventure works" than "Hey Cortana, Adventure Works, show my next trip to Las Vegas".

Consider using your app name as a prefix in cases where the voice command conflicts with existing **Cortana** functionality (calling, messaging, and so on). For example, "Adventure Works, message [travel agent] about Las Vegas trip".

Complete example

Here is a VCD file that demonstrates various ways to provide more natural language voice commands.

> **Note**
>
> It is valid to have multiple **ListenFor** elements, each with a different **RequireAppName** attribute value.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<VoiceCommands xmlns="https://schemas.microsoft.com/voicecommands/1.1">
  <CommandSet xml:lang="en-us" Name="commandSet_en-us">
    <AppName>Adventure Works</AppName>
    <Example> When is my trip to Las Vegas? </Example>
    <Command Name="whenIsTripToDestination">
      <Example> When is my trip to Las Vegas?</Example>
      <ListenFor RequireAppName="BeforePhrase">
        when is my] trip to {destination} </ListenFor>

      <!-- This ListenFor command will set up Cortana to accept commands
like
           "Show my next trip to Las Vegas on Adventure Works"; "Show my
next
           trip to Las Vegas using Adventure Works" -->
      <ListenFor RequireAppName="AfterPhrase">
        show [my] next trip to {destination} </ListenFor>

      <!-- This ListenFor command will set up Cortana to accept commands
when
           the user specifies your app name either before or after the
command.
           "Adventure Works, show my next trip to Las Vegas";
           "Show my next trip to Last Vegas on Adventure works" -->
      <ListenFor RequireAppName="BeforeOrAfterPhrase">
        show [my] next trip to {destination} </ListenFor>

      <!-- This ListenFor command will set up Cortana to accept commands
           when the user specifies your app name inline.
           "Show my next Adventure Works trip to Las Vegas" -->
      <ListenFor RequireAppName="ExplicitlySpecified">
        show [my] next {builtin:AppName} trip to {destination} </ListenFor>

      <Feedback> Looking for trip to {destination} </Feedback>
      <Navigate />
    </Command>
    <PhraseList Label="destination">
      <Item> Las Vegas </Item>
      <Item> Dallas </Item>
      <Item> New York </Item>
    </PhraseList>
  </CommandSet>
  <!-- Other CommandSets for other languages -->
</VoiceCommands>
```

Related articles

- Cortana interactions in Windows apps
- Cortana design guidelines
- VCD elements and attributes v1.2

- Cortana voice command sample

Multiple inputs

Article • 06/24/2021

Just as people use a combination of voice and gesture when communicating with each other, multiple types and modes of input can also be useful when interacting with an app.

To accommodate as many users and devices as possible, we recommend that you design your apps to work with as many input types as possible (gesture, speech, touch, touchpad, mouse, and keyboard). Doing so will maximize flexibility, usability, and accessibility.

To begin, consider the various scenarios in which your app handles input. Try to be consistent throughout your app, and remember that the platform controls provide built-in support for multiple input types.

- Can users interact with the application through multiple input devices?
- Are all input methods supported at all times? With certain controls? At specific times or circumstances?
- Does one input method take priority?

Single (or exclusive)-mode interactions

With single-mode interactions, multiple input types are supported, but only one can be used per action. For example, speech recognition for commands, and gestures for navigation; or, text entry using touch or gestures, depending on proximity.

Multimodal interactions

With multimodal interactions, multiple input methods in sequence are used to complete a single action.

Speech + gesture
The user points to a product, and then says “Add to cart.”

Speech + touch
The user selects a photo using press and hold, and then says “Send photo.”

Simulate user input through input injection

Article • 06/24/2021

Simulate and automate user input from devices such as keyboard, mouse, touch, pen, and gamepad in your Windows applications.

> **Important APIs**: Windows.UI.Input.Preview.Injection

Overview

Input injection enables your Windows application to simulate input from a variety of input devices and direct that input anywhere, including outside your app's client area (even to apps running with Administrator privileges, such as the Registry Editor).

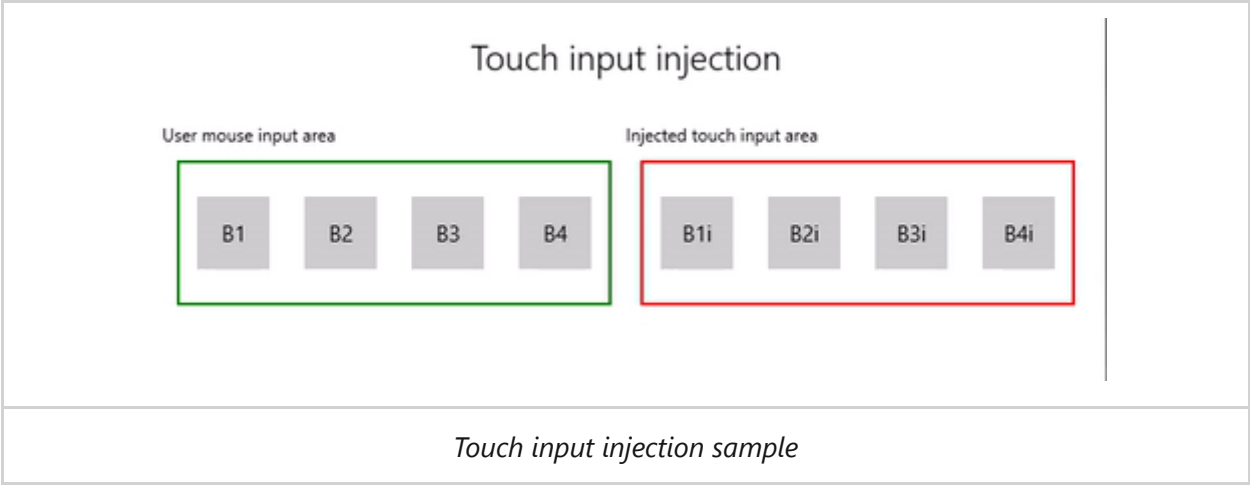
Input injection is useful for Windows apps and tools that need to provide functionality that includes accessibility, testing (ad-hoc, automated), and remote access and support features.

Setup

To use the input injection APIs in your Windows app you'll need to add the following to the app manifest:

1. Right click the **Package.appxmanifest** file and select **View code**.
2. Insert the following into the `Package` node:
    - `xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities"`
    - `IgnorableNamespaces="rescap"`
3. Insert the following into the `Capabilities` node:
    - `<rescap:Capability Name="inputInjectionBrokered" />`

Duplicate user input



Touch input injection sample

In this example, we demonstrate how to use the input injection APIs (Windows.UI.Input.Preview.Injection) to listen for mouse input events in one region of an app, and simulate corresponding touch input events in another region.

Download this sample from Input injection sample (mouse to touch)

1. First, we set up the UI (MainPage.xaml).

   We have two Grid areas (one for mouse input and one for injected touch input), each with four buttons.

   > **Note**
   >
   > The Grid background must be assigned a value (`Transparent`, in this case), otherwise pointer events are not detected.

   When any mouse clicks are detected in the input area, a corresponding touch event is injected into the input injection area. Button clicks from inject input are reported in the title area.

   XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <StackPanel Grid.Row="0"
                Margin="10">
        <TextBlock Style="{ThemeResource TitleTextBlockStyle}"
                   Name="titleText"
                   Text="Touch input injection"
                   HorizontalTextAlignment="Center" />
        <TextBlock Style="{ThemeResource BodyTextBlockStyle}"
```

```
                    Name="statusText"
                    HorizontalTextAlignment="Center" />
    </StackPanel>
    <Grid HorizontalAlignment="Center"
                     Grid.Row="1">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <TextBlock Grid.Column="0"
                   Grid.Row="0"
                   Style="{ThemeResource CaptionTextBlockStyle}"
                   Text="User mouse input area"/>
        <!-- Background must be set to something, otherwise pointer
events are not detected. -->
        <Grid Name="ContainerInput"
              Grid.Column="0"
              Grid.Row="1"
              HorizontalAlignment="Stretch"
              Background="Transparent"
              BorderBrush="Green"
              BorderThickness="2"
              MinHeight="100" MinWidth="300"
              Margin="10">
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Button Name="B1"
                    Grid.Column="0"
                    HorizontalAlignment="Center"
                    Width="50" Height="50"
                    Content="B1" />
            <Button Name="B2"
                    Grid.Column="1"
                    HorizontalAlignment="Center"
                    Width="50" Height="50"
                    Content="B2" />
            <Button Name="B3"
                    Grid.Column="2"
                    HorizontalAlignment="Center"
                    Width="50" Height="50"
                    Content="B3" />
            <Button Name="B4"
                    Grid.Column="3"
                    HorizontalAlignment="Center"
                    Width="50" Height="50"
                    Content="B4" />
        </Grid>
```

```
        <TextBlock Grid.Column="1"
                   Grid.Row="0"
                   Style="{ThemeResource CaptionTextBlockStyle}"
                   Text="Injected touch input area"/>
        <Grid Name="ContainerInject"
              Grid.Column="1"
              Grid.Row="1"
              HorizontalAlignment="Stretch"
              BorderBrush="Red"
              BorderThickness="2"
              MinHeight="100" MinWidth="300"
              Margin="10">
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Button Name="B1i" Click="Button_Click_Injected"
                    Content="B1i"
                    Grid.Column="0"
                    HorizontalAlignment="Center"
                    Width="50" Height="50" />
            <Button Name="B2i" Click="Button_Click_Injected"
                    Content="B2i"
                    Grid.Column="1"
                    HorizontalAlignment="Center"
                    Width="50" Height="50" />
            <Button Name="B3i" Click="Button_Click_Injected"
                    Content="B3i"
                    Grid.Column="2"
                    HorizontalAlignment="Center"
                    Width="50" Height="50" />
            <Button Name="B4i" Click="Button_Click_Injected"
                    Content="B4i"
                    Grid.Column="3"
                    HorizontalAlignment="Center"
                    Width="50" Height="50" />
        </Grid>
    </Grid>
</Grid>
```

2. Next, we initialize our app.

   In this snippet, we declare our global objects and declare listeners for pointer events (AddHandler) within the mouse input area that might be marked as handled in the button click events.

   The InputInjector object represents the virtual input device for sending the input data.

   In the `ContainerInput_PointerPressed` handler we call the touch injection function.

In the `ContainerInput_PointerReleased` handler, we call UninitializeTouchInjection to shut down the InputInjector object.

C#

```
public sealed partial class MainPage : Page
{
    /// <summary>
    /// The virtual input device.
    /// </summary>
    InputInjector _inputInjector;

    /// <summary>
    /// Initialize the app, set the window size,
    /// and add pointer input handlers for the container.
    /// </summary>
    public MainPage()
    {
        this.InitializeComponent();

        ApplicationView.PreferredLaunchViewSize =
            new Size(600, 200);
        ApplicationView.PreferredLaunchWindowingMode =
            ApplicationViewWindowingMode.PreferredLaunchViewSize;

        // Button handles PointerPressed/PointerReleased in
        // the Tapped routed event, but we need the container Grid
        // to handle them also. Add a handler for both
        // PointerPressedEvent and PointerReleasedEvent on the input
Grid
        // and set handledEventsToo to true.
        ContainerInput.AddHandler(PointerPressedEvent,
            new PointerEventHandler(ContainerInput_PointerPressed),
true);
        ContainerInput.AddHandler(PointerReleasedEvent,
            new PointerEventHandler(ContainerInput_PointerReleased),
true);
    }

    /// <summary>
    /// PointerReleased handler for all pointer conclusion events.
    /// PointerPressed and PointerReleased events do not always occur
    /// in pairs, so your app should listen for and handle any event
that
    /// might conclude a pointer down (such as PointerExited,
PointerCanceled,
    /// and PointerCaptureLost).
    /// </summary>
    /// <param name="sender">Source of the click event</param>
    /// <param name="e">Event args for the button click routed
event</param>
    private void ContainerInput_PointerReleased(
        object sender, PointerRoutedEventArgs e)
    {
```

```
        // Prevent most handlers along the event route from handling
event again.
        e.Handled = true;

        // Shut down the virtual input device.
        _inputInjector.UninitializeTouchInjection();
    }

    /// <summary>
    /// PointerPressed handler.
    /// PointerPressed and PointerReleased events do not always occur
    /// in pairs. Your app should listen for and handle any event that
    /// might conclude a pointer down (such as PointerExited,
    /// PointerCanceled, and PointerCaptureLost).
    /// </summary>
    /// <param name="sender">Source of the click event</param>
    /// <param name="e">Event args for the button click routed
event</param>
    private void ContainerInput_PointerPressed(
        object sender, PointerRoutedEventArgs e)
    {
        // Prevent most handlers along the event route from
        // handling the same event again.
        e.Handled = true;

        InjectTouchForMouse(e.GetCurrentPoint(ContainerInput));

    }
    ...
}
```

3. Here's the touch input injection function.

   First, we call TryCreate to instantiate the InputInjector object.

   Then, we call InitializeTouchInjection with an InjectedInputVisualizationMode of `Default`.

   After calculating the point of injection, we call InjectedInputTouchInfo to initialize the list of touch points to inject (for this example, we create one touch point corresponding to the mouse input pointer).

   Finally, we call InjectTouchInput twice, the first for a pointer down and the second for a pointer up.

```
C#
```

```
/// <summary>
/// Inject touch input on injection target corresponding
/// to mouse click on input target.
/// </summary>
```

```
/// <param name="pointerPoint">The mouse click pointer.</param>
private void InjectTouchForMouse(PointerPoint pointerPoint)
{
    // Create the touch injection object.
    _inputInjector = InputInjector.TryCreate();

    if (_inputInjector != null)
    {
        _inputInjector.InitializeTouchInjection(
            InjectedInputVisualizationMode.Default);

        // Create a unique pointer ID for the injected touch pointer.
        // Multiple input pointers would require more robust handling.
        uint pointerId = pointerPoint.PointerId + 1;

        // Get the bounding rectangle of the app window.
        Rect appBounds =

Windows.UI.ViewManagement.ApplicationView.GetForCurrentView().VisibleBo
unds;

        // Get the top left screen coordinates of the app window rect.
        Point appBoundsTopLeft = new Point(appBounds.Left,
appBounds.Top);

        // Get a reference to the input injection area.
        GeneralTransform injectArea =
            ContainerInject.TransformToVisual(Window.Current.Content);

        // Get the top left screen coordinates of the input injection
area.
        Point injectAreaTopLeft = injectArea.TransformPoint(new
Point(0, 0));

        // Get the screen coordinates (relative to the input area)
        // of the input pointer.
        int pointerPointX = (int)pointerPoint.Position.X;
        int pointerPointY = (int)pointerPoint.Position.Y;

        // Create the point for input injection and calculate its
screen location.
        Point injectionPoint =
            new Point(
                appBoundsTopLeft.X + injectAreaTopLeft.X +
pointerPointX,
                appBoundsTopLeft.Y + injectAreaTopLeft.Y +
pointerPointY);

        // Create a touch data point for pointer down.
        // Each element in the touch data list represents a single
touch contact.
        // For this example, we're mirroring a single mouse pointer.
        List<InjectedInputTouchInfo> touchData =
            new List<InjectedInputTouchInfo>
            {
```

```
                new InjectedInputTouchInfo
                {
                    Contact = new InjectedInputRectangle
                    {
                        Left = 30, Top = 30, Bottom = 30, Right = 30
                    },
                    PointerInfo = new InjectedInputPointerInfo
                    {
                        PointerId = pointerId,
                        PointerOptions =
                        InjectedInputPointerOptions.PointerDown |
                        InjectedInputPointerOptions.InContact |
                        InjectedInputPointerOptions.New,
                        TimeOffsetInMilliseconds = 0,
                        PixelLocation = new InjectedInputPoint
                        {
                            PositionX = (int)injectionPoint.X ,
                            PositionY = (int)injectionPoint.Y
                        }
                },
                Pressure = 1.0,
                TouchParameters =
                    InjectedInputTouchParameters.Pressure |
                    InjectedInputTouchParameters.Contact
            }
        };

        // Inject the touch input.
        _inputInjector.InjectTouchInput(touchData);

        // Create a touch data point for pointer up.
        touchData = new List<InjectedInputTouchInfo>
        {
            new InjectedInputTouchInfo
            {
                PointerInfo = new InjectedInputPointerInfo
                {
                    PointerId = pointerId,
                    PointerOptions =
InjectedInputPointerOptions.PointerUp
                }
            }
        };

        // Inject the touch input.
        _inputInjector.InjectTouchInput(touchData);
    }
}
```

4. Finally, we handle any Button Click routed events in the input injection area and update the UI with the name of the button clicked.

See also

Topic samples

- Input injection sample (mouse to touch)

Identify input devices

Article • 06/24/2021

Identify the input devices connected to a Windows app device and identify their capabilities and attributes.

> **Important APIs**: Windows.Devices.Input, Windows.UI.Input, Windows.UI.Xaml.Input

Retrieve mouse properties

The Windows.Devices.Input namespace contains the MouseCapabilities class used to retrieve the properties exposed by one or more connected mice. Just create a new **MouseCapabilities** object and get the properties you're interested in.

Note The values returned by the properties discussed here are based on all detected mice: Boolean properties return non-zero if at least one mouse supports a specific capability, and numeric properties return the maximum value exposed by any one mouse.

The following code uses a series of TextBlock elements to display the individual mouse properties and values.

CSharp

```
private void GetMouseProperties()
{
    MouseCapabilities mouseCapabilities = new
Windows.Devices.Input.MouseCapabilities();
    MousePresent.Text = mouseCapabilities.MousePresent != 0 ? "Yes" : "No";
    VertWheel.Text = mouseCapabilities.VerticalWheelPresent != 0 ? "Yes" :
"No";
    HorzWheel.Text = mouseCapabilities.HorizontalWheelPresent != 0 ? "Yes" :
"No";
    SwappedButtons.Text = mouseCapabilities.SwapButtons != 0 ? "Yes" : "No";
    NumButtons.Text = mouseCapabilities.NumberOfButtons.ToString();
}
```

Retrieve keyboard properties

The Windows.Devices.Input namespace contains the KeyboardCapabilities class used to retrieve whether a keyboard is connected. Just create a new **KeyboardCapabilities** object and get the KeyboardPresent property.

The following code uses a TextBlock element to display the keyboard property and value.

CSharp

```
private void GetKeyboardProperties()
{
    KeyboardCapabilities keyboardCapabilities = new
Windows.Devices.Input.KeyboardCapabilities();
    KeyboardPresent.Text = keyboardCapabilities.KeyboardPresent != 0 ? "Yes"
: "No";
}
```

Retrieve touch properties

The Windows.Devices.Input namespace contains the TouchCapabilities class used to retrieve whether any touch digitizers are connected. Just create a new **TouchCapabilities** object and get the properties you're interested in.

Note The values returned by the properties discussed here are based on all detected touch digitizers: Boolean properties return non-zero if at least one digitizer supports a specific capability, and numeric properties return the maximum value exposed by any one digitizer.

The following code uses a series of TextBlock elements to display the touch properties and values.

CSharp

```
private void GetTouchProperties()
{
    TouchCapabilities touchCapabilities = new
Windows.Devices.Input.TouchCapabilities();
    TouchPresent.Text = touchCapabilities.TouchPresent != 0 ? "Yes" : "No";
    Contacts.Text = touchCapabilities.Contacts.ToString();
}
```

Retrieve pointer properties

The Windows.Devices.Input namespace contains the PointerDevice class used to retrieve whether any detected devices support pointer input (touch, touchpad, mouse, or pen). Just create a new **PointerDevice** object and get the properties you're interested in.

Note The values returned by the properties discussed here are based on all detected pointer devices: Boolean properties return non-zero if at least one device supports a specific capability, and numeric properties return the maximum value exposed by any one pointer device.

The following code uses a table to display the properties and values for each pointer device.

CSharp

```
private void GetPointerDevices()
{
    IReadOnlyList<PointerDevice> pointerDevices =
Windows.Devices.Input.PointerDevice.GetPointerDevices();
    int gridRow = 0;
    int gridColumn = 0;

    for (int i = 0; i < pointerDevices.Count; i++)
    {
        // Pointer device type.
        TextBlock textBlock1 = new TextBlock();
        Grid_PointerProps.Children.Add(textBlock1);
        textBlock1.Text = (i + 1).ToString() + " Pointer Device Type:";
        Grid.SetRow(textBlock1, gridRow);
        Grid.SetColumn(textBlock1, gridColumn);

        TextBlock textBlock2 = new TextBlock();
        textBlock2.Text = pointerDevices[i].PointerDeviceType.ToString();
        Grid_PointerProps.Children.Add(textBlock2);
        Grid.SetRow(textBlock2, gridRow++);
        Grid.SetColumn(textBlock2, gridColumn + 1);

        // Is external?
        TextBlock textBlock3 = new TextBlock();
        Grid_PointerProps.Children.Add(textBlock3);
        textBlock3.Text = (i + 1).ToString() + " Is External?";
        Grid.SetRow(textBlock3, gridRow);
        Grid.SetColumn(textBlock3, gridColumn);

        TextBlock textBlock4 = new TextBlock();
        Grid_PointerProps.Children.Add(textBlock4);
        textBlock4.Text = pointerDevices[i].IsIntegrated.ToString();
        Grid.SetRow(textBlock4, gridRow++);
        Grid.SetColumn(textBlock4, gridColumn + 1);

        // Maximum contacts.
```

```
        TextBlock textBlock5 = new TextBlock();
        Grid_PointerProps.Children.Add(textBlock5);
        textBlock5.Text = (i + 1).ToString() + " Max Contacts:";
        Grid.SetRow(textBlock5, gridRow);
        Grid.SetColumn(textBlock5, gridColumn);

        TextBlock textBlock6 = new TextBlock();
        Grid_PointerProps.Children.Add(textBlock6);
        textBlock6.Text = pointerDevices[i].MaxContacts.ToString();
        Grid.SetRow(textBlock6, gridRow++);
        Grid.SetColumn(textBlock6, gridColumn + 1);

        // Physical device rectangle.
        TextBlock textBlock7 = new TextBlock();
        Grid_PointerProps.Children.Add(textBlock7);
        textBlock7.Text = (i + 1).ToString() + " Physical Device Rect:";
        Grid.SetRow(textBlock7, gridRow);
        Grid.SetColumn(textBlock7, gridColumn);

        TextBlock textBlock8 = new TextBlock();
        Grid_PointerProps.Children.Add(textBlock8);
        textBlock8.Text = pointerDevices[i].PhysicalDeviceRect.X.ToString()
+ "," +
            pointerDevices[i].PhysicalDeviceRect.Y.ToString() + "," +
            pointerDevices[i].PhysicalDeviceRect.Width.ToString() + "," +
            pointerDevices[i].PhysicalDeviceRect.Height.ToString();
        Grid.SetRow(textBlock8, gridRow++);
        Grid.SetColumn(textBlock8, gridColumn + 1);

        // Screen rectangle.
        TextBlock textBlock9 = new TextBlock();
        Grid_PointerProps.Children.Add(textBlock9);
        textBlock9.Text = (i + 1).ToString() + " Screen Rect:";
        Grid.SetRow(textBlock9, gridRow);
        Grid.SetColumn(textBlock9, gridColumn);

        TextBlock textBlock10 = new TextBlock();
        Grid_PointerProps.Children.Add(textBlock10);
        textBlock10.Text = pointerDevices[i].ScreenRect.X.ToString() + "," +
            pointerDevices[i].ScreenRect.Y.ToString() + "," +
            pointerDevices[i].ScreenRect.Width.ToString() + "," +
            pointerDevices[i].ScreenRect.Height.ToString();
        Grid.SetRow(textBlock10, gridRow++);
        Grid.SetColumn(textBlock10, gridColumn + 1);

        gridColumn += 2;
        gridRow = 0;
    }
```

Related articles

Samples

- Basic input sample
- Low latency input sample
- User interaction mode sample

Archive samples

- Input: Device capabilities sample

Drag and drop

Article • 07/12/2024

Drag and drop is an intuitive way to transfer data within an application or between applications on the Windows desktop. Drag and drop lets the user transfer data between applications or within an application using a standard gesture (press-hold-and-pan with the finger or press-and-pan with a mouse or a stylus).

> **Important APIs**: CanDrag property, AllowDrop property

The drag source, which is the application or area where the drag gesture is triggered, provides the data to be transferred by filling a data package object that can contain standard data formats, including text, RTF, HTML, bitmaps, storage items or custom data formats. The source also indicates the kind of operations it supports: copy, move or link. When the pointer is released, drop occurs. The drop target, which is the application or area underneath the pointer, processes the data package and returns the type of operation it performed.

During drag and drop, the drag UI provides a visual indication of the type of drag-and-drop operation that's taking place. This visual feedback is initially provided by the source but can be changed by the targets as the pointer moves over them.

Modern drag and drop is available on all devices that support UWP. It allows data transfer between or within any kind of application, including Classic Windows apps, although this article focuses on the XAML API for modern drag and drop. Once implemented, drag and drop works seamlessly in all directions, including app-to-app, app-to-desktop, and desktop-to app.

Here's an overview of what you need to do to enable drag and drop in your app:

1. Enable dragging on an element by setting its **CanDrag** property to true.
2. Build the data package. The system handles images and text automatically, but for other content, you'll need to handle the **DragStarting** and **DropCompleted** events and use them to construct your own data package.
3. Enable dropping by setting the **AllowDrop** property to **true** on all the elements that can receive dropped content.
4. Handle the **DragOver** event to let the system know what type of drag operations the element can receive.
5. Process the **Drop** event to receive the dropped content.

Enable dragging

To enable dragging on an element, set its **CanDrag** property to **true**. This make the element—and the elements it contains, in the case of collections like ListView—draggable.

Be specific about what's draggable. Users won't want to drag everything in your app, only certain items, such as images or text.

Here's how to set **CanDrag**.

XML

```
<Image x:Name="Image" CanDrag="True" Margin="10,292,10,0" Height="338"></Image>
```

You don't need to do any other work to allow dragging, unless you want to customize the UI (which is covered later in this article). Dropping requires a few more steps.

Construct a data package

In most cases, the system will construct a data package for you. The system automatically handles:

- Images
- Text

For other content, you'll need to handle the **DragStarting** and **DropCompleted** events and use them to construct your own DataPackage.

Enable dropping

The following markup shows how the **AllowDrop** property can be used to specify that an area of the app is a valid drop target for a dragged item (the specified area must not have a null background, it must be able to receive pointer input, and the item cannot be dropped anywhere other than the specified area).

> ⓘ **Note**
>
> Typically, a UI element has a null background by default. If you want users to be able to drop an item anywhere within your app, the app background cannot be null (set `Background="Transparent"` if the background should not be visible).

XML

```
<Grid AllowDrop="True" DragOver="Grid_DragOver" Drop="Grid_Drop"
      Background="LightBlue" Margin="10,10,10,353">
    <TextBlock>Drop anywhere in the blue area</TextBlock>
</Grid>
```

Handle the DragOver event

The DragOver event fires when a user has dragged an item over your app, but not yet dropped it. In this handler, you need to specify what kind of operations your app supports by using the AcceptedOperation property. Copy is the most common.

C#

```
private void Grid_DragOver(object sender, DragEventArgs e)
{
    e.AcceptedOperation = DataPackageOperation.Copy;
}
```

Process the Drop event

The Drop event occurs when the user releases items in a valid drop area. Process them by using the DataView property.

For simplicity in the example below, we'll assume the user dropped a single photo and access it directly. In reality, users can drop multiple items of varying formats simultaneously. Your app should handle this possibility by checking what types of files were dropped and how many there are, and process each accordingly. You should also consider notifying the user if they're trying to do something your app doesn't support.

C#

```
private async void Grid_Drop(object sender, DragEventArgs e)
{
    if (e.DataView.Contains(StandardDataFormats.StorageItems))
    {
        var items = await e.DataView.GetStorageItemsAsync();
        if (items.Count > 0)
        {
            var storageFile = items[0] as StorageFile;
            var bitmapImage = new BitmapImage();
            bitmapImage.SetSource(await storageFile.OpenAsync(FileAccessMode.Read));
            // Set the image on the main page to the dropped image
            Image.Source = bitmapImage;
        }
    }
}
```

Customize the UI

The system provides a default UI for dragging and dropping. However, you can also choose to customize various parts of the UI by setting custom captions and glyphs, or by opting not to show a UI at all. To customize the UI, use the DragEventArgs.DragUIOverride property.

C#

```
private void Grid_DragOverCustomized(object sender, DragEventArgs e)
{
    e.AcceptedOperation = DataPackageOperation.Copy;
    e.DragUIOverride.Caption = "Custom text here"; // Sets custom UI text
    // Sets a custom glyph
    e.DragUIOverride.SetContentFromBitmapImage(
        new BitmapImage(
            new Uri("ms-appx:///Assets/CustomImage.png",
UriKind.RelativeOrAbsolute)));
    e.DragUIOverride.IsCaptionVisible = true; // Sets if the caption is visible
    e.DragUIOverride.IsContentVisible = true; // Sets if the dragged content is
visible
    e.DragUIOverride.IsGlyphVisible = true; // Sets if the glyph is visibile
}
```

Open a context menu on an item you can drag with touch

When using touch, dragging a **UIElement** and opening its context menu share similar touch gestures; each begins with a press and hold. Here's how the system disambiguates between the two actions for elements in your app that support both:

- If a user presses and holds an item and begins dragging it within 500 milliseconds, the item is dragged and the context menu is not shown.
- If the user presses and holds but does not drag within 500 milliseconds, the context menu is opened.
- After the context menu is open, if the user tries to drag the item (without lifting their finger), the context menu is dismissed and the drag will start.

Designate an item in a ListView or GridView as a folder

You can specify a **ListViewItem** or **GridViewItem** as a folder. This is particularly useful for TreeView and File Explorer scenarios. To do so, explicitly set the **AllowDrop** property to **True** on that item.

The system will automatically show the appropriate animations for dropping into a folder versus a non-folder item. Your app code must continue to handle the **Drop** event on the folder item (as well as on the non-folder item) in order to update the data source and add the dropped item to the target folder.

Enable drag and drop reordering within ListViews

ListViews support drag-based reordering out of the box, using an API very similar to the **CanDrop** API described in this article. At minimum, you add the **AllowDrop** and

CanReorderItems properties.

See **ListViewBase.CanReorderItems** for more information.

Implementing custom drag and drop

The UIElement class does most of the work of implementing drag-and-drop for you. But if you want, you can implement your own version by using the APIs below.

Expand table

Functionality	WinAppSDK API Microsoft.UI.Input.DragDrop namespace	UWP API Windows.Applicationmodel.DataTransfer.DragDrop.Core namespace
DragPrimitive	DragOperation	CoreDragOperation
Create a data package	DataPackage	same
Hand off drag to the shell	DragOperation.StartAsync	CoreDragOperation.StartAsync
Receive drop from the shell	DragDropManager.TargetRequested ICoreDropOperationTarget	CoreDragDropManager.TargetRequested ICoreDropOperationTarget

See also

- App-to-app communication
- AllowDrop
- CanDrag
- DragOver
- AcceptedOperation
- DataView
- DragUIOverride
- Drop
- IsDragSource
- **DragStarting**
- **DropCompleted**

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

can also create and review issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Optical zoom and resizing

Article • 10/31/2023

This article describes Windows zooming and resizing elements and provides user experience guidelines for using these interaction mechanisms in your apps.

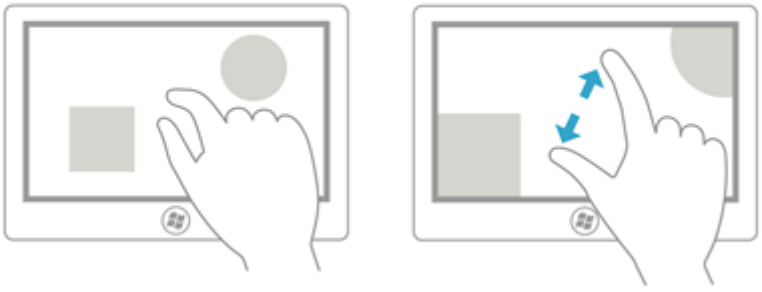
> **Important APIs**: Windows.UI.Input, Input (XAML)

Optical zoom lets users magnify their view of the content within a content area (it is performed on the content area itself), whereas resizing enables users to change the relative size of one or more objects without changing the view of the content area (it is performed on the objects within the content area).

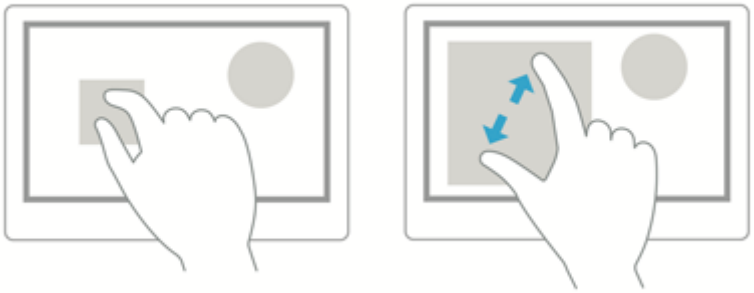
Both optical zoom and resizing interactions are performed through the pinch and stretch gestures (moving fingers farther apart zooms in and moving them closer together zooms out), or by holding the Ctrl key down while scrolling the mouse scroll wheel, or by holding the Ctrl key down (with the Shift key, if no numeric keypad is available) and pressing the plus (+) or minus (-) key.

The following diagrams demonstrate the differences between resizing and optical zooming.

Optical zoom: User selects an area, and then zooms into the entire area.



Resize: User selects an object within an area, and resizes that object.



Note Optical zoom shouldn't be confused with Semantic Zoom. Although the same gestures are used for both interactions, semantic zoom refers to the presentation and navigation of content organized within a single view (such as the folder structure of a computer, a library of documents, or a photo album).

Dos and don'ts

Use the following guidelines for apps that support either resizing or optical zooming:

- If maximum and minimum size constraints or boundaries are defined, use visual feedback to demonstrate when the user reaches or exceeds those boundaries.
- Use snap points to influence zooming and resizing behavior by providing logical points at which to stop the manipulation and ensure a specific subset of content is displayed in the viewport. Provide snap points for common zoom levels or logical views to make it easier for a user to select those levels. For example, photo apps might provide a resizing snap point at 100% or, in the case of mapping apps, snap points might be useful at city, state, and country/region views.

  Snap points enable users to be imprecise and still achieve their goals. If you're using XAML, see the snap points properties of **ScrollViewer**.

  There are two types of snap-points:
  - Proximity - After the contact is lifted, a snap point is selected if inertia stops within a distance threshold of the snap point. Proximity snap points still allow a zoom or resize to end between snap points.
  - Mandatory - The snap point selected is the one that immediately precedes or succeeds the last snap point crossed before the contact was lifted (depending on the direction and velocity of the gesture). A manipulation must end on a mandatory snap point.

- Use inertia physics. These include the following:
  - Deceleration: Occurs when the user stops pinching or stretching. This is similar to sliding to a stop on a slippery surface.
  - Bounce: A slight bounce-back effect occurs when a size constraint or boundary is passed.
- Space controls according to the Guidelines for targeting.
- Provide scaling handles for constrained resizing. Isometric, or proportional, resizing is the default if the handles are not specified.
- Don't use zooming to navigate the UI or expose additional controls within your app, use a panning region instead. For more info on panning, see Guidelines for panning.
- Don't put resizable objects within a resizable content area. Exceptions to this include:
  - Drawing applications where resizable items can appear on a resizable canvas or art board.
  - Webpages with an embedded object such as a map.

  Note In all cases, the content area is resized unless all touch points are within the resizable object.

Related articles

Samples

- Basic input sample
- Low latency input sample
- User interaction mode sample
- Focus visuals sample

Archive samples

- Input: XAML user input events sample
- Input: Device capabilities sample
- Input: Touch hit testing sample
- XAML scrolling, panning, and zooming sample
- Input: Simplified ink sample
- Input: Manipulations and gestures sample
- DirectX touch input sample

Guidelines for panning

Article • 06/24/2021

Panning or scrolling lets users navigate within a single view, to display the content of the view that does not fit within the viewport. Examples of views include the folder structure of a computer, a library of documents, or a photo album.

> **Important APIs**: Windows.UI.Input, Windows.UI.Xaml.Input

Dos and don'ts

Panning indicators and scroll bars

- Ensure panning/scrolling is possible before loading content into your app.
- Display panning indicators and scroll bars to provide location and size cues. Hide them if you provide a custom navigation feature.

  Note Unlike standard scroll bars, panning indicators are purely informative. They are not exposed to input devices and cannot be manipulated in any way.

Single-axis panning (one-dimensional overflow)

- Use one-axis panning for content regions that extend beyond one viewport boundary (vertical or horizontal).
  - Vertical panning for a one-dimensional list of items.
  - Horizontal panning for a grid of items.
- Don’t use mandatory snap-points with single-axis panning if a user must be able to pan and stop between snap-points. Mandatory snap-points guarantee that the user will stop on a snap-point. Use proximity snap-points instead.

Freeform panning (two-dimensional overflow)

- Use two-axis panning for content regions that extend beyond both viewport boundaries (vertical and horizontal).
  - Override the default rails behavior and use freeform panning for unstructured content where the user is likely to move in multiple directions.
- Freeform panning is typically suited to navigating within images or maps.

Paged view

- Use mandatory snap-points when the content is composed of discrete elements or you want to display an entire element. This can include pages of a book or magazine, a column of items, or individual images.
  - A snap-point should be placed at each logical boundary.
  - Each element should be sized or scaled to fit the view.

Logical and key points

- Use proximity snap-points if there are key points or logical places in the content that a user will likely stop. For example, a section header.

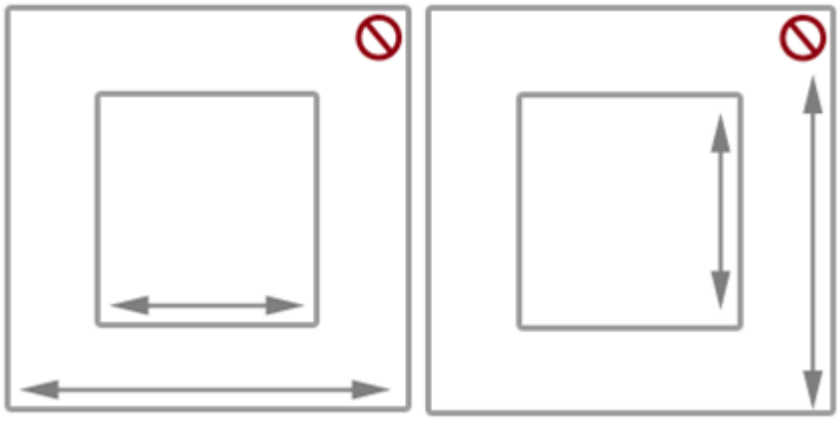
- If maximum and minimum size constraints or boundaries are defined, use visual feedback to demonstrate when the user reaches or exceeds those boundaries.

Chaining embedded or nested content

- Use single-axis panning (typically horizontal) and column layouts for text and grid-based content. In these cases, content typically wraps and flows naturally from column to column and keeps the user experience consistent and discoverable across Windows apps.

- Don't use embedded pannable regions to display text or item lists. Because the panning indicators and scroll bars are displayed only when the input contact is detected within the region, it is not an intuitive or discoverable user experience.

- Don't chain or place one pannable region within another pannable region if they both pan in the same direction, as shown here. This can result in the parent area being panned unintentionally when a boundary for the child area is reached. Consider making the panning axis perpendicular.



Additional usage guidance

Panning with touch, by using a swipe or slide gesture with one or more fingers, is like scrolling with the mouse. The panning interaction is most similar to rotating the mouse wheel or sliding the scroll box, rather than clicking the scroll bar. Unless a distinction is made in an API or required by some device-specific Windows UI, we simply refer to both interactions as panning.

> **Windows 10 Fall Creators Update - Behavior change** By default, instead of text selection, an active pen now scrolls/pans in Windows apps (like touch, touchpad, and passive pen). If your app depends on the previous behavior, you can override pen scrolling and revert to the previous behavior. For details, see the API reference topic for the ScrollViewer Class.

Depending on the input device, the user pans within a pannable region by using one of these:

- A mouse, touchpad, or active pen/stylus to click the scroll arrows, drag the scroll box, or click within the scroll bar.
- The wheel button of the mouse to emulate dragging the scroll box.
- The extended buttons (XBUTTON1 and XBUTTON2), if supported by the mouse.
- The keyboard arrow keys to emulate dragging the scroll box or the page keys to emulate clicking within the scroll bar.
- Touch, touchpad, or passive pen/stylus to slide or swipe the fingers in the desired direction.

Sliding involves moving the fingers slowly in the panning direction. This results in a one-to-one relationship, where the content pans at the same speed and distance as the fingers. Swiping, which involves rapidly sliding and lifting the fingers, results in the following physics being applied to the panning animation:

- Deceleration (inertia): Lifting the fingers causes panning to start decelerating. This is similar to sliding to a stop on a slippery surface.
- Absorption: Panning momentum during deceleration causes a slight bounce-back effect if either a snap point or a content area boundary is reached.

Types of panning

Windows supports three types of panning:

- Single axis - panning is supported in one direction only (horizontal or vertical).
- Rails - panning is supported in all directions. However, once the user crosses a distance threshold in a specific direction, then panning is restricted to that axis.
- Freeform - panning is supported in all directions.

Panning UI

The interaction experience for panning is unique to the input device while still providing similar functionality.

There are two panning display modes based on the input device detected:

- Panning indicators for touch.
- Scroll bars for other input devices, including mouse, touchpad, keyboard, and stylus.

Note Panning indicators are only visible when the touch contact is within the pannable region. Similarly, the scroll bar is only visible when the mouse cursor, pen/stylus cursor, or keyboard focus is within the scrollable region.

Panning indicators Panning indicators are similar to the scroll box in a scroll bar. They indicate the proportion of displayed content to total pannable area and the relative position of the displayed content in the pannable area.

The following diagram shows two pannable areas of different lengths and their panning indicators.

Panning behaviors Snap points Panning with the swipe gesture introduces inertia behavior into the interaction when the touch contact is lifted. With inertia, the content continues to pan until some distance threshold is reached without direct input from the user. Use snap points to modify this inertia behavior.

Snap points specify logical stops in your app content. Cognitively, snap points act as a paging mechanism for the user and minimize fatigue from excessive sliding or swiping in large pannable regions. With them, you can handle imprecise user input and ensure a specific subset of content or key information is displayed in the viewport.

There are two types of snap-points:

- Proximity - After the contact is lifted, a snap point is selected if inertia stops within a distance threshold of the snap point. Panning can still stop between proximity snap points.
- Mandatory - The snap point selected is the one that immediately precedes or succeeds the last snap point crossed before the contact was lifted (depending on the direction and velocity of the gesture). Panning must stop on a mandatory snap point.

Panning snap-points are useful for applications such as web browsers and photo albums that emulate paginated content or have logical groupings of items that can be dynamically regrouped to fit within a viewport or display.

The following diagrams show how panning to a certain point and releasing causes the content to automatically pan to a logical location.

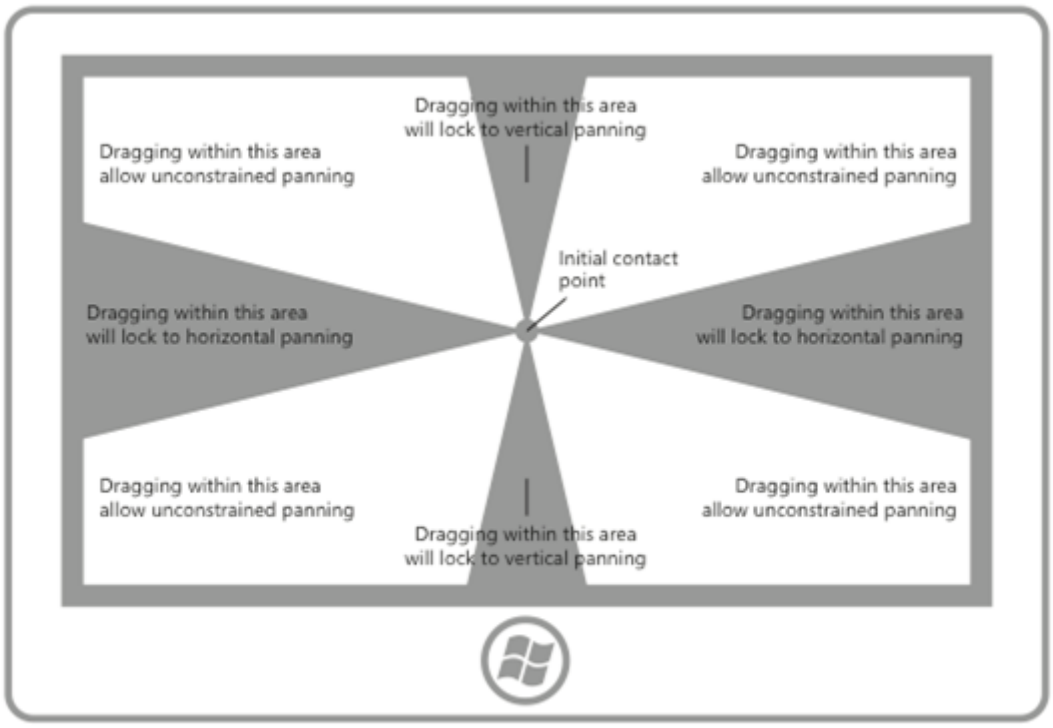
Swipe to pan.

Lift touch contact.

Pannable region stops at the snap point, not where the touch contact was lifted.

Rails Content can be wider and taller than the dimensions and resolution of a display device. For this reason, two-dimensional panning (horizontal and vertical) is often necessary. Rails improve the user experience in these cases by emphasizing panning along the axis of motion (vertical or horizontal).

The following diagram demonstrates the concept of rails.

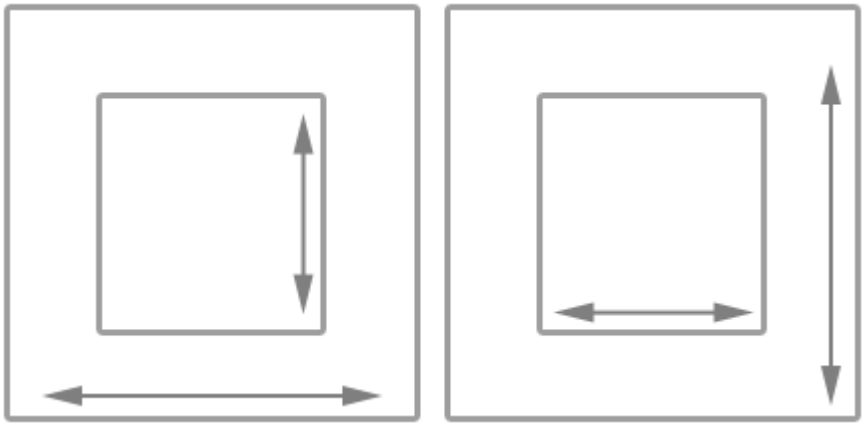


Chaining embedded or nested content

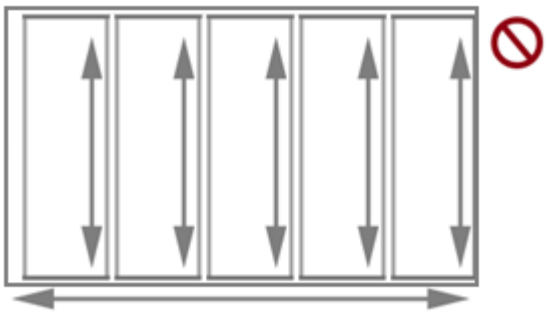
After a user hits a zoom or scroll limit on an element that has been nested within another zoomable or scrollable element, you can specify whether that parent element should continue the zooming or scrolling operation begun in its child element. This is called zoom or scroll chaining.

Chaining is used for panning within a single-axis content area that contains one or more single-axis or freeform panning regions (when the touch contact is within one of these child regions). When the panning boundary of the child region is reached in a specific direction, panning is then activated on the parent region in the same direction.

When a pannable region is nested inside another pannable region it's important to specify enough space between the container and the embedded content. In the following diagrams, one pannable region is placed inside another pannable region, each going in perpendicular directions. There is plenty of space for users to pan in each region.



Without enough space, as shown in the following diagram, the embedded pannable region can interfere with panning in the container and result in unintentional panning in one or more of the pannable regions.



This guidance is also useful for apps such as photo albums or mapping apps that support unconstrained panning within an individual image or map while also supporting single-axis panning within the album (to the previous or next images) or details area. In apps that provide a detail or options area corresponding to a freeform panning image or map, we recommend that the page layout start with the details and options area as the unconstrained panning area of the image or map might interfere with panning to the details area.

Related articles

- Custom user interactions
- Optimize ListView and GridView
- Keyboard accessibility

Samples

- Basic input sample
- Low latency input sample
- User interaction mode sample
- Focus visuals sample

Archive samples

- Input: XAML user input events sample
- Input: Device capabilities sample
- Input: Touch hit testing sample
- XAML scrolling, panning, and zooming sample
- Input: Simplified ink sample
- Input: Manipulations and gestures sample
- DirectX touch input sample

Rotation

Article • 06/24/2021

This article describes the new Windows UI for rotation and provides user experience guidelines that should be considered when using this new interaction mechanism in your Windows app.

> **Important APIs**: Windows.UI.Input, Windows.UI.Xaml.Input

Dos and don'ts

- Use rotation to help users directly rotate UI elements.

Additional usage guidance

Overview of rotation

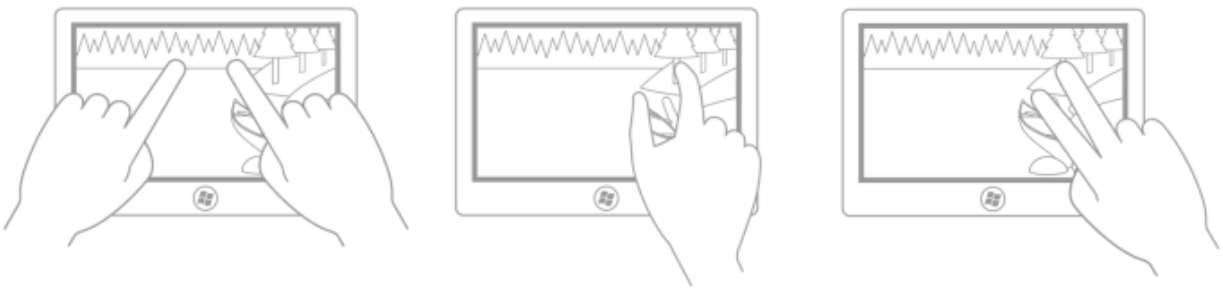
Rotation is the touch-optimized technique used by Windows apps to enable users to turn an object in a circular direction (clockwise or counter-clockwise).

Depending on the input device, the rotation interaction is performed using:

- A mouse or active pen/stylus to move the rotation gripper of a selected object.
- Touch or passive pen/stylus to turn the object in the desired direction using the rotate gesture.

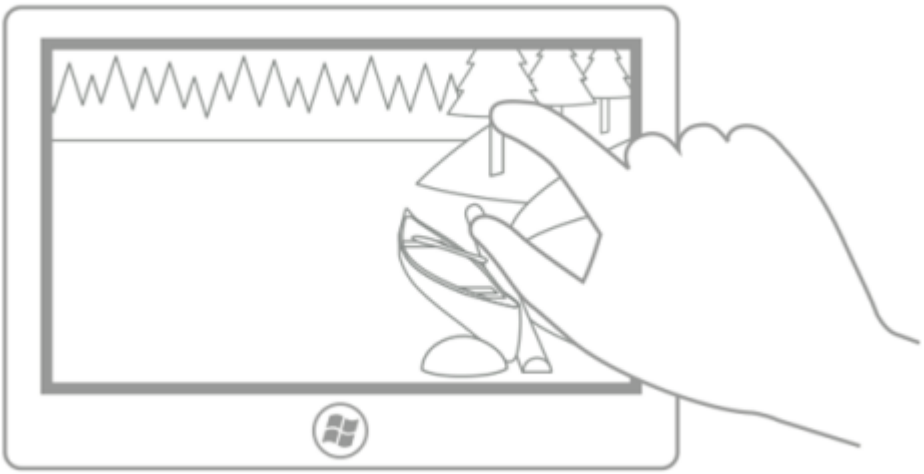
When to use rotation

Use rotation to help users directly rotate UI elements. The following diagrams show some of the supported finger positions for the rotation interaction.

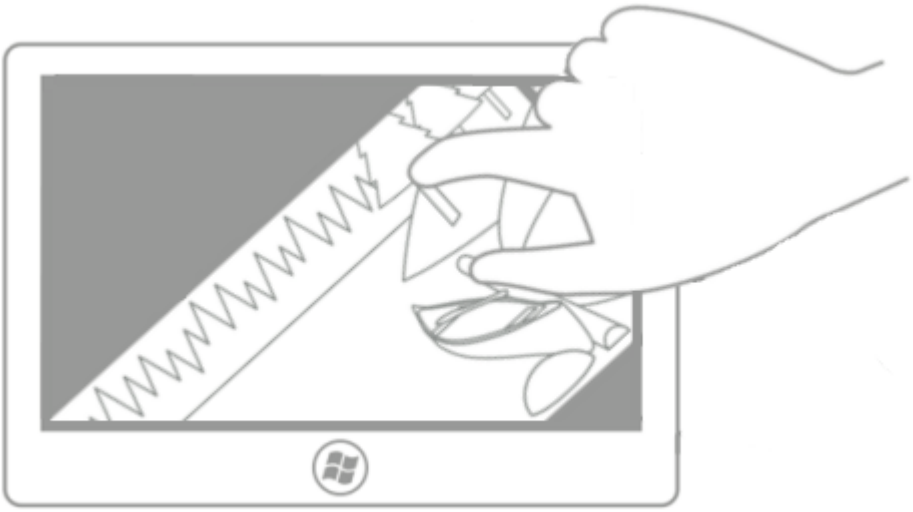


Note Intuitively, and in most cases, the rotation point is one of the two touch points unless the user can specify a rotation point unrelated to the contact points (for example, in a drawing or layout application). The following images demonstrate how the user experience can be degraded if the rotation point is not constrained in this way.

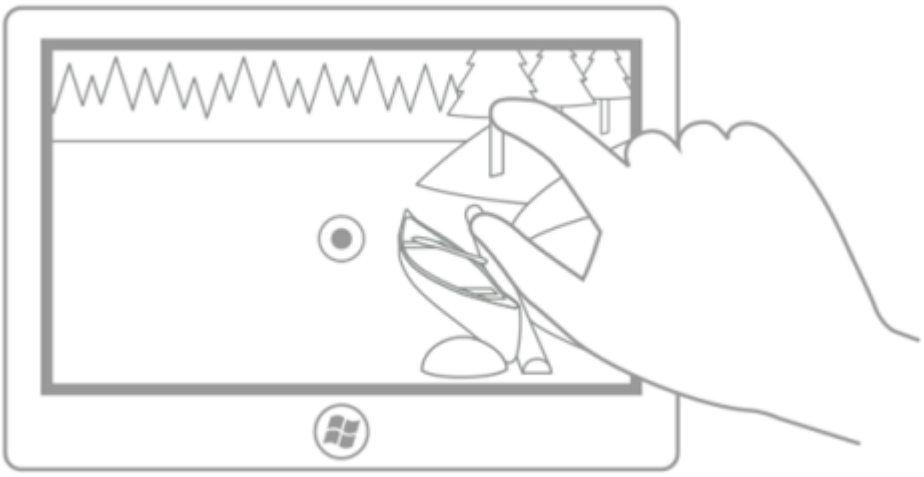
This first picture shows the initial (thumb) and secondary (index finger) touch points: the index finger is touching a tree and the thumb is touching a log.



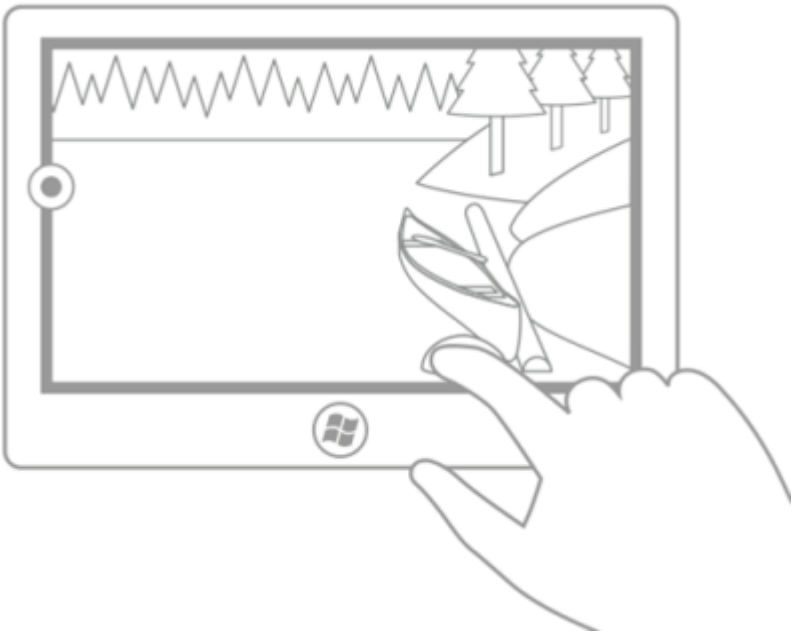
In this second picture, rotation is performed around the initial (thumb) touch point. After the rotation, the index finger is still touching the tree trunk and the thumb is still touching the log (the rotation point).



In this third picture, the center of rotation has been defined by the application (or set by the user) to be the center point of the picture. After the rotation, because the picture did not rotate around one of the fingers, the illusion of direct manipulation is broken (unless the user has chosen this setting).



In this last picture, the center of rotation has been defined by the application (or set by the user) to be a point in the middle of the left edge of the picture. Again, unless the user has chosen this setting, the illusion of direct manipulation is broken in this case.



Windows 10 supports three types of rotation: free, constrained, and combined.

Type	Description
Free rotation	Free rotation enables a user to rotate content freely anywhere in a 360 degree arc. When the user releases the object, the object remains in the chosen position. Free rotation is useful for drawing and layout applications such as Microsoft PowerPoint, Word, Visio, and Paint; and Adobe Photoshop, Illustrator, and Flash.

Type	Description
Constrained rotation	Constrained rotation supports free rotation during the manipulation but enforces snap points at 90 degree increments (0, 90, 180, and 270) upon release. When the user releases the object, the object automatically rotates to the nearest snap point. Constrained rotation is the most common method of rotation, and it functions in a similar way to scrolling content. Snap points let a user be imprecise and still achieve their goal. Constrained rotation is useful for applications such as web browsers and photo albums.
Combined rotation	Combined rotation supports free rotation with zones (similar to rails in Guidelines for panning) at each of the 90 degree snap points enforced by constrained rotation. If the user releases the object outside of one of 90 degree zones, the object remains in that position; otherwise, the object automatically rotates to a snap point. **Note** A user interface rail is a feature in which an area around a target constrains movement towards some specific value or location to influence its selection.

Related topics

Samples

- Basic input sample
- Low latency input sample
- User interaction mode sample
- Focus visuals sample

Archive samples

- Input: XAML user input events sample

- Input: Device capabilities sample
- Input: Touch hit testing sample
- XAML scrolling, panning, and zooming sample
- Input: Simplified ink sample
- Input: Gestures and manipulations with GestureRecognizer
- Input: Manipulations and gestures sample
- DirectX touch input sample

Guidelines for touch targets

Article • 10/04/2021

All interactive UI elements in your Windows application must be large enough for users to accurately access and use, regardless of device type or input method.

Supporting touch input (and the relatively imprecise nature of the touch contact area) requires further optimization with respect to target size and control layout as the larger, more complex set of input data reported by the touch digitizer is used to determine the user's intended (or most likely) target.

All UWP controls have been designed with default touch target sizes and layouts that enable you to build visually balanced and appealing apps that are comfortable, easy to use, and inspire confidence.

In this topic, we describe these default behaviors so you can design your app for maximum usability using both platform controls and custom controls (should your app require them).

> **Important APIs**: Windows.UI.Core, Windows.UI.Input, Windows.UI.Xaml.Input

Fluent Standard sizing

Fluent Standard sizing was created to provide a balance between information density and user comfort. Effectively, all items on the screen align to a 40x40 effective pixels (epx) target, which lets UI elements align to a grid and scale appropriately based on system level scaling.

> ⓘ **Note**
>
> For more info on effective pixels and scaling, see **Screen sizes and breakpoints**
>
> For more info on system level scaling, see **Alignment, margin, padding**.

Fluent Compact sizing

Applications can display a higher level of information density with *Fluent Compact sizing*. Compact sizing aligns UI elements to a 32x32 epx target, which lets UI elements to align to a tighter grid and scale appropriately based on system level scaling.

Examples

Compact sizing can be applied at the page or grid level.

Page level

XAML

```
<Page.Resources>
    <ResourceDictionary Source="ms-
appx:///Microsoft.UI.Xaml/DensityStyles/Compact.xaml" />
</Page.Resources>
```

Grid level

XAML

```
<Grid>
    <Grid.Resources>
        <ResourceDictionary Source="ms-
appx:///Microsoft.UI.Xaml/DensityStyles/Compact.xaml" />
    </Grid.Resources>
</Grid>
```

Target size

In general, set your touch target size to 7.5mm square range (40x40 pixels on a 135 PPI display at a 1.0x scaling plateau). Typically, UWP controls align with 7.5mm touch target (this can vary based on the specific control and any common usage patterns). See Control size and density for more detail.

These target size recommendations can be adjusted as required by your particular scenario. Here are some things to consider:

- Frequency of Touches - consider making targets that are repeatedly or frequently pressed larger than the minimum size.
- Error Consequence - targets that have severe consequences if touched in error should have greater padding and be placed further from the edge of the content area. This is especially true for targets that are touched frequently.
- Position in the content area.
- Form factor and screen size.
- Finger posture.

- Touch visualizations.

Related articles

- Design basics for Windows apps
- Screen sizes and breakpoints
- Control size and density
- Alignment, margin, padding

Samples

- Basic input sample
- Low latency input sample
- User interaction mode sample
- Focus visuals sample

Archive samples

- Input: XAML user input events sample
- Input: Device capabilities sample
- Input: Touch hit testing sample
- XAML scrolling, panning, and zooming sample
- Input: Simplified ink sample
- Input: Manipulations and gestures sample
- DirectX touch input sample

Guidelines for visual feedback

Article • 06/24/2021

Use visual feedback to show users when their interactions are detected, interpreted, and handled. Visual feedback can help users by encouraging interaction. It indicates the success of an interaction, which improves the user's sense of control. It also relays system status and reduces errors.

> **Important APIs**: Windows.Devices.Input, Windows.UI.Input, Windows.UI.Core

Recommendations

- Try to limit modifications of a control template to those directly related to your design intent, as extensive changes can impact the performance and accessibility of both the control and your application.
  - See XAML styles for more info on customizing the properties of a control, including visual state properties.
  - See the UserControl Class for details on making changes to a control template
  - Consider creating your own custom templated control if you need to make significant changes to a control template. For an example of a custom templated control, see the Custom Edit Control sample.
- Don't use touch visualizations in situations where they might interfere with the use of the app. For more info, see **ShowGestureFeedback**.
- Don't display feedback unless it is absolutely necessary. Keep the UI clean and uncluttered by not showing visual feedback unless you are adding value that is not available elsewhere.
- Try not to dramatically customize the visual feedback behaviors of the built-in Windows gestures, as this can create an inconsistent and confusing user experience.

Additional usage guidance

Contact visualizations are especially critical for touch interactions that require accuracy and precision. For example, your app should clearly indicate the location of a tap to let a user know if they missed their target, how much they missed it by, and what adjustments they must make.

Using the default XAML platform controls available ensures that your app works correctly on all devices and in all input situations. If your app features custom

interactions that require customized feedback, you should ensure the feedback is appropriate, spans input devices, and doesn't distract a user from their task. This can be a particular issue in game or drawing apps, where the visual feedback might conflict with or obscure critical UI.

> **ⓘ Important**
>
> We don't recommend changing the interaction behavior of the built-in gestures.

Feedback Across Devices

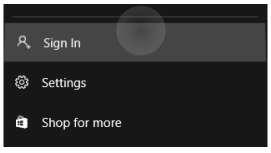
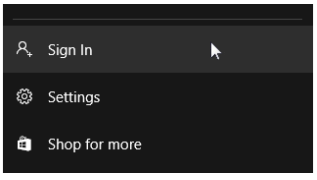
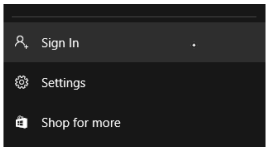
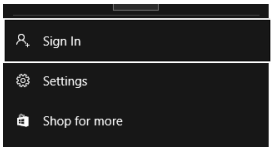
Visual feedback is generally dependent on the input device (touch, touchpad, mouse, pen/stylus, keyboard, and so on). For example, the built-in feedback for a mouse usually involves moving and changing the cursor, while touch and pen require contact visualizations, and keyboard input and navigation uses focus rectangles and highlighting.

Use ShowGestureFeedback to set feedback behavior for the platform gestures.

If customizing feedback UI, ensure you provide feedback that supports, and is suitable for, all input modes.

Here are some examples of built-in contact visualizations in Windows.

Expand table

			
Touch visualization	Mouse/touchpad visualization	Pen visualization	Keyboard visualization

High Visibility Focus Visuals

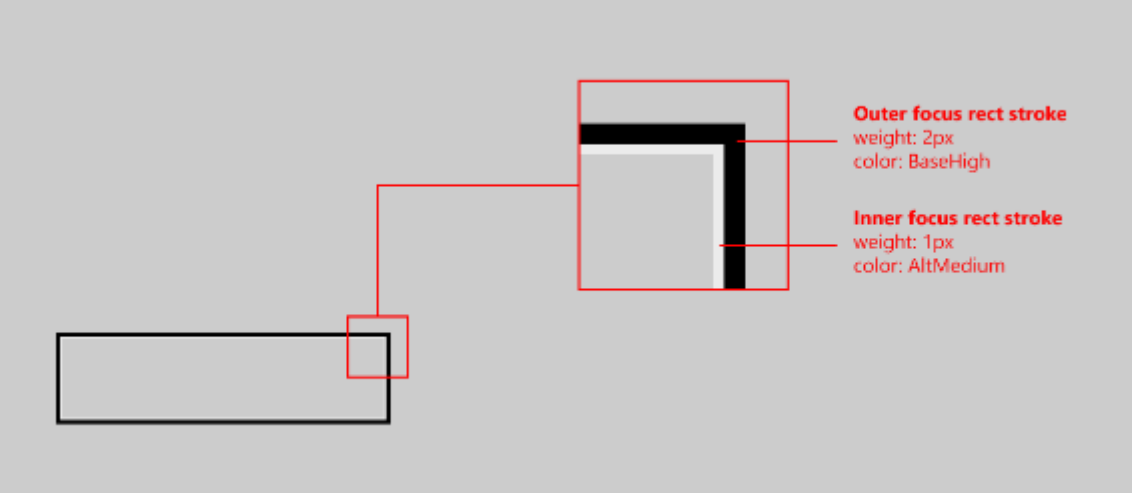
All Windows apps sport a more defined focus visual around interactable controls within the application. These new focus visuals are fully customizable as well as disableable when needed.

For the **10-foot experience** typical of Xbox and TV usage, Windows supports **Reveal focus**, a lighting effect that animates the border of focusable elements, such as a button, when they get focus through gamepad or keyboard input.

Color Branding & Customizing

Border Properties

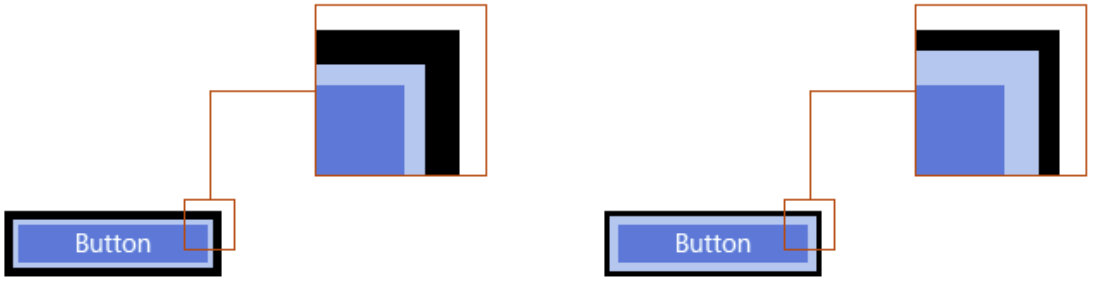
There are two parts to the high visibility focus visuals: the primary border and the secondary border. The primary border is **2px** thick, and runs around the *outside* of the secondary border. The secondary border is **1px** thick and runs around the *inside* of the primary border.



To change the thickness of either border type (primary or secondary) use the **FocusVisualPrimaryThickness** or **FocusVisualSecondaryThickness**, respectively:

XAML

```
<Slider Width="200" FocusVisualPrimaryThickness="5"
FocusVisualSecondaryThickness="2"/>
```



The margin is a property of type **Thickness**, and therefore the margin can be customized to appear only on certain sides of the control. See below:

The margin is the space between the control's visual bounds and the start of the focus visuals *secondary border*. The default margin is **1px** away from the control bounds. You can edit this margin on a per-control basis, by changing the **FocusVisualMargin** property:

```
XAML
<Slider Width="200" FocusVisualMargin="-5"/>
```

A negative margin will push the border away from the center of the control, and a positive margin will move the border closer to the center of the control.

To turn off focus visuals on the control entirely, simply disabled **UseSystemFocusVisuals**:

```
XAML
<Slider Width="200" UseSystemFocusVisuals="False"/>
```

The thickness, margin, or whether or not the app-developer wishes to have the focus visuals at all, is determined on a per-control basis.

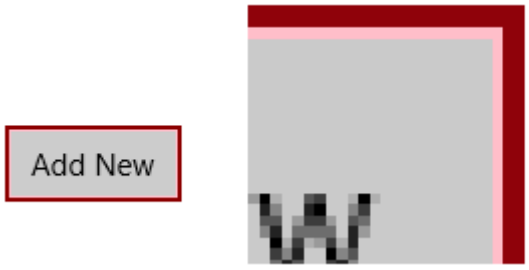
Color Properties

There are only two color properties for the focus visuals: the primary border color, and the secondary border color. These focus visual border colors can be changed per-control on a page level, and globally on an app-wide level:

To brand focus visuals app-wide, override the system brushes:

```
XAML
```

```
<SolidColorBrush x:Key="SystemControlFocusVisualPrimaryBrush"
Color="DarkRed"/>
<SolidColorBrush x:Key="SystemControlFocusVisualSecondaryBrush"
Color="Pink"/>
```



To change the colors on a per-control basis, just edit the focus visual properties on the desired control:

XAML

```
<Slider Width="200" FocusVisualPrimaryBrush="DarkRed"
FocusVisualSecondaryBrush="Pink"/>
```

Related articles

For designers

- Guidelines for panning

For developers

- Custom user interactions

Samples

- Basic input sample
- Low latency input sample
- User interaction mode sample
- Focus visuals sample

Archive samples

- Input: XAML user input events sample

- Input: Device capabilities sample
- Input: Touch hit testing sample
- XAML scrolling, panning, and zooming sample
- Input: Simplified ink sample
- Input: Windows 8 gestures sample
- Input: Manipulations and gestures sample
- DirectX touch input sample

Usability for Windows apps

Article • 05/13/2022

It's the little touches, an extra attention to detail, that can transform a good user experience into a truly inclusive user experience that meets the needs of users around the globe.

The design and coding instructions in this section can make your Windows app more inclusive by adding accessibility features, enabling globalization and localization, enabling users to customize their experience, and providing help when users need it.

Accessibility

Accessibility is about making your app usable by people who have limitations that prevent or impede the use of conventional user interfaces. For some situations, accessibility requirements are imposed by law. However, it's a good idea to address accessibility issues regardless of legal requirements so that your apps have the largest possible audience.

Accessibility portal

App settings

App settings let you the user customize your app, optimizing it for their individual needs and preferences. Providing the right settings and storing them properly can make a great user experience even better.

Guidelines

Best practices for creating and displaying app settings.

Store and retrieve app data

How to store and retrieve local, roaming, and temporary app data.

Globalization and localization

Windows is used worldwide by audiences that are diverse in terms of language, region, and culture. Your users speak a variety of different languages and in a variety of different

countries and regions. Some users speak more than one language. So, your app runs on configurations that involve many permutations of language, region, and culture system settings. Increase the potential market for your app by designing it to be readily adaptable, using *globalization* and *localization*.

Globalization and localization portal

In-app help

No matter how well you've designed your app, some users will need a little extra help.

Guidelines for app help

Applications can be complex, and providing effective help for your users can greatly improve their experience.

Instructional UI

Sometimes it can be helpful to teach the user about functions in your app that might not be obvious to them, such as specific touch interactions. In these cases, you need to present instructions to the user through the UI so they can discover and use features they might have missed.

In-app help

Most of the time, it's best for help to be displayed within the app, and to be displayed when the user chooses to view it. Consider the following guidelines when creating in-app help.

External help

Most of the time, it's best for help to be displayed within the app, and to be displayed when the user chooses to view it. Consider the following guidelines when creating in-app help.

Accessibility

Article • 05/13/2022

Accessibility is about building experiences that make your Windows application usable by people who use technology in a wide range of environments and approach your UI with a range of needs and experiences. For some situations, accessibility requirements are imposed by law. However, it's a good idea to address accessibility issues regardless of legal requirements so that your apps have the largest possible audience.

> There's also a Microsoft Store declaration regarding accessibility for your app!

Article	Description
Accessibility overview	This article is an overview of the concepts and technologies related to accessibility scenarios for Windows apps.
Designing inclusive software	Learn about evolving inclusive design with Windows apps for Windows. Design and build inclusive software with accessibility in mind.
Developing inclusive Windows apps	This article is a roadmap for developing accessible Windows apps.
Accessibility testing	Testing procedures to follow to ensure that your Windows app is accessible.
Accessibility in the Store	Describes the requirements for declaring your Windows app as accessible in the Microsoft Store.
Accessibility checklist	Provides a checklist to help you ensure that your Windows app is accessible.
Expose basic accessibility information	Basic accessibility info is often categorized into name, role, and value. This topic describes code to help your app expose the basic information that assistive technologies need.
Keyboard accessibility	If your app does not provide good keyboard access, users who are blind or have mobility issues can have difficulty using your app or may not be able to use it at all.
Screen readers and hardware system buttons	Screen-readers, such as Narrator, must be able to recognize and handle hardware system button events and communicate their state to users. In some cases, the screen reader might need to handle button events exclusively and not let them bubble up to other handlers.

Article	Description
Landmarks and Headings	Landmarks and headings define sections of a user interface that aid in efficient navigation for users of assistive technology such as screen readers.
High-contrast themes	Describes the steps needed to ensure your Windows app is usable when a high-contrast theme is active.
Accessible text requirements	This topic describes best practices for accessibility of text in an app, by assuring that colors and backgrounds satisfy the necessary contrast ratio. This topic also discusses the Microsoft UI Automation roles that text elements in a Windows app can have, and best practices for text in graphics.
Accessibility practices to avoid	Lists the practices to avoid if you want to create an accessible Windows app.
Custom automation peers	Describes the concept of automation peers for UI Automation, and how you can provide automation support for your own custom UI class.
Control patterns and interfaces	Lists the Microsoft UI Automation control patterns, the classes that clients use to access them, and the interfaces providers use to implement them.

Related topics

- **Windows.UI.Xaml.Automation**
- Get started with Narrator

Accessibility overview

Article • 03/31/2023

This article is an overview of the concepts and technologies related to accessibility scenarios for Windows apps.

Accessibility and your app

There are many possible disabilities or impairments, including limitations in mobility, vision, color perception, hearing, speech, cognition, and literacy. However, you can address most requirements by following the guidelines offered here. This means providing:

- Support for keyboard interactions and screen readers.
- Support for user customization, such as font, zoom setting (magnification), color, and high-contrast settings.
- Alternatives or supplements for parts of your UI.

Controls for XAML provide built-in keyboard support and support for assistive technologies such as screen readers, which take advantage of accessibility frameworks that already support UWP apps, HTML, and other UI technologies. This built-in support enables a basic level of accessibility that you can customize with very little work, by setting just a handful of properties. If you are creating your own custom XAML components and controls, you can also add similar support to those controls by using the concept of an *automation peer*.

In addition, data binding, style, and template features make it easy to implement support for dynamic changes to display settings and text for alternative UIs.

UI Automation

Accessibility support comes primarily from the integrated support for the Microsoft UI Automation framework. That support is provided through base classes and the built-in behavior of the class implementation for control types, and an interface representation of the UI Automation provider API. Each control class uses the UI Automation concepts of automation peers and automation patterns that report the control's role and content to UI Automation clients. The app is treated as a top-level window by UI Automation, and through the UI Automation framework all the accessibility-relevant content within that app window is available to a UI Automation client. For more info about UI Automation, see UI Automation Overview.

Assistive technology

Many user accessibility needs are met by assistive technology products installed by the user or by tools and settings provided by the operating system. This includes functionality such as screen readers, screen magnification, and high-contrast settings.

Assistive technology products include a wide variety of software and hardware. These products work through the standard keyboard interface and accessibility frameworks that report information about the content and structure of a UI to screen readers and other assistive technologies. Examples of assistive technology products include:

- The On-Screen Keyboard, which enables people to use a pointer in place of a keyboard to type text.
- Voice-recognition software, which converts spoken words into typed text.
- Screen readers, which convert text into spoken words or other forms such as Braille.
- The Narrator screen reader, which is specifically part of Windows. Narrator has a touch mode, which can perform screen reading tasks by processing touch gestures, for when there is no keyboard available.
- Programs or settings that adjust the display or areas of it, for example high contrast themes, dots per inch (dpi) settings of the display, or the Magnifier tool.

Apps that have good keyboard and screen reader support usually work well with various assistive technology products. In many cases, a UWP app works with these products without additional modification of information or structure. However, you may want to modify some settings for optimal accessibility experience or to implement additional support.

Some of the options that you can use for testing basic accessibility scenarios with assistive technologies are listed in Accessibility testing.

Screen reader support and basic accessibility information

Screen readers provide access to the text in an app by rendering it in some other format, such as spoken language or Braille output. The exact behavior of a screen reader depends on the software and on the user's configuration of it.

For example, some screen readers read the entire app UI when the user starts or switches to the app being viewed, which enables the user to receive all of the available informational content before attempting to navigate it. Some screen readers also read

the text associated with an individual control when it receives focus during tab navigation. This enables users to orient themselves as they navigate among the input controls of an application. Narrator is an example of a screen reader that provides both behaviors, depending on user choice.

The most important information that a screen reader or any other assistive technology needs in order to help users understand or navigate an app is an *accessible name* for the element parts of the app. In many cases, a control or element already has an accessible name that is calculated from other property values that you have otherwise provided. The most common case in which you can use an already-calculated name is with an element that supports and displays inner text. For other elements, you sometimes need to account for other ways to provide an accessible name by following best practices for element structure. And sometimes you need to provide a name that is explicitly intended as the accessible name for app accessibility. For a listing of how many of these calculated values work in common UI elements, and for more info about accessible names in general, see Basic accessibility information.

There are several other automation properties available (including the keyboard properties described in the next section). However, not all screen readers support all automation properties. In general, you should set all appropriate automation properties and test to provide the widest possible support for screen readers.

Keyboard support

To provide good keyboard support, you must ensure that every part of your application can be used with a keyboard. If your app uses mostly the standard controls and doesn't use any custom controls, you are most of the way there already. The basic XAML control model provides built-in keyboard support including tab navigation, text input, and control-specific support. The elements that serve as layout containers (such as panels) use the layout order to establish a default tab order. That order is often the correct tab order to use for an accessible representation of the UI. If you use **ListBox** and **GridView** controls to display data, they provide built-in arrow-key navigation. Or if you use a **Button** control, it already handles the Spacebar or Enter keys for button activation.

For more info about all the aspects of keyboard support, including tab order and key-based activation or navigation, see Keyboard accessibility.

Media and captioning

You typically display audiovisual media through a **MediaElement** object. You can use **MediaElement** APIs to control the media playback. For accessibility purposes, provide

controls that enable users to play, pause, and stop the media as needed. Sometimes, media includes additional components that are intended for accessibility, such as captioning or alternative audio tracks that include narrative descriptions.

Accessible text

Three main aspects of text are relevant to accessibility:

- Tools must determine whether the text is to be read as part of a tab-sequence traversal or only as part of an overall document representation. You can help control this determination by choosing the appropriate element for displaying the text or by adjusting properties of those text elements. Each text element has a specific purpose, and that purpose often has a corresponding UI Automation role. Using the wrong element can result in reporting the wrong role to UI Automation and creating a confusing experience for an assistive technology user.
- Many users have sight limitations that make it difficult for them to read text unless it has adequate contrast against the background. How this impacts the user is not intuitive for app designers who do not have that sight limitation. For example, for color-blind users, poor color choices in the design can prevent some users from being able to read the text. Accessibility recommendations that were originally made for web content define standards for contrast that can avoid these problems in apps as well. For more info, see Accessible text requirements.
- Many users have difficulty reading text that is simply too small. You can prevent this issue by making the text in your app's UI reasonably large in the first place. However, that's challenging for apps that display large quantities of text, or text interspersed with other visual elements. In such cases, make sure that the app correctly interacts with the system features that can scale up the display, so that any text in apps scales up along with it. (Some users change dpi values as an accessibility option. That option is available from **Make things on the screen larger** in **Ease of Access**, which redirects to a **Control Panel** UI for **Appearance and Personalization** / **Display**.)

Supporting high-contrast themes

UI controls use a visual representation that is defined as part of a XAML resource dictionary of themes. One or more of these themes is specifically used when the system is set for high contrast. When the user switches to high contrast, by looking up the appropriate theme from a resource dictionary dynamically, all your UI controls will use an appropriate high-contrast theme too. Just make sure that you haven't disabled the themes by specifying an explicit style or using another styling technique that prevents

the high-contrast themes from loading and overriding your style changes. For more info, see High-contrast themes.

Design for alternative UI

When you design your apps, consider how they may be used by people with limited mobility, vision, and hearing. Because assistive technology products make extensive use of standard UI, it is particularly important to provide good keyboard and screen-reader support even if you make no other adjustments for accessibility.

In many cases, you can convey essential information by using multiple techniques to widen your audience. For example, you can highlight information using both icon and color information to help users who are color blind, and you can display visual alerts along with sound effects to help users who are deaf or hard of hearing.

If necessary, you can provide alternative, accessible user interface elements that completely remove nonessential elements and animations, and provide other simplifications to streamline the user experience. The following code example demonstrates how to display one **UserControl** instance in place of another depending on a user setting.

XAML

XML

```
<StackPanel x:Name="LayoutRoot" Background="White">

  <CheckBox x:Name="ShowAccessibleUICheckBox"
Click="ShowAccessibleUICheckBox_Click">
    Show Accessible UI
  </CheckBox>

  <UserControl x:Name="ContentBlock">
    <local:ContentPage/>
  </UserControl>

</StackPanel>
```

Visual Basic

VB

```
Private Sub ShowAccessibleUICheckBox_Click(ByVal sender As Object,
    ByVal e As RoutedEventArgs)

    If (ShowAccessibleUICheckBox.IsChecked.Value) Then
```

```
        ContentBlock.Content = New AccessibleContentPage()
    Else
        ContentBlock.Content = New ContentPage()
    End If
End Sub
```

C#

```
C#

private void ShowAccessibleUICheckBox_Click(object sender, RoutedEventArgs
e)
{
    if ((sender as CheckBox).IsChecked.Value)
    {
        ContentBlock.Content = new AccessibleContentPage();
    }
    else
    {
        ContentBlock.Content = new ContentPage();
    }
}
```

Verification and publishing

For more info about accessibility declarations and publishing your app, see Accessibility in the Store.

> ⓘ **Note**
>
> Declaring the app as accessible is only relevant to the Microsoft Store.

Assistive technology support in custom controls

When you create a custom control, we recommend that you also implement or extend one or more **AutomationPeer** subclasses to provide accessibility support. In some cases, so long as you use the same peer class as was used by the base control class, the automation support for your derived class is adequate at a basic level. However, you should test this, and implementing a peer is still recommended as a best practice so that the peer can correctly report the class name of your new control class. Implementing a custom automation peer has a few steps involved. For more info, see Custom automation peers.

Assistive technology support in apps that support XAML / Microsoft DirectX interop

Microsoft DirectX content that's hosted in a XAML UI (using **SwapChainPanel** or **SurfaceImageSource**) is not accessible by default. The XAML SwapChainPanel DirectX interop sample shows how to create UI Automation peers for the hosted DirectX content. This technique makes the hosted content accessible through UI Automation.

Examples

> **Tip**

Open the WinUI 3 Gallery app and see the following Accessibility principles in action:

- **Screen reader support**
- **Keyboard support**
- **Color contrast**

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Related topics

- **Windows.UI.Xaml.Automation**
- Design for accessibility
- XAML accessibility sample
- Accessibility
- Get started with Narrator

Designing inclusive software for Windows

Article • 05/13/2022

Learn about evolving inclusive design with Windows apps for Windows. Design and build inclusive software with accessibility in mind.

At Microsoft, we're evolving our design principles and practices. These inform how our experiences look, feel, function, and behave. We're elevating our perspective.

This new design philosophy is called inclusive design. The idea is to design software with everyone in mind from the very beginning. This is in contrast to viewing accessibility as a technology you bolt on at the end of the development process in order to satisfy some small group of users.

"We define disability as a mismatch between the needs of the individual and the service, product or environment offered. Anyone can experience a disability. It is a common human trait to be excluded." - from the Inclusive video

Inclusive design creates better products for everyone. It's about considering the full range of human diversity. Consider the curb cutouts that you now find on most street corner sidewalks. They were clearly intended to be used by people in wheelchairs. But now nearly everyone uses them, including people with baby strollers, bicyclists, skateboarders. Even pedestrians will often use curb cutouts because they are there and provide a better experience. The television remote control could be considered an assistive technology (AT) for someone with physical limitations. And yet, today it is nearly impossible to buy a television without one. Before children learn to tie their shoes, they can wear slip-on or easy fastening shoes. Shoes that are easy to put on and take off are often preferred in cultures where shoes are removed before entering a home. They are also better for people with dexterity issues such as arthritis or even a temporarily broken wrist.

Inclusive design principles

The following 4 principles are guiding Microsoft's shift to inclusive design:

Think universal: We focus on what unifies people — human motivations, relationships, and abilities. This drives us to consider the broader social impact of our work. The result is an experience that has a diversity of ways for all people to participate.

Make it personal: Next, we challenge ourselves to create emotional connections. Human-to-human interactions can inspire better human-to-technology interaction. A person's unique circumstances can improve a design for everyone. The result is an experience that feels like it was created for one person.

Keep it simple: We start with simplicity as the ultimate unifier. When we reduce clutter people know what to do next. They're inspired to move forward into spaces that are clean, light, and open. The result is an experience that's honest and timeless.

Create delight: Delightful experiences evoke wonder and discovery. Sometimes it's magical. Sometimes it's a detail that's just right. We design these moments to feel like a welcomed change in tempo. The result is an experience that has momentum and flow.

Inclusive design users

There are essentially two types of users of assistive technology (AT):

1. Those who need it, because of disabilities or impairments, age-related conditions, or temporary conditions (such as limited mobility from a broken limb)
2. Those who use it out of preference, for a more comfortable or convenient computing experience

The majority of computer users (54 per-cent) are aware of some form of assistive technology, and 44 percent of computer users use some form of it, but many of them are not using AT that would benefit them (Forrester 2004).

A 2003-2004 study commissioned by Microsoft and conducted by Forrester Research found that over half — 57 percent — of computer users in the United States between the ages of 18 and 64 could benefit from assistive technology. Most of these users did not identify themselves as having a disability or being impaired but expressed certain task-related difficulties or impairments when using a computer. Forrester (2003) also found the following number of users with these specific difficulties: One in four experiences a visual difficulty. One in four experiences pain in the wrists or hands. One in five experiences hearing difficulty.

Besides permanent disabilities, the severity and types of difficulties an individual experiences can vary throughout their life. There is no such thing as a normal human. Our capabilities are always changing. Margaret Meade said, "We are all unique. Being all unique makes us all the same."

Microsoft is dedicated to conducting computer science and software engineering research with goals to enhance the computing experience and invent novel computing

technologies. See Current Microsoft Research and Development Projects aimed at making the computer more accessible, and easier to see, hear, and interact with.

Practical design steps

If you're all in, then this section is for you. It describes the practical design steps to consider when implementing inclusive design for your app.

Describe the target audience

Define the potential users of your app. Think through all of their different abilities and characteristics. For example, age, gender, language, deaf or hard of hearing users, visual impairments, cognitive abilities, learning style, mobility restrictions, and so on. Is your design meeting their individual needs?

Talk to actual humans with specific needs

Meet with potential users who have diverse characteristics. Make sure you are considering all of their needs when designing your app. For example, Microsoft discovered that deaf users were turning off toast notifications on their Xbox consoles. When we asked actual deaf users about this, we learned that toast notifications were obscuring a section of closed captioning. The fix was to display the toast slight higher on the screen. This was a simple solution that was not necessarily obvious from the telemetry data that initially revealed the behavior.

Choose a development framework wisely

In the design stage, the development framework you will use (i.e. UWP, Win32, web) is critical to the development of your product. If you have the luxury of choosing your framework, think about how much effort it will take to create your controls within the framework. What are the default or built-in accessibility properties that come with it? Which controls will you need to customize? When choosing your framework, you are essentially choosing how much of the accessibility controls you will get “for free” (that is, how much of the controls are already built-in) and how much will require additional development costs because of control customizations.

Use standard Windows controls whenever possible. These controls are already enabled with the technology necessary to interface with assistive technologies.

Design a logical hierarchy for your controls

Once you have your framework, design a logical hierarchy to map out your controls. The logical hierarchy of your app includes the layout and tab order of controls. When assistive technology (AT) programs, such as screen readers, read your UI, visual presentation is not sufficient; you must provide a programmatic alternative that makes sense structurally to your users. A logical hierarchy can help you do that. It is a way of studying the layout of your UI and structuring each element so that users can understand it. A logical hierarchy is mainly used:

1. To provide programs context for the logical (reading) order of the elements in the UI
2. To identify clear boundaries between custom controls and standard controls in the UI
3. To determine how pieces of the UI interact together

A logical hierarchy is a great way to address any potential usability issues. If you cannot structure the UI in a relatively simple manner, you may have problems with usability. A logical representation of a simple dialog box should not result in pages of diagrams. For logical hierarchies that become too deep or too wide, you may need to redesign your UI. For more information, download the Engineering Software for Accessibility eBook.

Design appropriate visual UI settings

When designing the visual UI, ensure that your product has a high contrast setting, uses the default system fonts and smoothing options, correctly scales to the dots per inch (dpi) screen settings, has default text with at least a 5:1 contrast ratio with the background, and has color combinations that will be easy for users with color deficiencies to differentiate.

High contrast setting

One of the built-in accessibility features in Windows is High Contrast mode, which heightens the color contrast of text and images. For some people, increasing the contrast in colors reduces eyestrain and makes it easier to read. When you verify your UI in high contrast mode, you want to check that controls, such as links, have been coded consistently and with system colors (not with hard-coded colors) to ensure that they will be able to see all the controls on the screen that a user not using high contrast would see.

System font settings

To ensure readability and minimize any unexpected distortions to the text, make sure that your product always adheres to the default system fonts and uses the anti-aliasing and smoothing options. If your product uses custom fonts, users may face significant readability issues and distractions when they customize the presentation of their UI (through the use of a screen reader or by using different font styles to view your UI, for instance).

High DPI resolutions

For users with vision impairments, having a scalable UI is important. User interfaces that do not scale correctly in high dots-per-inch (DPI) resolutions may cause important components to overlap or hide other components and can become inaccessible.

Color contrast ratio

The updated Section 508 of the Americans with Disability Act (ADA), as well as other legislations, requires that the default color contrasts between text and its background must be 5:1. For large texts (18-point font sizes, or 14 points and bolded) the required default contrast is 3:1.

Color combinations

About 7 percent of males (and less than 1 percent of females) have some form of color deficiency. Users with colorblindness have problems distinguishing between certain colors, so it is important that color alone is never used to convey status or meaning in an application. As for decorative images (such as icons or backgrounds), color combinations should be chosen in a manner that maximizes the perception of the image by colorblind users. If you design using these color recommendations from the beginning, your app will already be taking significant steps toward being inclusive.

Summary — seven steps for inclusive design

In summary, follow these seven steps to ensure your software is inclusive.

1. Decide if inclusive design is an important aspect to your software. If it is, learn and appreciate how it enables real users to live, work, and play, to help guide your design.
2. As you design solutions for your requirements, use controls provided by your framework (standard controls) as much as possible, and avoid any unnecessary effort and costs of custom controls.

3. Design a logical hierarchy for your product, noting where the standard controls, any custom controls, and keyboard focus are in the UI.
4. Design useful system settings (such as keyboard navigation, high contrast, and high dpi) into your product.
5. Implement your design, using the Microsoft accessibility developer hub and your framework's accessibility specification as a reference point.
6. Test your product with users who have functional needs to ensure they will be able to take advantage of the inclusive design techniques implemented in it.
7. Deliver your finished product and document your implementation for those who may work on the project after you.

Related topics

- Inclusive design
- Engineering Software for Accessibility
- Microsoft accessibility developer hub
- Developing inclusive Windows apps
- Accessibility

Developing inclusive Windows apps

Article • 05/13/2022

This article discusses how to develop accessible Windows apps. Specifically, it assumes that you understand how to design the logical hierarchy for your app. Learn to develop accessible Windows apps that include keyboard navigation, color and contrast settings, and support for assistive technologies.

If you have not yet done so, please start by reading Designing inclusive software.

There are three things you should do to make sure that your app is accessible:

1. Expose your UI elements to programmatic access.
2. Ensure that your app supports keyboard navigation for people who are unable to use a mouse or touchscreen.
3. Make sure that your app supports accessible color and contrast settings.

Programmatic access

Programmatic access is critical for creating accessibility in apps. This is achieved by setting the accessible name (required) and description (optional) for content and interactive UI elements in your app. This ensures that UI controls are exposed to assistive technology (AT) such as screen readers (for example, Narrator) or alternative output devices (such as Braille displays). Without programmatic access, the APIs for assistive technology cannot interpret information correctly, leaving the user unable to use the products sufficiently, or forcing the AT to use undocumented programming interfaces or techniques never intended to be used as an accessibility interface. When UI controls are exposed to assistive technology, the AT is able to determine what actions and options are available to the user.

For more information about making your app UI elements available to assistive technologies (AT), see Expose basic accessibility information.

Keyboard navigation

For users who are blind or have mobility issues, being able to navigate the UI with a keyboard is extremely important. However, only those UI controls that require user interaction to function should be given keyboard focus. Components that don't require an action, such as static images, do not need keyboard focus.

It is important to remember that unlike navigating with a mouse or touch, keyboard navigation is linear. When considering keyboard navigation, think about how your user will interact with your product and what the logical navigation will be. In Western cultures, people read from left to right, top to bottom. It is, therefore, common practice to follow this pattern for keyboard navigation.

When designing keyboard navigation, examine your UI, and think about these questions:

- How are the controls laid out or grouped in the UI?
- Are there a few significant groups of controls?
  - If yes, do those groups contain another level of groups?
- Among peer controls, should navigation be done by tabbing around, or via special navigation (such as arrow keys), or both?

The goal is to help the user understand how the UI is laid out and identify the controls that are actionable. If you are finding that there are too many tab stops before the user completes the navigation loop, consider grouping related controls together. Some controls that are related, such as a hybrid control, may need to be addressed at this early exploration stage. After you begin to develop your product, it is difficult to rework the keyboard navigation, so plan carefully and plan early!

To learn more about keyboard navigation among UI elements, see Keyboard accessibility.

Also, the Engineering Software for Accessibility eBook has an excellent chapter on this subject titled *Designing the Logical Hierarchy*.

Color and contrast

One of the built-in accessibility features in Windows is the High Contrast mode, which heightens the color contrast of text and images on the computer screen. For some people, increasing the contrast in colors reduces eyestrain and makes it easier to read. When you verify your UI in high contrast, you want to check that controls have been coded consistently and with system colors (not with hard-coded colors) to ensure that they will be able to see all the controls on the screen that a user not using high contrast would see.

XAML

```
XAML
```

```
<Button Background="{ThemeResource ButtonBackgroundThemeBrush}">OK</Button>
```

For more information about using system colors and resources, see XAML theme resources.

As long as you haven't overridden system colors, a UWP app supports high-contrast themes by default. If a user has chosen that they want the system to use a high-contrast theme from system settings or accessibility tools, the framework automatically uses colors and style settings that produce a high-contrast layout and rendering for controls and components in the UI.

For more information, see High-contrast themes.

If you have decided to use you own color theme instead of system colors, consider these guidelines:

Color contrast ratio – The updated Section 508 of the Americans with Disability Act, as well as other legislation, requires that the default color contrasts between text and its background must be 5:1. For large text (18-point font sizes, or 14 points and bolded), the required default contrast is 3:1.

Color combinations – About 7 percent of males (and less than 1 percent of females) have some form of color deficiency. Users with colorblindness have problems distinguishing between certain colors, so it is important that color alone is never used to convey status or meaning in an application. As for decorative images (such as icons or backgrounds), color combinations should be chosen in a manner that maximizes the perception of the image by colorblind users.

Accessibility checklist

Following is an abbreviated version of the accessibility checklist:

1. Set the accessible name (required) and description (optional) for content and interactive UI elements in your app.
2. Implement keyboard accessibility.
3. Visually verify your UI to ensure that the text contrast is adequate, elements render correctly in the high-contrast themes, and colors are used correctly.
4. Run accessibility tools, address reported issues, and verify the screen reading experience. (See Accessibility testing topic.)
5. Make sure your app manifest settings follow accessibility guidelines.
6. Declare your app as accessible in the Microsoft Store. (See the Accessibility in the store topic.)

For more detail, please see the full Accessibility checklist topic.

Related topics

- Designing inclusive software
- Inclusive design
- Accessibility practices to avoid
- Engineering Software for Accessibility
- Microsoft accessibility developer hub
- Accessibility

Accessibility testing

Article • 08/13/2022

This topic describes various tools and procedures to help you verify the accessibility implementation of your Windows and web applications.

Successful user experiences

Programmatic access and keyboard access are critical requirements for supporting accessibility in your application. Testing the accessibility of your Windows applications, assistive technology (AT) tools, and UI frameworks is crucial to ensure a successful user experience for people with various disabilities and limitations (including vision, learning, dexterity/mobility, and language/communication disabilities), or those who simply prefer using a keyboard.

Without adequate support for accessible technology (AT) such as screen-readers and on-screen keyboards, users with vision, learning, dexterity/mobility, and language/communication disabilities or limitations (and users who just prefer using the keyboard) could find it difficult, if not impossible, to use your application.

Accessibility testing tools

Accessibility Insights

Accessibility Insights helps developers find and fix accessibility issues in both websites and Windows applications.
https://www.youtube-nocookie.com/embed/XlvI91Q3c_8

- Accessibility Insights for Windows helps developers find and fix accessibility issues in Windows apps. The tool supports three primary scenarios:
  - **Live Inspect** lets developers verify that an element in an app has the right UI Automation properties simply by hovering over the element or setting keyboard focus on it.
  - **FastPass** - a lightweight, two-step process that helps developers identify common, high-impact accessibility issues in less than five minutes.
  - **Troubleshooting** allows you to diagnose and fix specific accessibility issues.
- Accessibility Insights for Web is an extension for Chrome and Microsoft Edge Insider that helps developers find and fix accessibility issues in web apps and sites. It supports two primary scenarios:

- **FastPass** - a lightweight, two-step process that helps developers identify common, high-impact accessibility issues in less than five minutes.
- **Assessment** - lets anyone verify that a web site is 100% compliant with accessibility standards and guidelines. Accessibility Insights also lets you review UI Automation elements, properties, control patterns, and events (similar to the Inspect and AccEvent legacy tools described in the following section).

Legacy testing tools

> **Note**
>
> The tools described here are still available in the Windows SDK, but we strongly recommend transitioning to **Accessibility Insights**.

The Windows Software Development Kit (SDK) includes several accessibility testing tools, including **AccScope**, **Inspect** and **UI Accessibility Checker**, among others.

You can launch the following accessibility testing tools either from a Microsoft Visual Studio command prompt or by navigating to the *bin* folder of wherever the Windows SDK is installed on your development machine.

AccScope

The **AccScope** Enables visual evaluation of an application's accessibility during the early design and development phases. AccScope is specifically intended for testing Narrator accessibility scenarios and uses the UI Automation information provided by an app to show where accessibility can be improved.

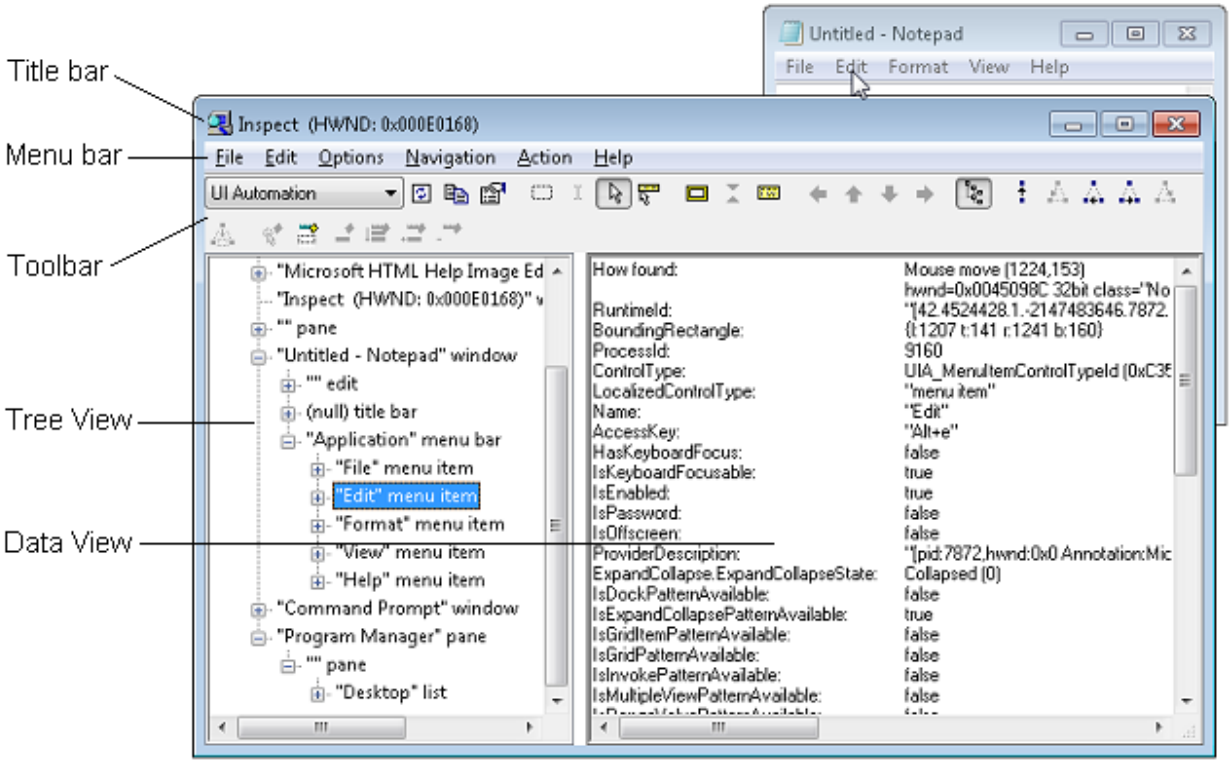
Inspect

Inspect enables you to select any UI element and view its accessibility data. You can view Microsoft UI Automation properties and control patterns and test the navigational structure of the automation elements in the UI Automation tree. It is especially useful for ensuring properties and control patterns are set correctly when extending a common control or creating a custom control.

Use **Inspect** as you develop the UI to verify how accessibility attributes are exposed in UI Automation. In some cases the attributes come from the UI Automation support that is already implemented for default XAML controls. In other cases the attributes come from

specific values that you have set in your XAML markup, as **AutomationProperties** attached properties.

The following image shows the **Inspect** tool querying the UI Automation properties of the **Edit** menu element in Notepad.



UI Accessibility Checker

UI Accessibility Checker (AccChecker) helps you discover potential accessibility issues at run time. AccChecker includes verification checks for UI Automation, Microsoft Active Accessibility, and Accessible Rich Internet Applications (ARIA). It can provide a static check for errors such as missing names, tree issues and more. It helps verify programmatic access and includes advanced features for automating accessibility testing. You can run **AccChecker** in UI or command line mode. To run the UI mode tool, open the *AccChecker* folder in the Windows SDK *bin* folder, run accccheckui.exe, and click the **Help** menu.

UI Automation Verify

UI Automation Verify (UIA Verify) is a framework for manual and automated testing of the UI Automation implementation in a control or application (results can be logged). **UIA Verify** can integrate into the test code and conduct regular, automated testing or spot checks of UI Automation scenarios and is useful for verifying that changes to applications with established features do not have new issues or regressions. **UIA Verify** can be found in the *UIAVerify* subfolder of the Windows SDK *bin* folder.

Accessible Event Watcher

Accessible Event Watcher (AccEvent) tests whether an app's UI elements fire proper UI Automation and Microsoft Active Accessibility events when UI changes occur. Changes in the UI can occur when the focus changes, or when a UI element is invoked, selected, or has a state or property change. AccEvent is typically used to debug issues and to validate that custom and extended controls are working correctly.

Accessibility testing procedures

Test keyboard accessibility

The best way to test your keyboard accessibility is to unplug your mouse or use the On-Screen Keyboard if you are using a tablet device. Test keyboard accessibility navigation by using the *Tab* key. You should be able to cycle through all interactive UI elements by using *Tab* key. For composite UI elements, verify that you can navigate among the parts of elements by using the arrow keys. For example, you should be able to navigate lists of items using keyboard keys. Finally, make sure that you can invoke all interactive UI elements with the keyboard once those elements have focus, typically by using the Enter or Spacebar key.

Verify the contrast ratio of visible text

Use color contrast tools to verify that the visible text contrast ratio is acceptable. The exceptions include inactive UI elements, and logos or decorative text that doesn't convey any information and can be rearranged without changing the meaning. See Accessible text requirements for more information on contrast ratio and exceptions. See Techniques for WCAG 2.0 G18 (Resources section) for tools that can test contrast ratios.

> **Note**
>
> Some of the tools listed by Techniques for WCAG 2.0 G18 can't be used interactively with a UWP app. You may need to enter foreground and background color values manually in the tool, make screen captures of app UI and then run the contrast ratio tool over the screen capture image, or run the tool while opening source bitmap files in an image editing program rather than while that image is loaded by the app.

Verify your app in high contrast

Use your app while a high-contrast theme is active to verify that all the UI elements display correctly. All text should be readable, and all images should be clear. Adjust the XAML theme-dictionary resources or control templates to correct any theme issues that come from controls. In cases where prominent high-contrast issues are not coming from themes or controls (such as from image files), provide separate versions to use when a high-contrast theme is active.

Verify your app with display settings

Use the system display options that adjust the display's dots per inch (dpi) value, and ensure that your app UI scales correctly when the dpi value changes. (Some users change dpi values as an accessibility option, it's available from **Ease of Access** as well as display properties.) If you find any issues, follow the Guidelines for layout scaling and provide additional resources for different scaling factors.

Verify main app scenarios by using Narrator

Use Narrator to test the screen reading experience for your app.

Use these steps to test your app using Narrator with a mouse and keyboard:

1. Start Narrator by pressing *Windows logo key* + *Ctrl* + *Enter*. In versions prior to Windows 10 version 1607, use *Windows logo key* + *Enter* to start Narrator.
2. Navigate your app with the keyboard by using the *Tab* key, the arrow keys, and the *Caps Lock* + *arrow keys*.
3. As you navigate your app, listen as Narrator reads the elements of your UI and verify the following:

    - For each control, ensure that Narrator reads all visible content. Also ensure that Narrator reads each control's name, any applicable state (checked, selected, and so on), and the control type (button, check box, list item, and so on).
    - If the element is interactive, verify that you can use Narrator to invoke its action by pressing *Caps Lock* + *Enter*.
    - For each table, ensure that Narrator correctly reads the table name, the table description (if available), and the row and column headings.

4. Press *Caps Lock* + *Shift* + *Enter* to search your app and verify that all of your controls appear in the search list, and that the control names are localized and readable.

5. Turn off your monitor and try to accomplish main app scenarios by using only the keyboard and Narrator. To get the full list of Narrator commands and shortcuts, press *Caps Lock + F1*.

Starting with Windows 10 version 1607, we introduced a new developer mode in Narrator. Turn on developer mode when Narrator is already running by pressing *Control + Caps Lock + F12*. When developer mode is enabled, the screen will be masked and will highlight only the accessible objects and the associated text that is exposed programmatically to Narrator. This gives a you a good visual representation of the information that is exposed to Narrator.

Use these steps to test your app using Narrator's touch mode:

> ⓘ **Note**
>
> Narrator automatically enters touch mode on devices that support 4+ contacts. Narrator doesn't support multi-monitor scenarios or multi-touch digitizers on the primary screen.

1. Get familiar with the UI and explore the layout.

    - **Navigate through the UI by using single-finger swipe gestures.** Use left or right swipes to move between items, and up or down swipes to change the category of items being navigated. Categories include all items, links, tables, headers, and so on. Navigating with single-finger swipe gestures is similar to navigating with *Caps Lock + Arrow*.
    - **Use tab gestures to navigate through focusable elements.** A three-finger swipe to the right or left is the same as navigating with *Tab* and *Shift + Tab* on a keyboard.
    - **Spatially investigate the UI with a single finger.** Drag a single finger up and down, or left and right, to have Narrator read the items under your finger. You can use the mouse as an alternative because it uses the same hit-testing logic as dragging a single finger.
    - **Read the entire window and all its contents with a three finger swipe up**. This is equivalent to using *Caps Lock + W*.

    If there is important UI that you cannot reach, you may have an accessibility issue.

2. Interact with a control to test its primary and secondary actions, and its scrolling behavior.

    Primary actions include things like activating a button, placing a text caret, and setting focus to the control. Secondary actions include actions such as selecting a

list item or expanding a button that offers multiple options.

- To test a primary action: Double tap, or press with one finger and tap with another.
- To test a secondary action: Triple tap, or press with one finger and double tap with another.
- To test scrolling behavior: Use two-finger swipes to scroll in the desired direction.

Some controls provide additional actions. To display the full list, enter a single four-finger tap.

If a control responds to the mouse or keyboard but does not respond to a primary or secondary touch interaction, the control might need to implement additional UI Automation control patterns.

You should also consider using the **AccScope** tool to test Narrator accessibility scenarios with your app. The **AccScope tool topic** describes how to configure **AccScope** to test Narrator scenarios.

Examine the UI Automation representation for your app

Several of the UI Automation testing tools mentioned previously provide a way to view your app in a way that deliberately does not consider what the app looks like, and instead represents the app as a structure of UI Automation elements. This is how UI Automation clients, mainly assistive technologies, will be interacting with your app in accessibility scenarios.

The **AccScope** tool provides a particularly interesting view of your app because you can see the UI Automation elements either as a visual representation or as a list. If you use the visualization, you can drill down into the parts in a way that you'll be able to correlate with the visual appearance of your app's UI. You can even test the accessibility of your earliest UI prototypes before you've assigned all the logic to the UI, making sure that both the visual interaction and accessibility-scenario navigation for your app is in balance.

One aspect that you can test is whether there are elements appearing in the UI Automation element view that you don't want to appear there. If you find elements you want to omit from the view, or conversely if there are elements missing, you can use the **AutomationProperties.AccessibilityView** XAML attached property to adjust how XAML controls appear in accessibility views. After you've looked at the basic accessibility views, this is also a good opportunity to recheck your tab sequences or spatial navigation as

enabled by arrow keys to make sure users can reach each of the parts that are interactive and exposed in the control view.

Related topics

- Accessibility
- Practices to avoid
- UI Automation
- Accessibility in Windows
- Get started with Narrator

Accessibility in the Store

Article • 01/04/2023

Describes the requirements for declaring your Windows app as accessible in the Microsoft Store.

While submitting your app to the Microsoft Store for certification, you can declare your app as accessible. Declaring your app as accessible makes it easier to discover for users who are interested in accessible apps, such as users who have visual impairments. Users discover accessible apps by using the **Accessible** filter while searching the Microsoft Store. Declaring your app as accessible also adds the **Accessible** tag to your app's description.

By declaring your app as accessible, you state that it has the basic accessibility information that users need for primary scenarios using one or more of the following:

- The keyboard.
- A high contrast theme.
- A variable dots per inch (dpi) setting.
- Common assistive technology such as the Windows accessibility features, including Narrator, Magnifier, and On-Screen Keyboard.

You should declare your app as accessible if you built and tested it for accessibility. This means that you did the following:

- Set all the relevant accessibility information for UI elements, including name, role, value, and so on.
- Implemented full keyboard accessibility, enabling the user to:
  - Accomplish primary app scenarios by using only the keyboard.
  - Tab among UI elements in a logical order.
  - Navigate among UI elements within a control by using the arrow keys.
  - Use keyboard shortcuts to reach primary app functionality.
  - Use Narrator touch gestures for Tab and arrow equivalency for devices with no keyboard.
- Ensured that your app UI is visually accessible: has a minimum text contrast ratio of 4.5:1, does not rely on color alone to convey information, and so on.
- Used accessibility testing tools such as **Inspect** and **UIAVerify** to verify your accessibility implementation, and resolved all priority 1 errors reported by such tools.
- Verified your app's primary scenarios from end to end by using Narrator, Magnifier, On-Screen Keyboard, a high contrast theme, and adjusted dpi settings.

See the Accessibility checklist for a review of these procedures and links to resources that will help you accomplish them.

Related topics

- Accessibility

Accessibility checklist

Article • 06/24/2021

Provides a checklist to help you ensure that your Windows app is accessible .

Here we provide a checklist you can use to ensure that your app is accessible.

1. Set the accessible name (required) and description (optional) for content and interactive UI elements in your app.

   An accessible name is a short, descriptive text string that a screen reader uses to announce a UI element. Some UI elements such as **TextBlock** and **TextBox** promote their text content as the default accessible name; see Basic accessibility information.

   You should set the accessible name explicitly for images or other controls that do not promote inner text content as an implicit accessible name. You should use labels for form elements so that the label text can be used as a **LabeledBy** target in the Microsoft UI Automation model for correlating labels and inputs. If you want to provide more UI guidance for users than is typically included in the accessible name, accessible descriptions and tooltips help users understand the UI.

   For more info, see Accessible name and Accessible description.

2. Implement keyboard accessibility:

   - Test the default tab index order for a UI. Adjust the tab index order if necessary, which may require enabling or disabling certain controls, or changing the default values of **TabIndex** on some of the UI elements.
   - Use controls that support arrow-key navigation for composite elements. For default controls, the arrow-key navigation is typically already implemented.
   - Use controls that support keyboard activation. For default controls, particularly those that support the UI Automation **Invoke** pattern, keyboard activation is typically available; check the documentation for that control.
   - Set access keys or implement accelerator keys for specific parts of the UI that support interaction.
   - For any custom controls that you use in your UI, verify that you have implemented these controls with correct **AutomationPeer** support for activation, and defined overrides for key handling as needed to support activation, traversal and access or accelerator keys.

   For more info, see Keyboard interactions.

3. Ensure text is a readable size

    - Windows includes various accessibility tools and settings that users can take advantage of and adjust to their own needs and preferences for reading text. These include:
        - The Magnifier tool, which enlarges a selected area of the UI. You should ensure the layout of text in your app doesn't make it difficult to use Magnifier for reading.
        - Global scale and resolution settings in **Settings->System->Display->Scale and layout**. Exactly which sizing options are available can vary as this depends on the capabilities of the display device.
        - Text size settings in **Settings->Ease of access->Display**. Adjust the **Make text bigger** setting to specify only the size of text in supporting controls across all applications and screens (all UWP text controls support the text scaling experience without any customization or templating).

    > ⓘ **Note**
    >
    > The **Make everything bigger** setting lets a user specify their preferred size for text and apps in general on their primary screen only.

4. Visually verify your UI to ensure that the text contrast is adequate, elements render correctly in the high-contrast themes, and colors are used correctly.

    - Use a color analyzer tool to verify that the visual text contrast ratio is at least 4.5:1.
    - Switch to a high contrast theme and verify that the UI for your app is readable and usable.
    - Ensure that your UI doesn't use color as the only way to convey information.

    For more info, see High-contrast themes and Accessible text requirements.

5. Run accessibility tools, address reported issues, and verify the screen reading experience.

    Use tools such as **Inspect** to verify programmatic access, run diagnostic tools such as **AccChecker** to discover common errors, and verify the screen reading experience with Narrator.

    For more info, see Accessibility testing.

6. Make sure your app manifest settings follow accessibility guidelines.

7. Declare your app as accessible in the Microsoft Store.

   If you implemented the baseline accessibility support, declaring your app as accessible in the Microsoft Store can help reach more customers and get some additional good ratings.

   For more info, see Accessibility in the Store.

Related topics

- Accessible text requirements
- Text scaling
- Accessibility
- Design for accessibility
- Practices to avoid

Expose basic accessibility information

Article • 01/04/2023

Basic accessibility info is often categorized into name, role, and value. This topic describes code to help your app expose the basic information that assistive technologies need.

Accessible name

An accessible name is a short, descriptive text string that a screen reader uses to announce a UI element. Set the accessible name for UI elements so that have a meaning that is important for understanding the content or interacting with the UI. Such elements typically include images, input fields, buttons, controls, and regions.

This table describes how to define or obtain an accessible name for various types of elements in a XAML UI.

Element type	Description
Static text	For **TextBlock** and **RichTextBlock** elements, an accessible name is automatically determined from the visible (inner) text. All of the text in that element is used as the name. See Name from inner text.
Images	The XAML **Image** element does not have a direct analog to the HTML **alt** attribute of **img** and similar elements. Either use **AutomationProperties.Name** to provide a name, or use the captioning technique. See Accessible names for images.
Form elements	The accessible name for a form element should be the same as the label that is displayed for that element. See Labels and LabeledBy.
Buttons and links	By default, the accessible name of a button or link is based on the visible text, using the same rules as described in Name from inner text. In cases where a button contains only an image, use **AutomationProperties.Name** to provide a text-only equivalent of the button's intended action.

Most container elements such as panels do not promote their content as accessible name. This is because it is the item content that should report a name and corresponding role, not its container. The container element might report that it is an element that has children in a Microsoft UI Automation representation, such that the assistive technology logic can traverse it. But users of assistive technologies don't generally need to know about the containers and thus most containers aren't named.

Role and value

The controls and other UI elements that are part of the XAML vocabulary implement UI Automation support for reporting role and value as part of their definitions. You can use UI Automation tools to examine the role and value information for the controls, or you can read the documentation for the AutomationPeer implementations of each control. The available roles in a UI Automation framework are defined in the AutomationControlType enumeration. UI Automation clients such as assistive technologies can obtain role information by calling methods that the UI Automation framework exposes by using the control's **AutomationPeer**.

Not all controls have a value. Controls that do have a value report this information to UI Automation through the peers and patterns that are supported by that control. For example, a TextBox form element does have a value. An assistive technology can be a UI Automation client and can discover both that a value exists and what the value is. In this specific case the **TextBox** supports the IValueProvider pattern through the TextBoxAutomationPeer definitions.

> ⓘ Note
>
> For cases where you use **AutomationProperties.Name** or other techniques to supply the accessible name explicitly, do not include the same text as is used by the control role or type information in the accessible name. For example do not include strings such as "button" or "list" in the name. The role and type information comes from a different UI Automation property (**LocalizedControlType**) that is supplied by the default control support for UI Automation. Many assistive technologies append the **LocalizedControlType** to the accessible name, so duplicating the role in the accessible name can result in unnecessarily repeated words. For example, if you give a **Button** control an accessible name of "button" or include "button" as the last part of the name, this might be read by screen readers as "button button". You should test this aspect of your accessibility info using Narrator.

Influencing the UI Automation tree views

The UI Automation framework has a concept of tree views, where UI Automation clients can retrieve the relationships between elements in a UI using three possible views: raw, control, and content. The control view is the view that's often used by UI Automation clients because it provides a good representation and organization of the elements in a UI that are interactive. Testing tools usually enable you to choose which tree view to use when the tool presents the organization of elements.

By default, any Control derived class and a few other elements will appear in the control view when the UI Automation framework represents the UI for a Windows app. But sometimes you don't want an element to appear in the control view because of UI composition, where that element is duplicating information or presenting information that's unimportant for accessibility scenarios. Use the attached property AutomationProperties.AccessibilityView to change how elements are exposed to the tree views. If you put an element in the **Raw** tree, most assistive technologies won't report that element as part of their views. To see some examples of how this works in existing controls, open the generic.xaml design reference XAML file in a text editor, and search for **AutomationProperties.AccessibilityView** in the templates.

Name from inner text

To make it easier to use strings that already exist in the visible UI for accessible name values, many of the controls and other UI elements provide support for automatically determining a default accessible name based on inner text within the element, or from string values of content properties.

- TextBlock, RichTextBlock, TextBox and **RichTextBlock** each promote the value of the **Text** property as the default accessible name.
- Any ContentControl subclass uses an iterative "ToString" technique to find strings in its Content value, and promotes these strings as the default accessible name.

> ⓘ **Note**
>
> As enforced by UI Automation, the accessible name length cannot be greater than 2048 characters. If a string used for automatic accessible name determination exceeds that limit, the accessible name is truncated at that point.

Accessible names for images

To support screen readers and to provide the basic identifying information for each element in the UI, you sometimes must provide text alternatives to non-textual information such as images and charts (excluding any purely decorative or structural elements). These elements don't have inner text so the accessible name won't have a calculated value. You can set the accessible name directly by setting the AutomationProperties.Name attached property as shown in this example.

XAML

XML

```
<!-- Comment -->
<Image Source="product.png"
  AutomationProperties.Name="An image of a customer using the product."/>
```

Alternatively, consider including a text caption that appears in the visible UI and that also serves as the label-associated accessibility information for the image content. Here's an example:

XAML

XML

```
<Image HorizontalAlignment="Left" Width="480" x:Name="img_MyPix"
  Source="snoqualmie-NF.jpg"
  AutomationProperties.LabeledBy="{Binding ElementName=caption_MyPix}"/>
<TextBlock x:Name="caption_MyPix">Mount Snoqualmie Skiing</TextBlock>
```

Labels and LabeledBy

The preferred way to associate a label with a form element is to use a **TextBlock** with an **x:Name** for label text, and then to set the **AutomationProperties.LabeledBy** attached property on the form element to reference the labeling **TextBlock** by its XAML name. If you use this pattern, assistive technologies can use the label text as the accessible name for the form field. Here's an example that shows this technique.

XAML

XML

```
<StackPanel x:Name="LayoutRoot" Background="White">
   <StackPanel Orientation="Horizontal">
     <TextBlock Name="lbl_FirstName">First name</TextBlock>
     <TextBox
      AutomationProperties.LabeledBy="{Binding ElementName=lbl_FirstName}"
      Name="tbFirstName" Width="100"/>
   </StackPanel>
   <StackPanel Orientation="Horizontal">
     <TextBlock Name="lbl_LastName">Last name</TextBlock>
     <TextBox
      AutomationProperties.LabeledBy="{Binding ElementName=lbl_LastName}"
      Name="tbLastName" Width="100"/>
   </StackPanel>
 </StackPanel>
```

Accessible description (optional)

An accessible description provides additional accessibility information about a particular UI element. You typically provide an accessible description when an accessible name alone does not adequately convey an element's purpose.

The Narrator screen reader reads an element's accessible description only when the user requests more information about the element by pressing CapsLock+F.

The accessible name is meant to identify the control rather than to fully document its behavior. If a brief description is not enough to explain the control, you can set the **AutomationProperties.HelpText** attached property in addition to **AutomationProperties.Name**.

Testing accessibility early and often

Ultimately, the best approach for supporting screen readers is to test your app using a screen reader yourself. That will show you how the screen reader behaves and what basic accessibility information might be missing from the app. Then you can adjust the UI or UI Automation property values accordingly. For more info, see Accessibility testing.

One of the tools you can use for testing accessibility is called **AccScope**. The **AccScope** tool is particularly useful because you can see visual representations of your UI that represent how assistive technologies might view your app as an automation tree. In particular, there's a Narrator mode that gives a view of how Narrator gets text from your app and how it organizes the elements in the UI. AccScope is designed so that it can be used and be useful throughout a development cycle for an app, even during the preliminary design phase. For more info see AccScope.

Accessible names from dynamic data

Windows supports many controls that can be used to display values that come from an associated data source, through a feature known as *data binding*. When you populate lists with data items, you may need to use a technique that sets accessible names for data-bound list items after the initial list is populated. For more info, see "Scenario 4" in the XAML accessibility sample.

Accessible names and localization

To make sure that the accessible name is also an element that is localized, you should use correct techniques for storing localizable strings as resources and then referencing

the resource connections with x:Uid directive values. If the accessible name is coming from an explicitly set **AutomationProperties.Name** usage, make sure that the string there is also localizable.

Note that attached properties such as the **AutomationProperties** properties use a special qualifying syntax for the resource name, so that the resource references the attached property as applied to a specific element. For example, the resource name for **AutomationProperties.Name** as applied to a UI element named `MediumButton` is: `MediumButton.[using:Windows.UI.Xaml.Automation]AutomationProperties.Name`.

Related topics

- Accessibility
- **AutomationProperties.Name**
- XAML accessibility sample
- Accessibility testing

Keyboard accessibility

Article • 06/11/2024

Building keyboard accessibility (for traditional, modified, or keyboard emulation hardware) into your app, provides users who are blind, have low vision, motor disabilities, or have little or no use of their hands, the ability to navigate through and use the full functionality of your app. In addition, users without disabilities might choose the keyboard for navigation due to preference or efficiency.

If your app does not provide good keyboard access, users who are blind or have mobility issues might have difficulty using your app.

Keyboard navigation among UI elements

To interact with a control using the keyboard, the control must have focus. To receive focus (without using a pointer), the control must be accessible through tab navigation. By default, the tab order of controls is the same as the order in which they are added to a design surface, declared in XAML, or programmatically added to a container.

Typically, the default tab order is based on how controls are defined in XAML, particularly as that is the order in which the controls are traversed by screen readers. However, the default order does not necessarily correspond to the visual order. The actual display position might depend on the parent layout container and various properties of child elements that can affect the layout.

To ensure your app has an optimal tab order, test the behavior yourself. If you use a grid or table for your layout, the order in which users might read the screen versus the tab order could be very different. This isn't always a problem, but just make sure to test your app's functionality through both touch and keyboard to verify that your UI is optimized for both input methods.

You can make the tab order match the visual order by either adjusting the XAML or overriding the default tab order. THe following example shows how to use the **TabIndex** property with a **Grid** layout that uses column-first tab navigation.

XAML

```
<Grid>
  <Grid.RowDefinitions>...</Grid.RowDefinitions>
  <Grid.ColumnDefinitions>...</Grid.ColumnDefinitions>

  <TextBlock Grid.Column="1" HorizontalAlignment="Center">Groom</TextBlock>
  <TextBlock Grid.Column="2" HorizontalAlignment="Center">Bride</TextBlock>
```

```
  <TextBlock Grid.Row="1">First name</TextBlock>
  <TextBox x:Name="GroomFirstName" Grid.Row="1" Grid.Column="1"
TabIndex="1"/>
  <TextBox x:Name="BrideFirstName" Grid.Row="1" Grid.Column="2"
TabIndex="3"/>

  <TextBlock Grid.Row="2">Last name</TextBlock>
  <TextBox x:Name="GroomLastName" Grid.Row="2" Grid.Column="1"
TabIndex="2"/>
  <TextBox x:Name="BrideLastName" Grid.Row="2" Grid.Column="2"
TabIndex="4"/>
</Grid>
```

In some cases, you might want to exclude a specific control from the tab order. This is typically accomplished by making the control noninteractive by setting its **IsEnabled** property to **false**. A disabled control is automatically excluded from the tab order.

If you want to exclude an interactive control from the tab order, you can set the **IsTabStop** property to **false**.

By default, UI elements that support focus are typically included in the tab order. Some exceptions to this include certain text-display types (such as **RichTextBlock**) that support focus for text selection and clipboard access but are not in the tab order because they are static text elements. These controls are not conventionally interactive (they can't be invoked, and don't require text input, but do support the Text control pattern that supports finding and adjusting selection points in text). Text controls will still be detected by assistive technologies, and read aloud in screen readers, but that relies on techniques other than tab order.

Whether you adjust **TabIndex** values or use the default order, these rules apply:

- If **TabIndex** is not set on an element, the default value is Int32.MaxValue and the tab order is based on declaration order in the XAML or child collections.
- If **TabIndex** is set on an element:
  - UI elements with **TabIndex** equal to 0 are added to the tab order based on declaration order in XAML or child collections.
  - UI elements with **TabIndex** greater than 0 are added to the tab order based on the **TabIndex** value.
  - UI elements with **TabIndex** less than 0 are added to the tab order and appear before any zero value.

The following code snippet shows a collection of elements with various **TabIndex** settings (`B` is assigned the value of Int32.MaxValue, or 2,147,483,647).

XAML

```
<StackPanel Background="#333">
  <StackPanel Background="#FF33FF">
    <Button>A</Button>
    <Button TabIndex="2147483647">B</Button>
    <Button>C</Button>
  </StackPanel>
  <StackPanel Background="#33FFFF">
    <Button TabIndex="1">D</Button>
    <Button TabIndex="1">E</Button>
    <Button TabIndex="0">F</Button>
  </StackPanel>
</StackPanel>
```

This results in the following tab order:

1. F
2. D
3. E
4. A
5. B
6. C

Keyboard navigation between application panes with F6

An application pane is a logical area of prominent, related UI within an application window (for example, Microsoft Edge panes include the address bar, the bookmark bar, the tab bar, and the content panel). The F6 key can be used to navigate between these panes, where groups of child elements can then be accessed using standard keyboard navigation.

While keyboard navigation can provide an accessibly-compliant UI, making an accessibly-*usable* UI often require a few more steps. Typically, this includes:

- Listening to **F6** to navigate between important sections of your UI.
- Adding **keyboard shortcuts** for common actions in your UI.
- Adding **access keys** to important controls in your UI.

See Keyboard shortcuts below and Access keys for more guidance about implementing shortcuts and access keys.

Optimize for F6

F6 lets keyboard users efficiently navigate between panes of UI without tabbing through potentially hundreds of controls.

For example, F6 in Microsoft Edge cycles between the address bar, the bookmark bar, the tab bar, and the content panel. As a web page can potentially have hundreds of tabable controls, F6 can make it easier for keyboard users to reach the tab bar and address bar without using application-specific shortcuts.

The F6 tab cycle can also loosely correspond to landmarks or headings in content, though it doesn't need to match exactly. F6 should focus on large, distinct regions in your UI, whereas landmarks can be more granular. For example, you might mark an app bar and its search box as landmarks, but only include the app bar itself in the F6 cycle.

> **ⓘ Important**
>
> You must implement F6 navigation in your app as it is not supported natively.

Where possible, regions in the F6 cycle should have an accessible name: either through a landmark or by manually adding an **AutomationProperties.Name** to the "root" element of the region.

Shift-F6 should cycle in the opposite direction.

Keyboard navigation within a UI element

For composite controls, it is important to ensure proper inner navigation among the contained elements. A composite control can manage the currently active child element to reduce the overhead of having all child elements support focus. The composite control is included in the tab order and handles keyboard navigation events itself. Many composite controls already have some inner navigation logic built into their event handling. For example, arrow-key traversal of items is enabled by default on the **ListView**, **GridView**, **ListBox** and **FlipView** controls.

Keyboard alternatives to pointer actions and events for specific control elements

UI elements that can be clicked should also be invokable through the keyboard. To use the keyboard with a UI element, the element must have focus (only classes that derive from **Control** support focus and tab navigation).

For UI elements that can be invoked, implement keyboard event handlers for the Spacebar and Enter keys. This ensures basic keyboard accessibility support and lets

users reach all interactive UI elements and activate functionality by using the keyboard only.

Where an element does not support focus, you can create your own custom control. In this case, to enable focus, you must set the **IsTabStop** property to **true** and you must provide a visual indication of the focused visual state with a focus indicator.

However, it can be easier to use control composition so that the support for tab stops, focus, and Microsoft UI Automation peers and patterns are handled by the control within which you choose to compose your content. For example, instead of handling a pointer-pressed event on an **Image**, wrap that element in a **Button** to get pointer, keyboard, and focus support.

XAML

```
<!--Don't do this.-->
<Image Source="sample.jpg" PointerPressed="Image_PointerPressed"/>

<!--Do this instead.-->
<Button Click="Button_Click"><Image Source="sample.jpg"/></Button>
```

Keyboard shortcuts

In addition to implementing keyboard navigation and activation, it is also good practice to implement keyboard shortcuts such as keyboard accelerators and access keys for important or frequently used functionality.

A *shortcut* is a keyboard combination that provides an efficient way for the user to access app functionality. There are two kinds of shortcut:

- *Accelerators* are shortcuts that invoke an app command. Your app may or may not provide specific UI that corresponds to the command. Accelerators typically consist of the Ctrl key plus a letter key.
- *Access keys* are shortcuts that set focus to specific UI in your application. Access keys typicaly consist of the Alt key plus a letter key.

Always provide an easy way for users who rely on screen readers and other assistive technology to discover your app's shortcut keys. Communicate shortcut keys by using tooltips, accessible names, accessible descriptions, or some other form of on-screen communication. At a minimum, shortcut keys should be well documented in your app's Help content.

You can document access keys through screen readers by setting the AutomationProperties.AccessKey attached property to a string that describes the shortcut key. There is also an AutomationProperties.AcceleratorKey attached property for documenting non-mnemonic shortcut keys, although screen readers generally treat both properties the same way. Try to document shortcut keys in multiple ways, using tooltips, automation properties, and written Help documentation.

The following example demonstrates how to document shortcut keys for media play, pause, and stop buttons.

XAML

```
<Grid KeyDown="Grid_KeyDown">

  <Grid.RowDefinitions>
    <RowDefinition Height="Auto" />
    <RowDefinition Height="Auto" />
  </Grid.RowDefinitions>

  <MediaElement x:Name="DemoMovie" Source="xbox.wmv"
    Width="500" Height="500" Margin="20" HorizontalAlignment="Center" />

  <StackPanel Grid.Row="1" Margin="10"
    Orientation="Horizontal" HorizontalAlignment="Center">

    <Button x:Name="PlayButton" Click="MediaButton_Click"
      ToolTipService.ToolTip="Shortcut key: Ctrl+P"
      AutomationProperties.AcceleratorKey="Control P">
      <TextBlock>Play</TextBlock>
    </Button>

    <Button x:Name="PauseButton" Click="MediaButton_Click"
      ToolTipService.ToolTip="Shortcut key: Ctrl+A"
      AutomationProperties.AcceleratorKey="Control A">
      <TextBlock>Pause</TextBlock>
    </Button>

    <Button x:Name="StopButton" Click="MediaButton_Click"
      ToolTipService.ToolTip="Shortcut key: Ctrl+S"
      AutomationProperties.AcceleratorKey="Control S">
      <TextBlock>Stop</TextBlock>
    </Button>
  </StackPanel>
</Grid>
```

ⓘ Important

Setting the AutomationProperties.AcceleratorKey or AutomationProperties.AccessKey doesn't enable keyboard functionality. This only

indicates what keys should be used to the UI Automation framework and then can be passed on to users via assistive technologies.

Key handling is implemented in code-behind, not XAML. You still need to attach handlers for **KeyDown** or **KeyUp** events on the relevant control in order to actually implement the keyboard shortcut behavior in your app. Also, the underline text decoration for an access key is not provided automatically. You must explicitly underline the text for the specific key in your mnemonic as inline **Underline** formatting if you wish to show underlined text in the UI.

For simplicity, the preceding example omits the use of resources for strings such as "Ctrl+A". However, you must also consider shortcut keys during localization. Localizing shortcut keys is relevant because the choice of key to use as the shortcut key typically depends on the visible text label for the element.

For more guidance about implementing shortcut keys, see Shortcut keys in the Windows User Experience Interaction Guidelines.

Implementing a key event handler

Input events (such as the key events) use an event concept called *routed events*. A routed event can bubble up through the child elements of a parent composite control, such that the parent control can handle events for multiple child elements. This event model is convenient for defining shortcut key actions for a control that contains several child elements, none of which can have focus or be part of the tab order.

For example code that shows how to write a key event handler that includes checking for modifiers such as the Ctrl key, see Keyboard interactions.

Keyboard navigation for custom controls

We recommend using the arrow keys as keyboard shortcuts for navigating among child elements in cases where the child elements have a spacial relationship to each other. If tree-view nodes have separate sub-elements for handling expand-collapse and node activation, use the left and right arrow keys to provide keyboard expand-collapse functionality. If you have an oriented control that supports directional traversal within the control content, use the appropriate arrow keys.

Generally you implement custom key handling for custom controls by including an override of the **OnKeyDown** and **OnKeyUp** methods as part of the class logic.

An example of a visual state for a focus indicator

As mentioned earlier, any custom control that supports focus should have a visual focus indicator. Typically, that focus indicator is just a rectangle outlining the bounding rectangle of the control. The **Rectangle** for visual focus is a peer element to the rest of the control's composition in a control template, but is initially set with a **Visibility** value of **Collapsed** because the control isn't focused yet. When the control does get focus, a visual state is invoked that specifically sets the **Visibility** of the focus visual to **Visible**. Once focus is moved elsewhere, another visual state is called, and the **Visibility** becomes **Collapsed**.

All focusable XAML controls display an appropriate visual focus indicator when focused. The user's selected them can also affect the indicator appearance (particularly if the user is using a high contrast mode). If you're using the XAML controls in your UI (and are not replacing the control templates), you don't need to do anything extra to get default visual focus indicators. However, if you intend to retemplate a control, or if you're curious about how XAML controls provide their visual focus indicators, the remainder of this section explains how this is done in XAML and the control logic.

Here's some example XAML that comes from the default XAML template for a **Button**.

XAML

XML

```
<ControlTemplate TargetType="Button">
...
    <Rectangle
      x:Name="FocusVisualWhite"
      IsHitTestVisible="False"
      Stroke="{ThemeResource FocusVisualWhiteStrokeThemeBrush}"
      StrokeEndLineCap="Square"
      StrokeDashArray="1,1"
      Opacity="0"
      StrokeDashOffset="1.5"/>
    <Rectangle
      x:Name="FocusVisualBlack"
      IsHitTestVisible="False"
      Stroke="{ThemeResource FocusVisualBlackStrokeThemeBrush}"
      StrokeEndLineCap="Square"
      StrokeDashArray="1,1"
      Opacity="0"
      StrokeDashOffset="0.5"/>
...
</ControlTemplate>
```

So far this is just the composition. To control the focus indicator's visibility, you define visual states that toggle the **Visibility** property. This is done using the VisualStateManager and the VisualStateManager.VisualStateGroups attached property, as applied to the root element that defines the composition.

XAML

```
<ControlTemplate TargetType="Button">
  <Grid>
    <VisualStateManager.VisualStateGroups>
       <!--other visual state groups here-->
       <VisualStateGroup x:Name="FocusStates">
         <VisualState x:Name="Focused">
           <Storyboard>
             <DoubleAnimation
               Storyboard.TargetName="FocusVisualWhite"
               Storyboard.TargetProperty="Opacity"
               To="1" Duration="0"/>
             <DoubleAnimation
               Storyboard.TargetName="FocusVisualBlack"
               Storyboard.TargetProperty="Opacity"
               To="1" Duration="0"/>
         </VisualState>
         <VisualState x:Name="Unfocused" />
         <VisualState x:Name="PointerFocused" />
       </VisualStateGroup>
     <VisualStateManager.VisualStateGroups>
<!--composition is here-->
   </Grid>
</ControlTemplate>
```

Note how only one of the named states adjusts **Visibility** directly whereas the others are apparently empty. With visual states, as soon as the control uses another state from the same **VisualStateGroup**, any animations applied by the previous state are immediately canceled. Because the default **Visibility** from composition is **Collapsed**, the rectangle will not appear. The control logic controls this by listening for focus events like **GotFocus** and changing the states with **GoToState**. Often this is already handled for you if you are using a default control or customizing based on a control that already has that behavior.

Keyboard accessibility and devices without a hardware keyboard

Some devices don't have a dedicated, hardware keyboard and rely on a Soft Input Panel (SIP) instead. Screen readers can read text input from the **Text** SIP and users can discover where their fingers are because the screen reader can detect that the user is scanning keys, and reads the scanned key name aloud. Also, some of the keyboard-

oriented accessibility concepts can be mapped to related assistive technology behaviors that don't use a keyboard at all. For example, even though a SIP won't include a Tab key, Narrator supports a touch gesture that's the equivalent of pressing the Tab key, so having a useful tab order through the controls in a UI is still crtical for accessibility. Narrator also supports many other touch gestures, including arrow keys for navigating within complex controls (see Narrator keyboard commands and touch gestures).

Examples

Open the WinUI 3 Gallery app and see keyboard accessibility support in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Related topics

- Accessibility
- Keyboard interactions
- Touch keyboard sample
- XAML accessibility sample

Screen readers and hardware system buttons

Article • 06/24/2021

Screen-readers, such as Narrator, must be able to recognize and handle hardware system button events and communicate their state to users. In some cases, the screen reader might need to handle these hardware button events exclusively and not let them bubble up to other handlers.

Beginning with Windows 10 version 2004, UWP applications can listen for and handle the **Fn** hardware system button events in the same way as other hardware buttons. Previously, this system button acted only as a modifier for how other hardware buttons reported their events and state.

> **Note**
>
> Fn button support is OEM-specific and can include features such as the ability to toggle/lock on or off (vs. a press-and-hold key combination), along with a corresponding lock indicator light (which might not be helpful to users who are blind or have a vision impairment).

Fn button events are exposed through a new SystemButtonEventController Class in the Windows.UI.Input namespace. The SystemButtonEventController object supports the following events:

- SystemFunctionButtonPressed
- SystemFunctionButtonReleased
- SystemFunctionLockChanged
- SystemFunctionLockIndicatorChanged

> **Important**
>
> The SystemButtonEventController cannot receive these events if they have already been handled by a higher priority handler.

Examples

In the following examples, we show how to create a SystemButtonEventController based on a DispatcherQueue and handle the four events supported by this object.

It is common for more than one of the supported events to fire when the Fn button is pressed. For example, pressing the Fn button on a Surface keyboard fires SystemFunctionButtonPressed, SystemFunctionLockChanged, and SystemFunctionLockIndicatorChanged at the same time.

1. In this first snippet, we simply include the required namespaces and specify some global objects, including the DispatcherQueue and the DispatcherQueueController objects for managing the SystemButtonEventController thread.

   We then specify the event tokens returned when registering the SystemButtonEventController event-handling delegates.

   C++/WinRT

   ```
   namespace winrt
   {
       using namespace Windows::System;
       using namespace Windows::UI::Input;
   }

   ...

   // Declare related members
   winrt::DispatcherQueueController _queueController;
   winrt::DispatcherQueue _queue;
   winrt::SystemButtonEventController _controller;
   winrt::event_token _fnKeyDownToken;
   winrt::event_token _fnKeyUpToken;
   winrt::event_token _fnLockToken;
   ```

2. We also specify an event token for the SystemFunctionLockIndicatorChanged event along with a bool to indicate whether the application is in "Learning Mode" (where the user is simply trying to explore the keyboard without performing any functions).

   C++/WinRT

   ```
   winrt::event_token _fnLockIndicatorToken;
   bool _isLearningMode = false;
   ```

3. This third snippet includes the corresponding event handler delegates for each event supported by the SystemButtonEventController object.

Each event handler announces the event that has occurred. In addition, the FunctionLockIndicatorChanged handler also controls whether the app is in "Learning" mode (`_isLearningMode` = true), which prevents the event from bubbling to other handlers and lets the user explore keyboard features without actually performing the action.

C++/WinRT

```
void SetupSystemButtonEventController()
{
    // Create dispatcher queue controller and dispatcher queue
    _queueController =
winrt::DispatcherQueueController::CreateOnDedicatedThread();
    _queue = _queueController.DispatcherQueue();

    // Create controller based on new created dispatcher queue
    _controller =
winrt::SystemButtonEventController::CreateForDispatcherQueue(_queue);

    // Add Event Handler for each different event
    _fnKeyDownToken = _controller->FunctionButtonPressed(
        [](const winrt::SystemButtonEventController& /*sender*/, const
winrt:: FunctionButtonEventArgs& args)
        {
            // Mock function to read the sentence "Fn button is
pressed"
            PronounceFunctionButtonPressedMock();
            // Set Handled as true means this event is consumed by this
controller
            // no more targets will receive this event
            args.Handled(true);
        });

        _fnKeyUpToken = _controller->FunctionButtonReleased(
            [](const winrt::SystemButtonEventController& /*sender*/,
const winrt:: FunctionButtonEventArgs& args)
            {
                // Mock function to read the sentence "Fn button is up"
                PronounceFunctionButtonReleasedMock();
                // Set Handled as true means this event is consumed by
this controller
                // no more targets will receive this event
                args.Handled(true);
            });

    _fnLockToken = _controller->FunctionLockChanged(
        [](const winrt::SystemButtonEventController& /*sender*/, const
winrt:: FunctionLockChangedEventArgs& args)
        {
            // Mock function to read the sentence "Fn shift is
locked/unlocked"
            PronounceFunctionLockMock(args.IsLocked());
```

```
            // Set Handled as true means this event is consumed by this
controller
            // no more targets will receive this event
            args.Handled(true);
        });

    _fnLockIndicatorToken = _controller->FunctionLockIndicatorChanged(
        [](const winrt::SystemButtonEventController& /*sender*/, const
winrt:: FunctionLockIndicatorChangedEventArgs& args)
        {
            // Mock function to read the sentence "Fn lock indicator is
on/off"
            PronounceFunctionLockIndicatorMock(args.IsIndicatorOn());
            // In learning mode, the user is exploring the keyboard.
They expect the program
            // to announce what the key they just pressed WOULD HAVE
DONE, without actually
            // doing it. Therefore, handle the event when in learning
mode so the key is ignored
            // by the system.
            args.Handled(_isLearningMode);
        });
}
```

See also

SystemButtonEventController Class

Landmarks and Headings

Article • 06/11/2024

Landmarks and **headings** help users of assistive technology (AT) navigate a UI more efficiently by uniquely identifying different sections of a user interface.

Overview

A user interface is typically organized in a visually efficient way, allowing a sighted user to quickly skim for what interests them without having to slow down to read *all* the content. A screen reader user needs to have this same skimming ability. Marking content as landmarks and headings provides the user of a screen reader the option to skim content similar to the way a sighted user would.

The concepts of ARIA landmarks, ARIA headings, and HTML headings have been used in web content for years to allow faster navigation by screen reader users. Web pages utilize landmarks and headings to make their content more usable by allowing the AT user to quickly get to the large chunk (landmark) and smaller chunk (heading).

Specifically, screen readers have commands allowing users to jump between landmarks and jump between headings (next/previous or specific heading level).

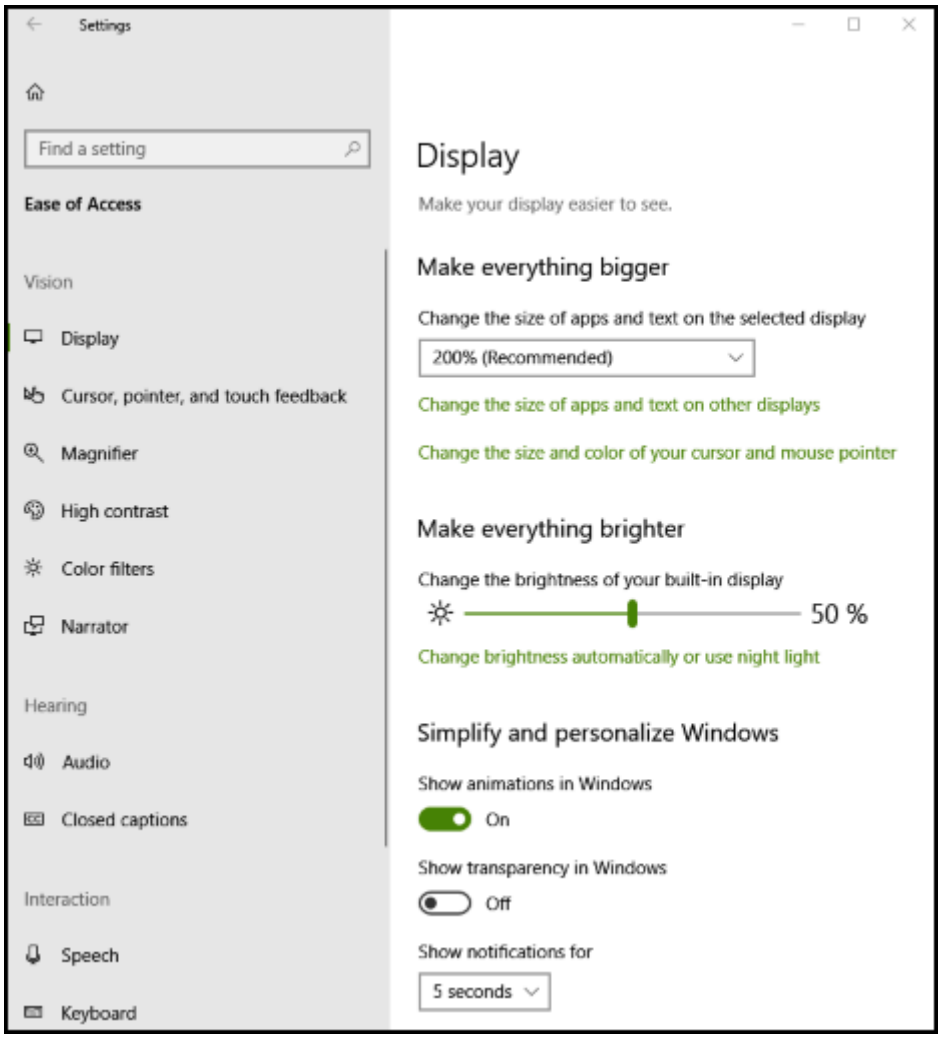
Landmarks enable content to be grouped into various categories such as *search*, *navigation*, *main content*, and so on. Once grouped, the AT user can quickly navigate between the groups. This quick navigation allows the user to skip potentially substantial amounts of content that previously required navigation item by item.

For example, when using a tab panel, consider making it a *navigation* landmark. When using a search edit box, consider making it a *search* landmark, and consider setting your main content as a *main content* landmark.

Whether within a landmark or even outside a landmark, consider annotating sub-elements as headings with logical heading levels.

The Windows Settings app

The following image shows the **Ease of Access** page in a previous version of the Windows Settings app.



For this page, the search edit box is wrapped within a search landmark, the navigation elements on the left are wrapped within a navigation landmark, and the main content on the right is wrapped within a main content landmark.

Within the navigation landmark there is a main group heading called **Ease of Access** (heading level 1) with sub-options of **Vision**, **Hearing**, and so on (heading level 2). Within the main content, **Display** is set to heading level 1 with sub-groups such as **Make everything bigger** set to heading level 2.

The Settings app would be accessible without landmarks and headings, but it becomes much more usable with them. In this case, a user with a screen reader can quickly get to the group (landmark) they're interested in, and from there they can then quickly get to the sub-group (heading).

Usage

Use AutomationProperties.LandmarkTypeProperty to identify the type of landmark for a UI element. This landmark UI element would then encapsulate all the other UI elements relevant to that landmark.

Use AutomationProperties.LocalizedLandmarkTypeProperty to name the landmark. If you select a predefined landmark type, such as main or navigation, these names will be used for the landmark name. However, if you set the landmark type to custom, you must name the landmark through this property (you can also use this property to override the default names from the pre-defined landmark types).

Use AutomationProperties.HeadingLevel to set the UI element as a heading of a specific level from *Level1* through *Level9*.

Use the F6 key and handler to support navigation between landmarks, which is a common pattern in complex apps like File Explorer and Outlook. See Keyboard navigation between application panes with F6 for more guidance.

Examples

See the Code samples for resolving common programmatic accessibility issues in Windows desktop apps to resolve many common programmatic accessibility issues in Windows desktop apps.

These code samples are referenced directly by Microsoft Accessibility Insights for Windows, which can help to identify accessibility issues in your app UI.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Contrast themes

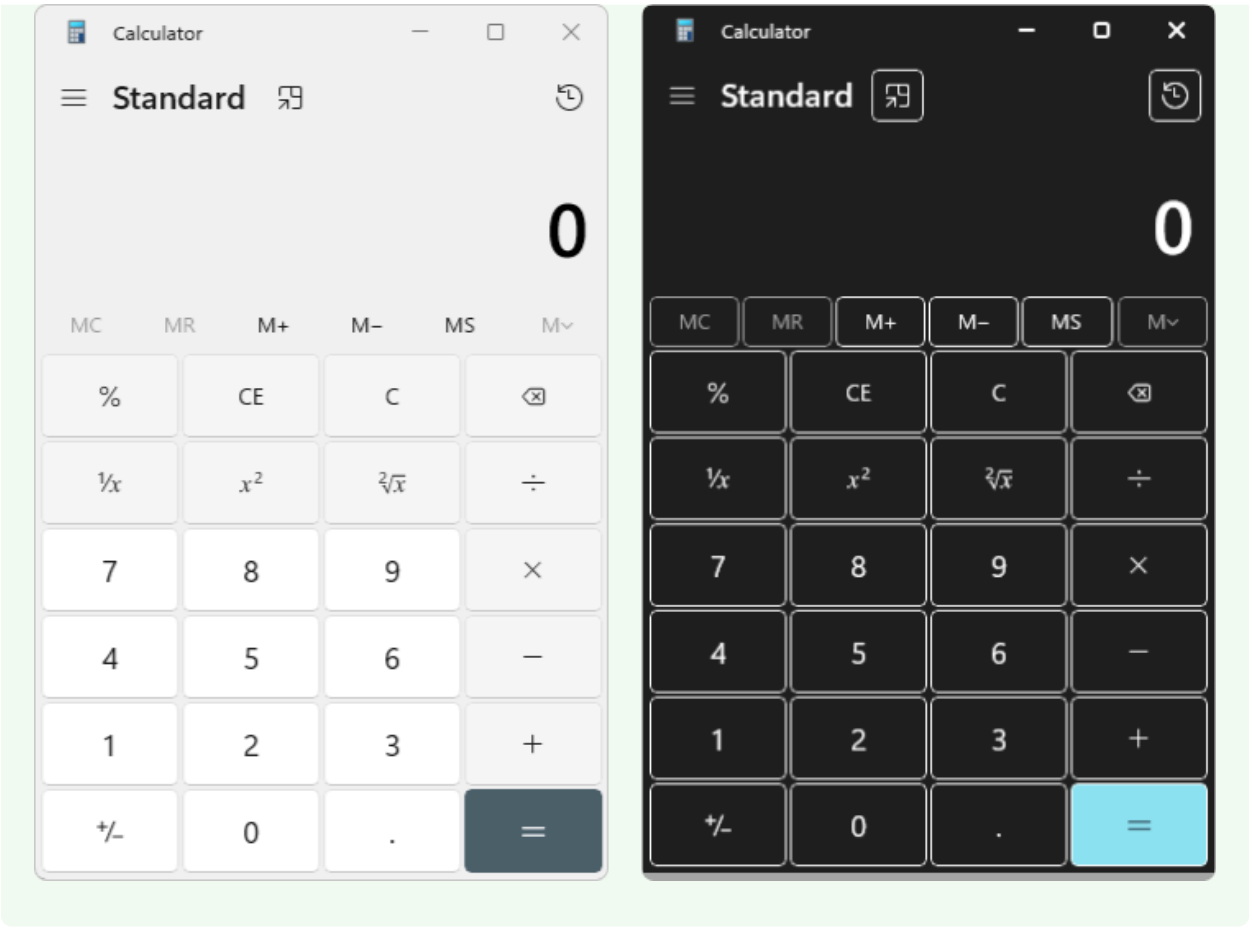
Article • 02/27/2025

Contrast themes use a small palette of colors (with a contrast ratio of at least 7:1) to help make elements in the UI easier to see, reduce eye strain, improve text readability, and accommodate user preferences.

> **Note**
>
> Don't confuse contrast themes with light and dark themes, which support a much larger color palette and don't necessarily increase contrast or make things easier to see. For more on light and dark themes, see Color.

To see how your app behaves with contrast themes, enable and customize them through the *Settings* > *Accessibility* > *Contrast themes* page.

> **Tip**
>
> You can also press the left-Alt key + Shift key + Print screen (PrtScn on some keyboards) to quickly turn contrast themes on or off. If you have not selected a theme previously, the Aquatic theme is used by default (shown in the following image).



Setting HighContrastAdjustment to None

Windows apps have HighContrastAdjustment turned on by default. This sets all text color to white with a solid black highlight behind it, ensuring sufficient contrast against all backgrounds. If you are using brushes correctly, this setting should be turned off.

Detecting high contrast

You can programmatically check if the current theme is a contrast theme through the AccessibilitySettings class (you must call the **AccessibilitySettings** constructor from a scope where the app is initialized and is already displaying content).

Creating theme dictionaries

A ResourceDictionary.ThemeDictionaries object can indicate theme colors that are different from the system-defined colors by specifying brushes for the **Default** (Dark), **Light**, and **HighContrast** contrast themes.

> **Tip**

Contrast theme refers to the feature in general, while **HighContrast** refers to the specific dictionary being referenced.

1. In App.xaml, create a **ThemeDictionaries** collection with both a **Default** and a **HighContrast** ResourceDictionary (a **Light** ResourceDictionary is not necessary for this example).

2. In the **Default** dictionary, create the type of Brush you need (usually a **SolidColorBrush**). Give it an *x:Key* name corresponding to its intended use (a **StaticResource** referencing an existing system brush would also be appropriate).

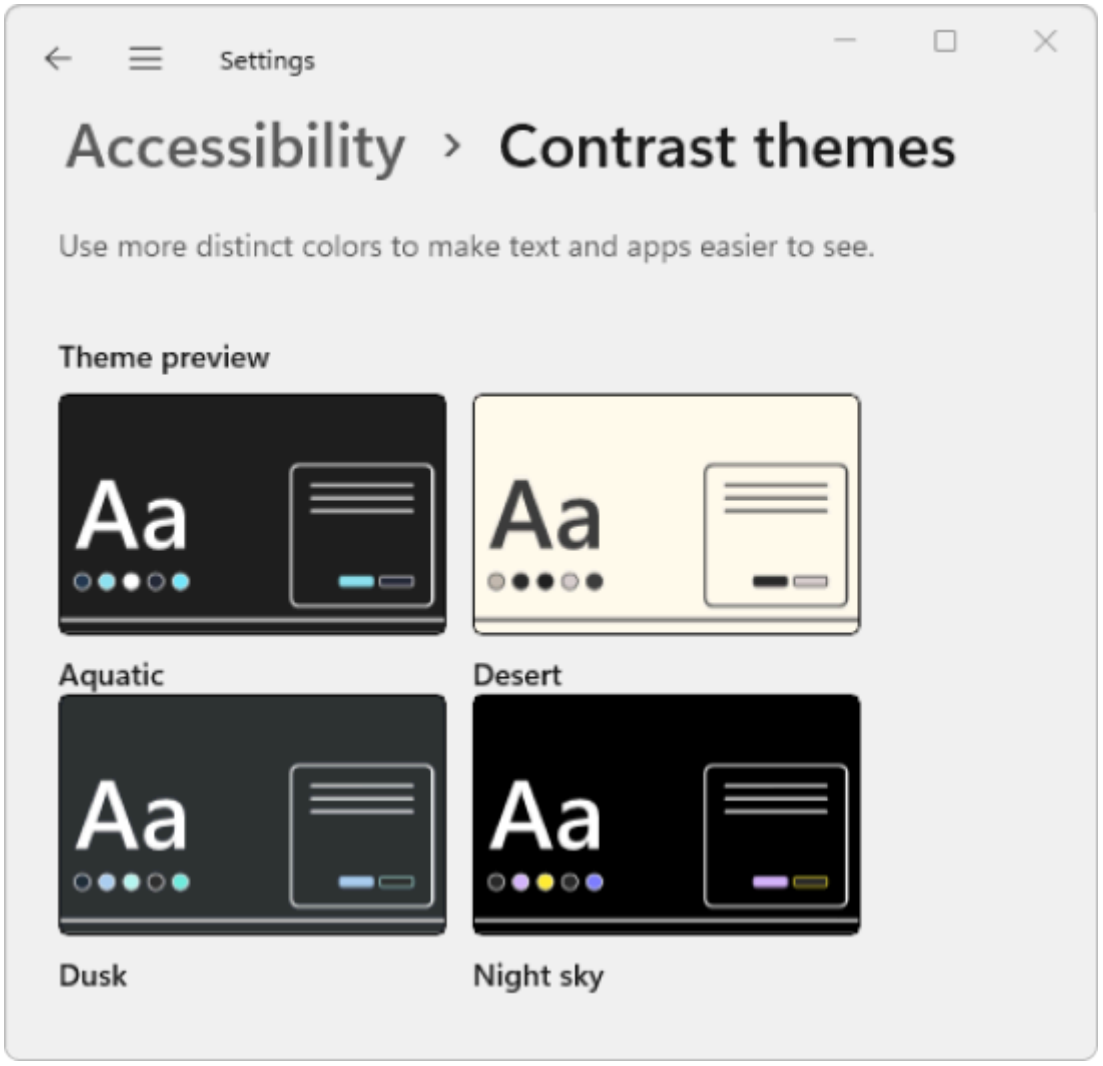
3. In the **HighContrast** ResourceDictionary (shown in the following code snippet), specify an appropriate **SystemColor** brush. See Contrast colors for details on picking one of the dynamic system **HighContrast** colors for the **SystemColor** brush.

XAML

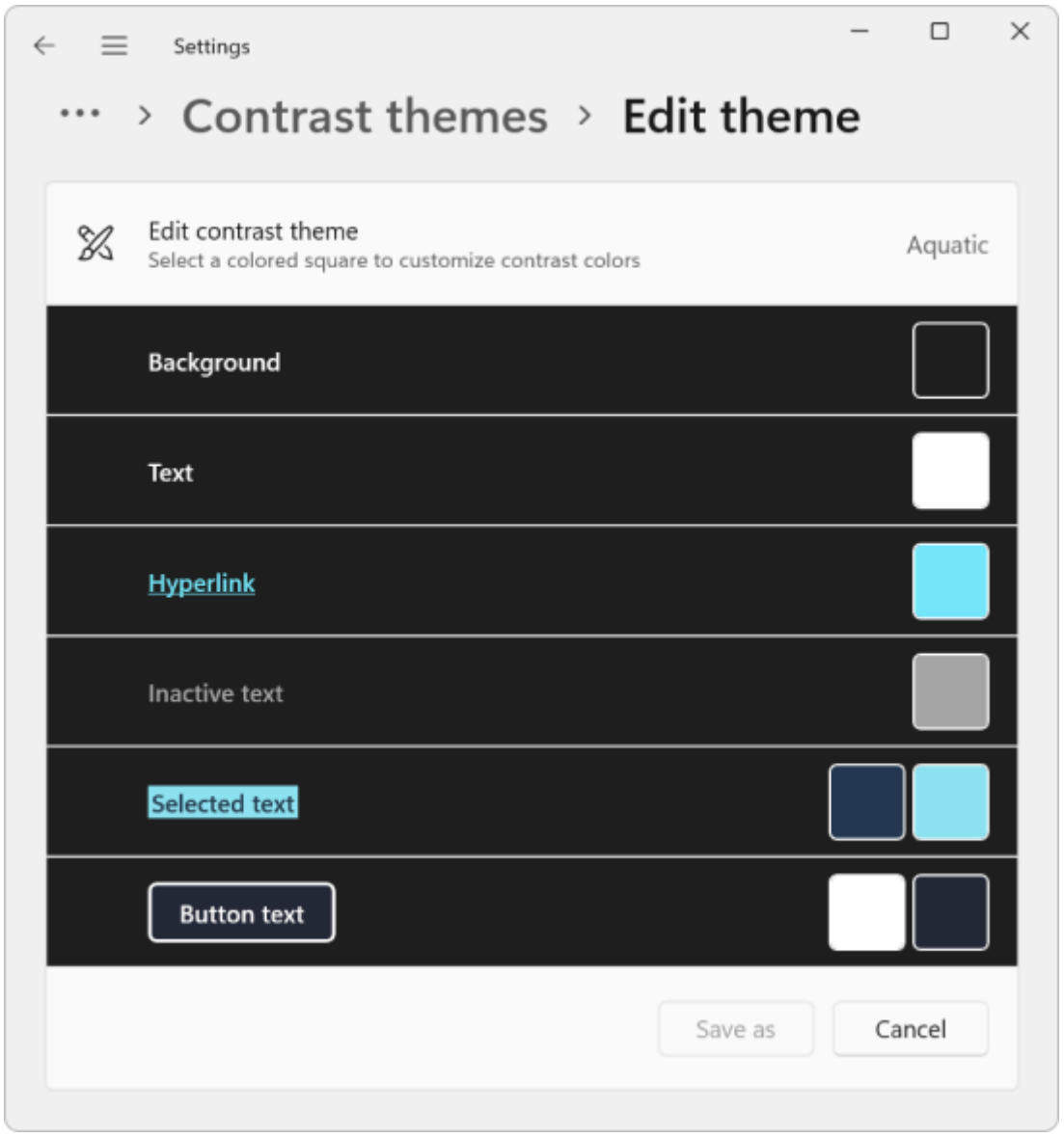
```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.ThemeDictionaries>
            <!-- Default is a fallback if a more precise theme isn't
called
            out below -->
            <ResourceDictionary x:Key="Default">
                <SolidColorBrush x:Key="BrandedPageBackgroundBrush"
Color="#E6E6E6" />
            </ResourceDictionary>

            <!-- Optional, Light is used in light theme.
            If included, Default will be used for Dark theme -->
            <ResourceDictionary x:Key="Light">
                <SolidColorBrush x:Key="BrandedPageBackgroundBrush"
Color="#E6E6E6" />
            </ResourceDictionary>

            <!-- HighContrast is used in all high contrast themes -->
            <ResourceDictionary x:Key="HighContrast">
                <SolidColorBrush x:Key="BrandedPageBackgroundBrush"
Color="{ThemeResource SystemColorWindowColor}" />
            </ResourceDictionary>
        </ResourceDictionary.ThemeDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

Contrast colors

On the *Settings* > *Ease of access* > *Contrast themes* page (shown in the following image), users can select from four default contrast themes: **Aquatic**, **Desert**, **Dusk**, and **Night sky**.



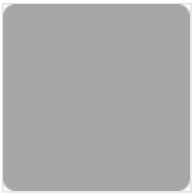
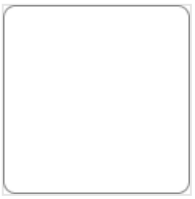
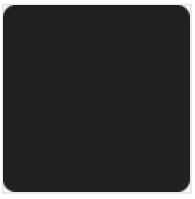
After the user selects an option, they can choose to immediately apply it, or they can edit the theme. The following image shows the Edit theme dialog for the **Aquatic** contrast theme.



This table shows the contrast theme colors and their recommended pairings. Each **SystemColor** resource is a variable that automatically updates the color when the user switches contrast themes.

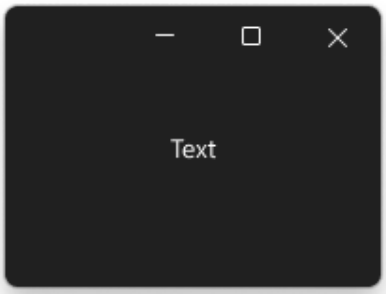
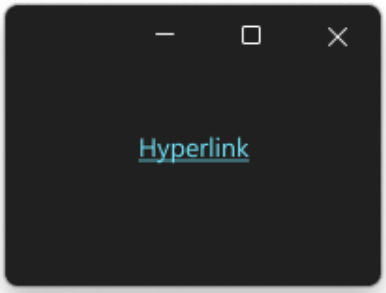
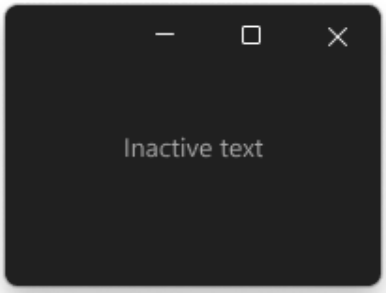
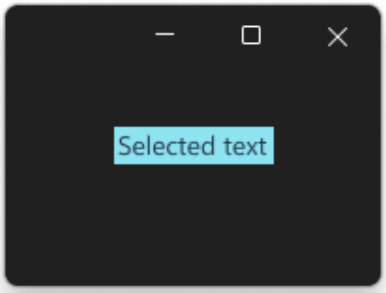
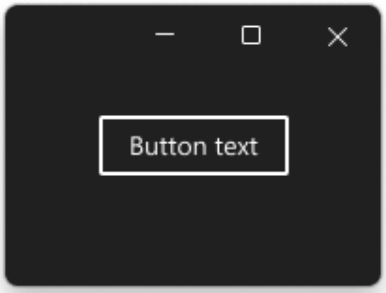
Expand table

Color swatch	Description
	SystemColorWindowColor Background of pages, panes, popups, and windows. Pair with **SystemColorWindowTextColor**
	SystemColorWindowTextColor Headings, body copy, lists, placeholder text, app and window borders, any UI that can't be interacted with. Pair with **SystemColorWindowColor**

Color swatch	Description
	SystemColorHotlightColor Hyperlinks. Pair with **SystemColorWindowColor**
	**SystemColorGrayTextColor** Inactive or disabled UI. Pair with **SystemColorWindowColor**
	SystemColorHighlightTextColor Foreground color of text or UI that is selected, interacted with (hover, pressed), or in progress. Pair with **SystemColorHighlightColor**
	SystemColorHighlightColor Background or accent color of UI that is selected, interacted with (hover, pressed), or in progress. Pair with **SystemColorHighlightTextColor**
	**SystemColorButtonTextColor** Foreground color of buttons and any UI that can be interacted with. Pair with **SystemColorButtonFaceColor**
	**SystemColorButtonFaceColor** Background color of buttons and any UI that can be interacted with. Pair with **SystemColorButtonTextColor**

The next table shows how the colors appear when used on a background set to **SystemColorWindowColor**.

Expand table

Example	Values
	SystemColorWindowTextColor
	SystemColorHotlightColor
	SystemColorGrayTextColor
	SystemColorHighlightTextColor + SystemColorHighlightColor
	SystemColorButtonTextColor + SystemColorButtonFaceColor

In the following code snippet, we show how to pick a resource for **BrandedPageBackgroundBrush**. **SystemColorWindowColor** is a good choice here as **BrandedPageBackgroundBrush** indicates that it will be used for a background.

XAML

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.ThemeDictionaries>
            <!-- Default is a fallback if a more precise theme isn't called
            out below -->
            <ResourceDictionary x:Key="Default">
                <SolidColorBrush x:Key="BrandedPageBackgroundBrush"
Color="#E6E6E6" />
            </ResourceDictionary>

            <!-- Optional, Light is used in light theme.
            If included, Default will be used for Dark theme -->
            <ResourceDictionary x:Key="Light">
                <SolidColorBrush x:Key="BrandedPageBackgroundBrush"
Color="#E6E6E6" />
            </ResourceDictionary>

            <!-- HighContrast is used in all high contrast themes -->
            <ResourceDictionary x:Key="HighContrast">
                <SolidColorBrush x:Key="BrandedPageBackgroundBrush" Color="
{ThemeResource SystemColorWindowColor}" />
            </ResourceDictionary>
        </ResourceDictionary.ThemeDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

The resource is then assigned to the background of an element.

XAML

```
<Grid Background="{ThemeResource BrandedPageBackgroundBrush}">
```

We use `{ThemeResource}` twice in the preceding example, once to reference **SystemColorWindowColor** and again to reference **BrandedPageBackgroundBrush**. Both are required for your app to theme correctly at run time. This is a good time to test out the functionality in your app. The **Grid** background will automatically update as you switch to a high contrast theme. It will also update when switching between different high contrast themes.

> ⓘ **Note**
>
> **WinUI 2.6 and newer**

There are eight high contrast system brushes available for referencing through a **ResourceKey** (see the following example for **SystemColorWindowTextColorBrush**).

```
<StaticResource x:Key="MyControlBackground"
ResourceKey="SystemColorWindowTextColorBrush" />
```

The brush names match one of the eight previously mentioned system colors exactly (with "Brush" appended). We recommend using a **StaticResource** instead of a local **SolidColorBrush** for performance reasons.

Best practices

Here are some recommendations for customizing the contrast theme colors in your Windows app.

- Test in all four high contrast themes while your app is running.
- Be consistent.
- Make sure **HighContrastAdjustment** is set to `None` in your app (it is turned on by default). See Setting HighContrastAdjustment to None.
- **Do not** hard code a color in the HighContrast theme. Instead, use the **SystemColor** `Color` and `ColorBrush` resources. For more detail, see Hard-coded colors.
- **Do not** mix background/foreground pairs that are not compatible
- **Do not** choose color resource for aesthetics. Remember, the colors change with the theme.
- **Do not** use `SystemColorGrayTextColor` for body copy that is secondary or acts as hint text. This is intended for disabled content only.
- **Do not** use `SystemColorHotlightColor` and corresponding brush as both are reserved for hyperlinks.

Tip

It's often helpful to look at the WinUI Gallery app to see how common controls use the **SystemColor** brushes. If installed already, open them by clicking the following links: WinUI 3 Gallery or WinUI 2 Gallery.

If they are not installed, you can download the WinUI 3 Gallery and the WinUI 2 Gallery from the Microsoft Store.

You can also get the source code for both from GitHub (use the *main* branch for WinUI 3 and the *winui2* branch for WinUI 2).

Hard-coded colors

Platform controls provide built-in support for contrast themes, but you should be careful when customizing your application UI. Two of the most common issues occur when either the color of an element is hard-coded or an incorrect **SystemColor** resource is used.

In the following code snippet, we show a Grid element declared with a background color set to `#E6E6E6` (a very light grey). If you hard-code the color in this way, you also override the background color across all themes. For example, if the user selects the **Aquatic** contrast theme, instead of white text on a near black background, the text color in this app changes to white while the background remains light grey. The very low contrast between text and background could make this app very difficult to use.

XAML

```
<Grid Background="#E6E6E6">
```

Instead, we recommend using the **{ThemeResource} markup extension** to reference a color in the **ThemeDictionaries** collection of a **ResourceDictionary**. This enables the automatic substitution of colors and brushes based on the user's current theme.

XAML

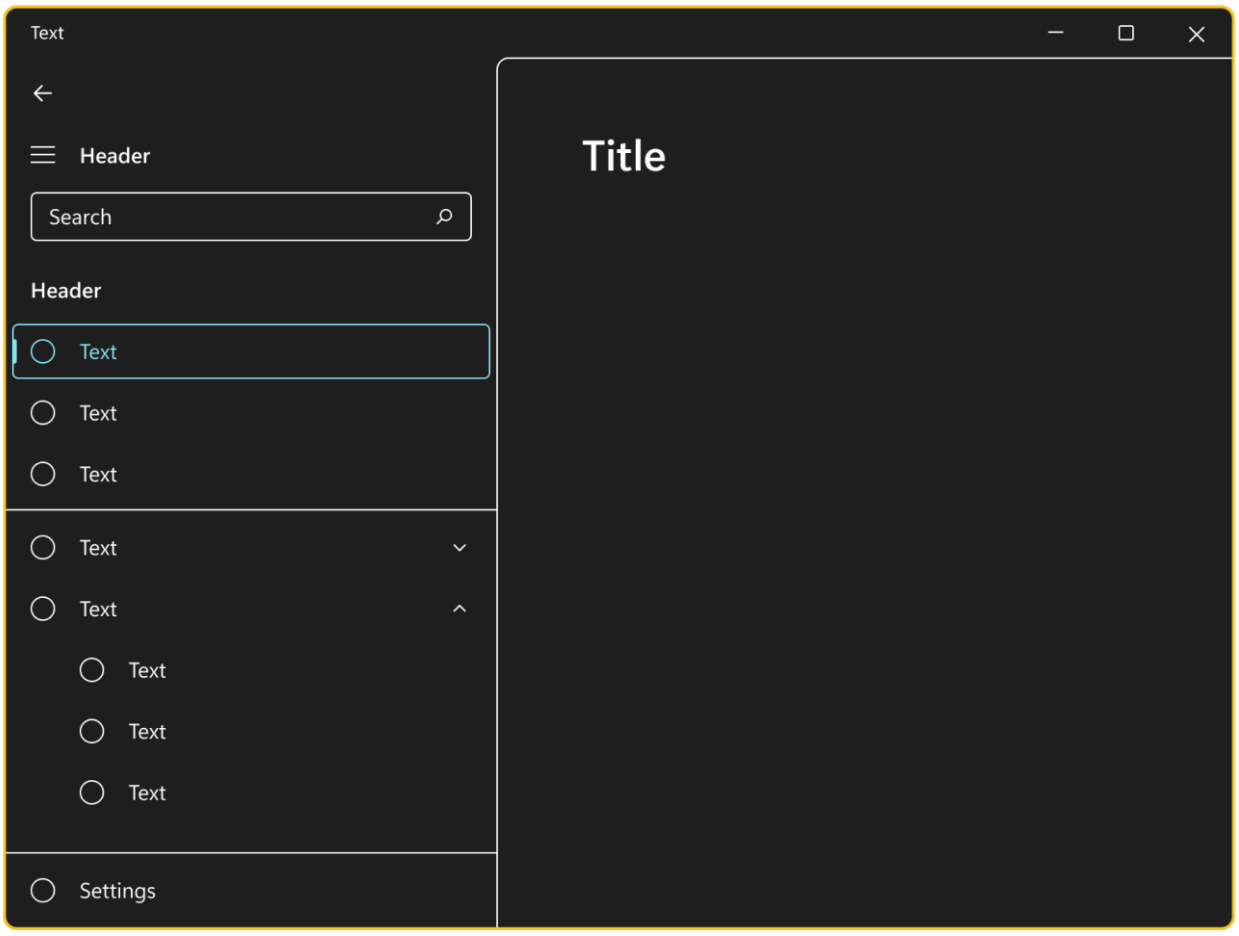
```
<Grid Background="{ThemeResource BrandedPageBackgroundBrush}">
```

Borders

Pages, panes, popups, and bars should all use **SystemColorWindowColor** for their background. Add a contrast theme-only border only where necessary to preserve important boundaries in your UI.

> **Tip**
>
> We recommend using 2px borders for transitory surfaces such as flyouts and dialogs.

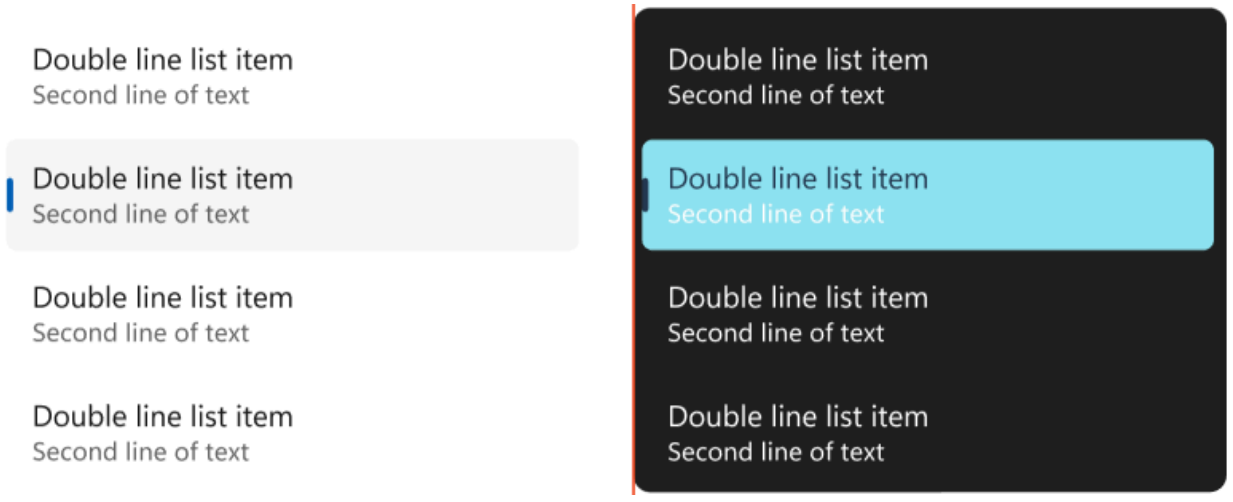
The navigation pane and the page both share the same background color in contrast themes. To distinguish them, a contrast theme-only border is essential.



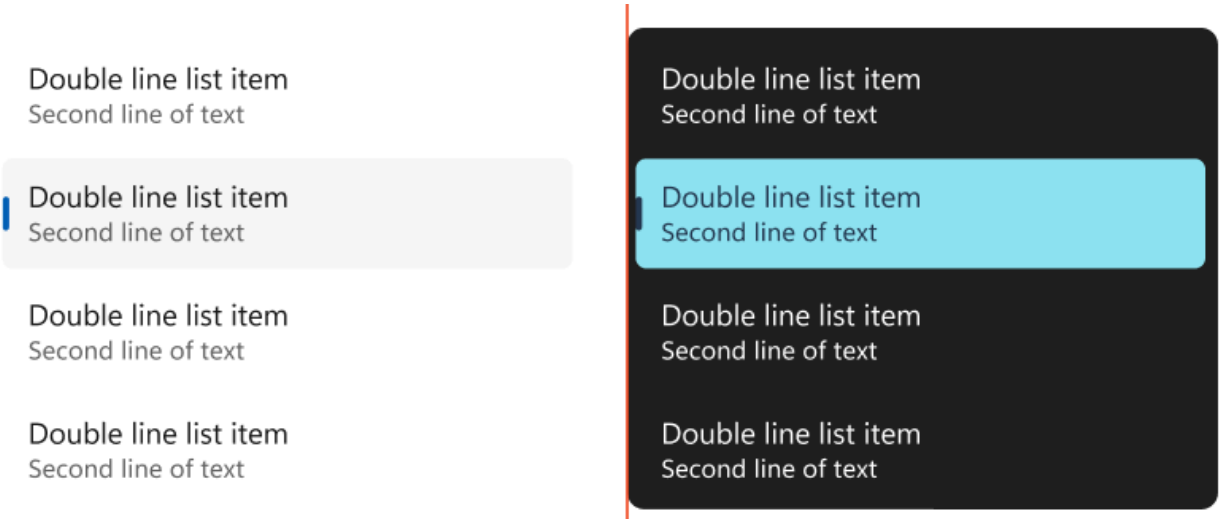
List items with colored text

In contrast themes, items in a ListView have their background set to **SystemColorHighlightColor** when the user hovers over, presses, or selects them. A common issue with complex list items occurs when the content of the list item fails to invert its color, making the items impossible to read.

Be careful when you set the TextBlock.Foreground in the DataTemplate of the **ListView** (typically done to establish visual hierarchy). The **Foreground** property is set on the **ListViewItem**, and each **TextBlock** in the DataTemplate inherits the correct **Foreground** color. Setting **Foreground** breaks this inheritance.



You can resolve this by setting **Foreground** conditionally through a **Style** in a ThemeDictionaries collection. As the **Foreground** is not set by **SecondaryBodyTextBlockStyle** in **HighContrast**, the color will invert correctly.



The following code snippet (from an App.xaml file) shows an example ThemeDictionaries collection in a **ListView** data template.

XAML

```
<ResourceDictionary.ThemeDictionaries>
    <ResourceDictionary x:Key="Default">
        <Style
            x:Key="SecondaryBodyTextBlockStyle"
            TargetType="TextBlock"
            BasedOn="{StaticResource BodyTextBlockStyle}">
            <Setter Property="Foreground"
                Value="{StaticResource
SystemControlForegroundBaseMediumBrush}" />
        </Style>
    </ResourceDictionary>

    <ResourceDictionary x:Key="Light">
        <Style
            x:Key="SecondaryBodyTextBlockStyle"
            TargetType="TextBlock"
            BasedOn="{StaticResource BodyTextBlockStyle}">
            <Setter Property="Foreground"
                Value="{StaticResource
SystemControlForegroundBaseMediumBrush}" />
        </Style>
    </ResourceDictionary>

    <ResourceDictionary x:Key="HighContrast">
        <!-- The Foreground Setter is omitted in HighContrast -->
        <Style
            x:Key="SecondaryBodyTextBlockStyle"
            TargetType="TextBlock"
            BasedOn="{StaticResource BodyTextBlockStyle}" />
```

```
    </ResourceDictionary>
</ResourceDictionary.ThemeDictionaries>

<!-- Usage in your DataTemplate... -->
<DataTemplate>
    <StackPanel>
        <TextBlock Style="{StaticResource BodyTextBlockStyle}" Text="Double
line list item" />

        <!-- Note how ThemeResource is used to reference the Style -->
        <TextBlock Style="{ThemeResource SecondaryBodyTextBlockStyle}"
Text="Second line of text" />
    </StackPanel>
</DataTemplate>
```

Examples

Open the WinUI 3 Gallery app and see color contrast accessibility support in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Related topics

- Accessibility
- UI contrast and settings sample
- XAML accessibility sample
- XAML high contrast sample
- **AccessibilitySettings**

Accessible text requirements

Article • 11/27/2024

This topic describes best practices for accessibility of text in an app, by assuring that colors and backgrounds satisfy the necessary contrast ratio. This topic also discusses the Microsoft UI Automation roles that text elements in a Universal Windows Platform (UWP) app can have, and best practices for text in graphics.

Contrast ratios

Although users always have the option to switch to a high-contrast mode, your app design for text should regard that option as a last resort. A much better practice is to make sure that your app text meets certain established guidelines for the level of contrast between text and its background. Evaluation of the level of contrast is based on deterministic techniques that do not consider color hue. For example, if you have red text on a green background, that text might not be readable to someone with a color blindness impairment. Checking and correcting the contrast ratio can prevent these types of accessibility issues.

The recommendations for text contrast documented here are based on a web accessibility standard, G18: Ensuring that a contrast ratio of at least 4.5:1 exists between text (and images of text) and background behind the text. This guidance exists in the *W3C Techniques for WCAG 2.0* specification.

To be considered accessible, visible text must have a minimum luminosity contrast ratio of 4.5:1 against the background. Exceptions include logos and incidental text, such as text that is part of an inactive UI component.

Text that is decorative and conveys no information is excluded. For example, if random words are used to create a background, and the words can be rearranged or substituted without changing meaning, the words are considered to be decorative and do not need to meet this criterion.

Use color contrast tools to verify that the visible text contrast ratio is acceptable. See Techniques for WCAG 2.0 G18 (Resources section) for tools that can test contrast ratios.

> **Note**
>
> Some of the tools listed by Techniques for WCAG 2.0 G18 can't be used interactively with a UWP app. You may need to enter foreground and background

color values manually in the tool, or make screen captures of app UI and then run the contrast ratio tool over the screen capture image.

Text element roles

A UWP app can use these default elements (commonly called *text elements* or *textedit controls*):

- **TextBlock**: role is **Text**
- **TextBox**: role is **Edit**
- **RichTextBlock** (and overflow class **RichTextBlockOverflow**): role is **Text**
- **RichEditBox**: role is **Edit**

When a control reports that is has a role of **Edit**, assistive technologies assume that there are ways for users to change the values. So if you put static text in a **TextBox**, you are misreporting the role and thus misreporting the structure of your app to the accessibility user.

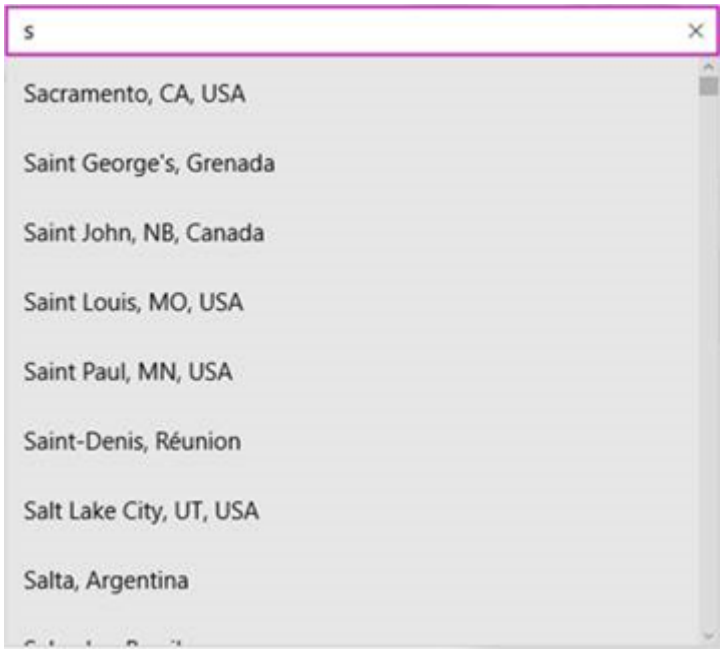
In the text models for XAML, there are two elements that are primarily used for static text, **TextBlock** and **RichTextBlock**. Neither of these are a **Control** subclass, and as such neither of them are keyboard-focusable or can appear in the tab order. But that does not mean that assistive technologies can't or won't read them. Screen readers are typically designed to support multiple modes of reading the content in an app, including a dedicated reading mode or navigation patterns that go beyond focus and the tab order, like a "virtual cursor". So don't put your static text into focusable containers just so that tab order gets the user there. Assistive technology users expect that anything in the tab order is interactive, and if they encounter static text there, that is more confusing than helpful. You should test this out yourself with Narrator to get a sense of the user experience with your app when using a screen reader to examine your app's static text.

Auto-suggest accessibility

When a user types into an entry field and a list of potential suggestions appears, this type of scenario is called auto-suggest. This is common in the **To:** line of a mail field, the Cortana search box in Windows, the URL entry field in Microsoft Edge, the location entry field in the Weather app, and so on. If you are using a XAML **AutosuggestBox** or the HTML intrinsic controls, then this experience is already hooked up for you by default. To make this experience accessible the entry field and the list must be associated. This is explained in the Implementing auto-suggest section.

Narrator has been updated to make this type of experience accessible with a special suggestions mode. At a high level, when the edit field and list are connected properly the end user will:

- Know the list is present and when the list closes
- Know how many suggestions are available
- Know the selected item, if any
- Be able to move Narrator focus to the list
- Be able to navigate through a suggestion with all other reading modes



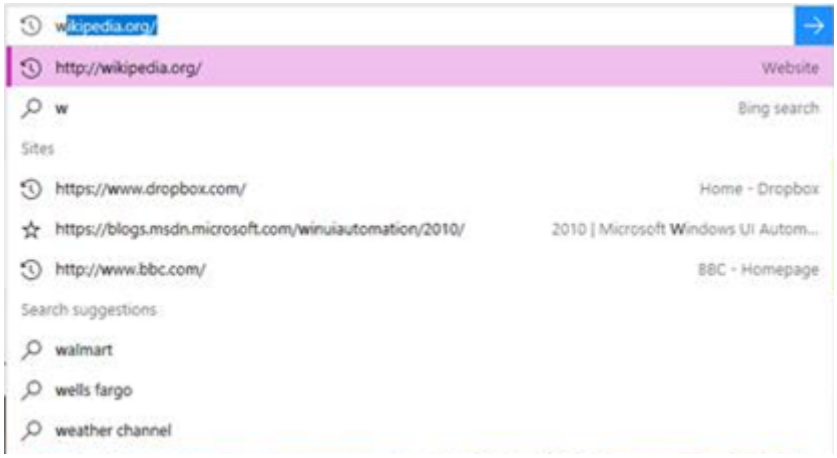
Example of a suggestion list

Implementing auto-suggest

To make this experience accessible the entry field and the list must be associated in the UIA tree. This association is done with the UIA_ControllerForPropertyId property in desktop apps or the ControlledPeers property in UWP apps.

At a high level there are 2 types of auto-suggest experiences.

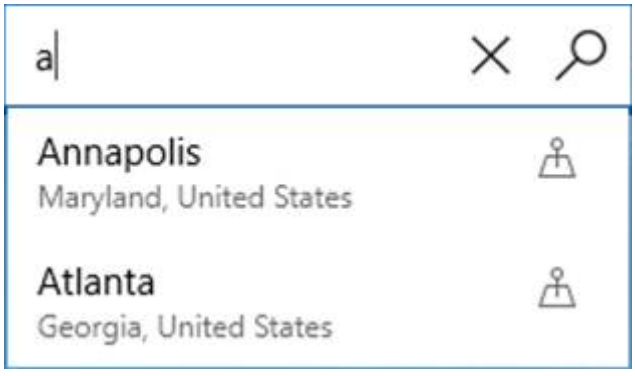
Default selection
If a default selection is made in the list, Narrator looks for a **UIA_SelectionItem_ElementSelectedEventId** event in a desktop app, or the **AutomationEvents.SelectionItemPatternOnElementSelected** event to be fired in a UWP app. Every time the selection changes, when the user types another letter and the suggestions have been updated or when a user navigates through the list, the **ElementSelected** event should be fired.



Example where there is a default selection

No default selection

If there is no default selection, such as in the Weather app's location box, then Narrator looks for the desktop **UIA_LayoutInvalidatedEventId** event or the UWP **LayoutInvalidated** event to be fired on the list every time the list is updated.



Example where there is no default selection

XAML implementation

If you are using the default XAML **AutosuggestBox**, then everything is already hooked up for you. If you are making your own auto-suggest experience using a **TextBox** and a list then you will need to set the list as **AutomationProperties.ControlledPeers** on the **TextBox**. You must fire the **AutomationPropertyChanged** event for the **ControlledPeers** property every time you add or remove this property and also fire your own **SelectionItemPatternOnElementSelected** events or **LayoutInvalidated** events depending on your type of scenario, which was explained previously in this article.

HTML implementation

If you are using the intrinsic controls in HTML, then the UIA implementation has already been mapped for you. Below is an example of an implementation that is already hooked up for you:

HTML

```
<label>Sites <input id="input1" type="text" list="datalist1" /></label>
<datalist id="datalist1">
        <option value="http://www.google.com/" label="Google"></option>
        <option value="http://www.reddit.com/" label="Reddit"></option>
</datalist>
```

If you are creating your own controls, you must set up your own ARIA controls, which are explained in the W3C standards.

Text in graphics

Whenever possible, avoid including text in a graphic. For example, any text that you include in the image source file that is displayed in the app as an **Image** element is not automatically accessible or readable by assistive technologies. If you must use text in graphics, make sure that the **AutomationProperties.Name** value that you provide as the equivalent of "alt text" includes that text or a summary of the text's meaning. Similar considerations apply if you are creating text characters from vectors as part of a **Path**, or by using **Glyphs**.

Text font size and scale

Users can have difficulty reading text in an app when the fonts uses are simply too small, so make sure any text in your application is a reasonable size in the first place.

Once you've done the obvious, Windows includes various accessibility tools and settings that users can take advantage of and adjust to their own needs and preferences for reading text. These include:

- The Magnifier tool, which enlarges a selected area of the UI. You should ensure the layout of text in your app doesn't make it difficult to use Magnifier for reading.
- Global scale and resolution settings in **Settings->System->Display->Scale and layout**. Exactly which sizing options are available can vary as this depends on the capabilities of the display device.
- Text size settings in **Settings->Ease of access->Display**. Adjust the **Make text bigger** setting to specify only the size of text in supporting controls across all applications and screens (all UWP text controls support the text scaling experience without any customization or templating).

> ⓘ Note

The **Make everything bigger** setting lets a user specify their preferred size for text and apps in general on their primary screen only.

Various text elements and controls have an **IsTextScaleFactorEnabled** property. This property has the value **true** by default. When **true**, the size of text in that element can be scaled. The scaling affects text that has a small **FontSize** to a greater degree than it affects text that has a large **FontSize**. You can disable automatic resizing by setting an element's **IsTextScaleFactorEnabled** property to **false**.

See Text scaling for more details.

Add the following markup to an app and run it. Adjust the **Text size** setting, and see what happens to each **TextBlock**.

XAML

```xml
<TextBlock Text="In this case, IsTextScaleFactorEnabled has been left set to its default value of true."
    Style="{StaticResource BodyTextBlockStyle}"/>

<TextBlock Text="In this case, IsTextScaleFactorEnabled has been set to false."
    Style="{StaticResource BodyTextBlockStyle}"
IsTextScaleFactorEnabled="False"/>
```

We don't recommend that you disable text scaling as scaling UI text universally across all apps is an important accessibility experience for users.

You can also use the **TextScaleFactorChanged** event and the **TextScaleFactor** property to find out about changes to the **Text size** setting on the phone. Here's how:

C#

```
{
    ...
    var uiSettings = new Windows.UI.ViewManagement.UISettings();
    uiSettings.TextScaleFactorChanged += UISettings_TextScaleFactorChanged;
    ...
}

private async void
UISettings_TextScaleFactorChanged(Windows.UI.ViewManagement.UISettings
sender, object args)
{
```

```
    var messageDialog = new
Windows.UI.Popups.MessageDialog(string.Format("It's now {0}",
sender.TextScaleFactor), "The text scale factor has changed");
    await messageDialog.ShowAsync();
}
```

The value of **TextScaleFactor** is a double in the range [1,2.25]. The smallest text is scaled up by this amount. You might be able to use the value to, say, scale graphics to match the text. But remember that not all text is scaled by the same factor. Generally speaking, the larger text is to begin with, the less it's affected by scaling.

These types have an **IsTextScaleFactorEnabled** property:

- **ContentPresenter**
- **Control** and derived classes
- **FontIcon**
- **RichTextBlock**
- **TextBlock**
- **TextElement** and derived classes

Examples

Open the WinUI 3 Gallery app and see text accessibility support in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Related topics

- Text scaling
- Accessibility
- Basic accessibility information
- XAML text display sample
- XAML text editing sample
- XAML accessibility sample

Accessibility practices to avoid

Article • 01/04/2023

If you want to create an accessible Windows app, see this list of practices to avoid:

- **Avoid building custom UI elements if you can use the default Windows controls** or controls that have already implemented Microsoft UI Automation support. Standard Windows controls are accessible by default and usually require adding only a few accessibility attributes that are app-specific. In contrast, implementing the **AutomationPeer** support for a true custom control is somewhat more involved (see Custom automation peers).

- **Don't put static text or other non-interactive elements into the tab order** (for example, by setting the **TabIndex** property for an element that is not interactive). If non-interactive elements are in the tab order, that is against keyboard accessibility guidelines because it decreases efficiency of keyboard navigation for users. Many assistive technologies use tab order and the ability to focus an element as part of their logic for how to present an app's interface to the assistive technology user. Text-only elements in the tab order can confuse users who expect only interactive elements in the tab order (buttons, check boxes, text input fields, combo boxes, lists, and so on).

- **Avoid using absolute positioning of UI elements** (such as in a **Canvas** element) because the presentation order often differs from the child element declaration order (which is the de facto logical order). Whenever possible, arrange UI elements in document or logical order to ensure that screen readers can read those elements in the correct order. If the visible order of UI elements can diverge from the document or logical order, use explicit tab index values (set **TabIndex**) to define the correct reading order.

- **Don’t use color as the only way to convey information.** Users who are color blind cannot receive information that is conveyed only through color, such as in a color status indicator. Include other visual cues, preferably text, to ensure that information is accessible.

- **Don’t automatically refresh an entire app canvas** unless it is really necessary for app functionality. If you need to automatically refresh page content, update only certain areas of the page. Assistive technologies generally must assume that a refreshed app canvas is a totally new structure, even if the effective changes were minimal. The cost of this to the assistive technology user is that any document

view or description of the refreshed app now must be recreated and presented to the user again.

A deliberate page navigation that is initiated by the user is a legitimate case for refreshing the app's structure. But make sure that the UI item that initiates the navigation is correctly identified or named to give some indication that invoking it will result in a context change and page reload.

> ⓘ **Note**
>
> If you do refresh content within a region, consider setting the **AccessibilityProperties.LiveSetting** accessibility property on that element to one of the non-default settings **Polite** or **Assertive**. Some assistive technologies can map this setting to the Accessible Rich Internet Applications (ARIA) concept of live regions and can thus inform the user that a region of content has changed.

- **Don’t use UI elements that flash more than three times per second.** Flashing elements can cause some people to have seizures. It is best to avoid using UI elements that flash.
- **Don’t change user context or activate functionality automatically.** Context or activation changes should occur only when the user takes a direct action on a UI element that has focus. Changes in user context include changing focus, displaying new content, and navigating to a different page. Making context changes without involving the user can be disorienting for users who have disabilities. The exceptions to this requirement include displaying submenus, validating forms, displaying help text in another control, and changing context in response to an asynchronous event.

Related topics

- Accessibility
- Accessibility in the Store
- Accessibility checklist

Custom automation peers

Article • 01/04/2023

Describes the concept of automation peers for Microsoft UI Automation, and how you can provide automation support for your own custom UI class.

UI Automation provides a framework that automation clients can use to examine or operate the user interfaces of a variety of UI platforms and frameworks. If you are writing a Windows app, the classes that you use for your UI already provide UI Automation support. You can derive from existing, non-sealed classes to define a new kind of UI control or support class. In the process of doing so, your class might add behavior that should have accessibility support but that the default UI Automation support does not cover. In this case, you should extend the existing UI Automation support by deriving from the **AutomationPeer** class that the base implementation used, adding any necessary support to your peer implementation, and informing the Windows app control infrastructure that it should create your new peer.

UI Automation enables not only accessibility applications and assistive technologies, such as screen readers, but also quality-assurance (test) code. In either scenario, UI Automation clients can examine user-interface elements and simulate user interaction with your app from other code outside your app. For info about UI Automation across all platforms and in its wider meaning, see UI Automation Overview.

There are two distinct audiences who use the UI Automation framework.

- **UI Automation *clients*** call UI Automation APIs to learn about all of the UI that is currently displayed to the user. For example, an assistive technology such as a screen reader acts as a UI Automation client. The UI is presented as a tree of automation elements that are related. The UI Automation client might be interested in just one app at a time, or in the entire tree. The UI Automation client can use UI Automation APIs to navigate the tree and to read or change information in the automation elements.
- **UI Automation *providers*** contribute information to the UI Automation tree, by implementing APIs that expose the elements in the UI that they introduced as part of their app. When you create a new control, you should now act as a participant in the UI Automation provider scenario. As a provider, you should ensure that all UI Automation clients can use the UI Automation framework to interact with your control for both accessibility and testing purposes.

Typically there are parallel APIs in the UI Automation framework: one API for UI Automation clients and another, similarly named API for UI Automation providers. For

the most part, this topic covers the APIs for the UI Automation provider, and specifically the classes and interfaces that enable provider extensibility in that UI framework. Occasionally we mention UI Automation APIs that the UI Automation clients use, to provide some perspective, or provide a lookup table that correlates the client and provider APIs. For more info about the client perspective, see UI Automation Client Programmer's Guide.

> **Note**
>
> UI Automation clients don't typically use managed code and aren't typically implemented as a UWP app (they are usually desktop apps). UI Automation is based on a standard and not a specific implementation or framework. Many existing UI Automation clients, including assistive technology products such as screen readers, use Component Object Model (COM) interfaces to interact with UI Automation, the system, and the apps that run in child windows. For more info on the COM interfaces and how to write a UI Automation client using COM, see UI Automation Fundamentals.

Determining the existing state of UI Automation support for your custom UI class

Before you attempt to implement an automation peer for a custom control, you should test whether the base class and its automation peer already provides the accessibility or automation support that you need. In many cases, the combination of the **FrameworkElementAutomationPeer** implementations, specific peers, and the patterns they implement can provide a basic but satisfactory accessibility experience. Whether this is true depends on how many changes you made to the object model exposure to your control versus its base class. Also, this depends on whether your additions to base class functionality correlate to new UI elements in the template contract or to the visual appearance of the control. In some cases your changes might introduce new aspects of user experience that require additional accessibility support.

Even if using the existing base peer class provides the basic accessibility support, it is still a best practice to define a peer so that you can report precise **ClassName** information to UI Automation for automated testing scenarios. This consideration is especially important if you are writing a control that is intended for third-party consumption.

Automation peer classes

The UWP builds on existing UI Automation techniques and conventions used by previous managed-code UI frameworks such as Windows Forms, Windows Presentation Foundation (WPF) and Microsoft Silverlight. Many of the control classes and their function and purpose also have their origin in a previous UI framework.

By convention, peer class names begin with the control class name and end with "AutomationPeer". For example, **ButtonAutomationPeer** is the peer class for the **Button** control class.

> **Note**
>
> For purposes of this topic, we treat the properties that are related to accessibility as being more important when you implement a control peer. But for a more general concept of UI Automation support, you should implement a peer in accordance with recommendations as documented by the UI Automation Provider Programmer's Guide and UI Automation Fundamentals. Those topics don't cover the specific AutomationPeer APIs that you would use to provide the information in the UWP framework for UI Automation, but they do describe the properties that identify your class or provide other information or interaction.

Peers, patterns and control types

A *control pattern* is an interface implementation that exposes a particular aspect of a control's functionality to a UI Automation client. UI Automation clients use the properties and methods exposed through a control pattern to retrieve information about capabilities of the control, or to manipulate the control's behavior at run time.

Control patterns provide a way to categorize and expose a control's functionality independent of the control type or the appearance of the control. For example, a control that presents a tabular interface uses the **Grid** control pattern to expose the number of rows and columns in the table, and to enable a UI Automation client to retrieve items from the table. As other examples, the UI Automation client can use the **Invoke** control pattern for controls that can be invoked, such as buttons, and the **Scroll** control pattern for controls that have scroll bars, such as list boxes, list views, or combo boxes. Each control pattern represents a separate type of functionality, and control patterns can be combined to describe the full set of functionality supported by a particular control.

Control patterns relate to UI as interfaces relate to COM objects. In COM, you can query an object to ask what interfaces it supports and then use those interfaces to access functionality. In UI Automation, UI Automation clients can query a UI Automation element to find out which control patterns it supports, and then interact with the

element and its peered control through the properties, methods, events, and structures exposed by the supported control patterns.

One of the main purposes of an automation peer is to report to a UI Automation client which control patterns the UI element can support through its peer. To do this, UI Automation providers implement new peers that change the **GetPattern** method behavior by overriding the **GetPatternCore** method. UI Automation clients make calls that the UI Automation provider maps to calling **GetPattern**. UI Automation clients query for each specific pattern that they want to interact with. If the peer supports the pattern, it returns an object reference to itself; otherwise it returns **null**. If the return is not **null**, the UI Automation client expects that it can call APIs of the pattern interface as a client, in order to interact with that control pattern.

A *control type* is a way to broadly define the functionality of a control that the peer represents. This is a different concept than a control pattern because while a pattern informs UI Automation what info it can get or what actions it can perform through a particular interface, the control type exists one level above that. Each control type has guidance about these aspects of UI Automation:

- UI Automation control patterns: A control type might support more than one pattern, each of which represents a different classification of info or interaction. Each control type has a set of control patterns that the control must support, a set that is optional, and a set that the control must not support.
- UI Automation property values: Each control type has a set of properties that the control must support. These are the general properties, as described in UI Automation Properties Overview, not the ones that are pattern-specific.
- UI Automation events: Each control type has a set of events that the control must support. Again these are general, not pattern-specific, as described in UI Automation Events Overview.
- UI Automation tree structure: Each control type defines how the control must appear in the UI Automation tree structure.

Regardless of how automation peers for the framework are implemented, UI Automation client functionality isn't tied to the UWP, and in fact it's likely that existing UI Automation clients such as assistive technologies will use other programming models, such as COM. In COM, clients can **QueryInterface** for the COM control pattern interface that implements the requested pattern or the general UI Automation framework for properties, events or tree examination. For the patterns, the UI Automation framework marshals that interface code across into UWP code running against the app's UI Automation provider and the relevant peer.

When you implement control patterns for a managed-code framework such as a UWP app using C# or Microsoft Visual Basic, you can use .NET Framework interfaces to represent these patterns instead of using the COM interface representation. For example, the UI Automation pattern interface for a Microsoft .NET provider implementation of the **Invoke** pattern is **IInvokeProvider**.

For a list of control patterns, provider interfaces, and their purpose, see Control patterns and interfaces. For the list of the control types, see UI Automation Control Types Overview.

Guidance for how to implement control patterns

The control patterns and what they're intended for are part of a larger definition of the UI Automation framework, and don't just apply to the accessibility support for a UWP app. When you implement a control pattern you should make sure you're implementing it in a way that matches the guidance as documented in these docs and also in the UI Automation specification. If you're looking for guidance, you can generally use the Microsoft documentation and won't need to refer to the specification. Guidance for each pattern is documented here: Implementing UI Automation Control Patterns. You'll notice that each topic under this area has an "Implementation Guidelines and Conventions" section and "Required Members" section. The guidance usually refers to specific APIs of the relevant control pattern interface in the Control Pattern Interfaces for Providers reference. Those interfaces are the native/COM interfaces (and their APIs use COM-style syntax). But everything you see there has an equivalent in the **Windows.UI.Xaml.Automation.Provider** namespace.

If you're using the default automation peers and expanding on their behavior, those peers have already been written in conformance to UI Automation guidelines. If they support control patterns, you can rely on that pattern support conforming with guidance at Implementing UI Automation Control Patterns. If a control peer reports that it's representative of a control type defined by UI Automation, then the guidance documented at Supporting UI Automation Control Types has been followed by that peer.

Nevertheless you might need additional guidance for control patterns or control types in order to follow the UI Automation recommendations in your peer implementation. That would be particularly true if you're implementing pattern or control type support that doesn't yet exist as a default implementation in a UWP control. For example, the pattern for annotations isn't implemented in any of the default XAML controls. But you might have an app that uses annotations extensively and therefore you want to surface that functionality to be accessible. For this scenario, your peer should implement

IAnnotationProvider and should probably report itself as the **Document** control type with appropriate properties to indicate that your documents support annotation.

We recommend that you use the guidance that you see for the patterns under Implementing UI Automation Control Patterns or control types under Supporting UI Automation Control Types as orientation and general guidance. You might even try following some of the API links for descriptions and remarks as to the purpose of the APIs. But for syntax specifics that are needed for UWP app programming, find the equivalent API within the **Windows.UI.Xaml.Automation.Provider** namespace and use those reference pages for more info.

Built-in automation peer classes

In general, elements implement an automation peer class if they accept UI activity from the user, or if they contain information needed by users of assistive technologies that represent the interactive or meaningful UI of apps. Not all UWP visual elements have automation peers. Examples of classes that implement automation peers are **Button** and **TextBox**. Examples of classes that do not implement automation peers are **Border** and classes based on **Panel**, such as **Grid** and **Canvas**. A **Panel** has no peer because it is providing a layout behavior that is visual only. There is no accessibility-relevant way for the user to interact with a **Panel**. Whatever child elements a **Panel** contains are instead reported to UI Automation trees as child elements of the next available parent in the tree that has a peer or element representation.

UI Automation and UWP process boundaries

Typically, UI Automation client code that accesses a UWP app runs out-of-process. The UI Automation framework infrastructure enables information to get across the process boundary. This concept is explained in more detail in UI Automation Fundamentals.

OnCreateAutomationPeer

All classes that derive from **UIElement** contain the protected virtual method **OnCreateAutomationPeer**. The object initialization sequence for automation peers calls **OnCreateAutomationPeer** to get the automation peer object for each control and thus to construct a UI Automation tree for run-time use. UI Automation code can use the peer to get information about a control's characteristics and features and to simulate interactive use by means of its control patterns. A custom control that supports automation must override **OnCreateAutomationPeer** and return an instance of a class that derives from **AutomationPeer**. For example, if a custom control derives from the

ButtonBase class, the object returned by **OnCreateAutomationPeer** should derive from ButtonBaseAutomationPeer.

If you're writing a custom control class and intend to also supply a new automation peer, you should override the OnCreateAutomationPeer method for your custom control so that it returns a new instance of your peer. Your peer class must derive directly or indirectly from AutomationPeer.

For example, the following code declares that the custom control `NumericUpDown` should use the peer `NumericUpDownPeer` for UI Automation purposes.

C#

```
using Windows.UI.Xaml.Automation.Peers;
...
public class NumericUpDown : RangeBase {
    public NumericUpDown() {
    // other initialization; DefaultStyleKey etc.
    }
    ...
    protected override AutomationPeer OnCreateAutomationPeer()
    {
        return new NumericUpDownAutomationPeer(this);
    }
}
```

VB

```
Public Class NumericUpDown
    Inherits RangeBase
    ' other initialization; DefaultStyleKey etc.
       Public Sub New()
       End Sub
       Protected Overrides Function OnCreateAutomationPeer() As
AutomationPeer
              Return New NumericUpDownAutomationPeer(Me)
       End Function
End Class
```

C++/WinRT

```
// NumericUpDown.idl
namespace MyNamespace
{
    runtimeclass NumericUpDown :
Windows.UI.Xaml.Controls.Primitives.RangeBase
    {
        NumericUpDown();
        Int32 MyProperty;
```

```
    }
}

// NumericUpDown.h
...
struct NumericUpDown : NumericUpDownT<NumericUpDown>
{
    ...
    Windows::UI::Xaml::Automation::Peers::AutomationPeer
OnCreateAutomationPeer()
    {
        return
winrt::make<MyNamespace::implementation::NumericUpDownAutomationPeer>
(*this);
    }
};
```

C++

```
//.h
public ref class NumericUpDown sealed :
Windows::UI::Xaml::Controls::Primitives::RangeBase
{
// other initialization not shown
protected:
    virtual AutomationPeer^ OnCreateAutomationPeer() override
    {
         return ref new NumericUpDownAutomationPeer(this);
    }
};
```

> **Note**
>
> The OnCreateAutomationPeer implementation should do nothing more than initialize a new instance of your custom automation peer, passing the calling control as owner, and return that instance. Do not attempt additional logic in this method. In particular, any logic that could potentially lead to destruction of the AutomationPeer within the same call may result in unexpected runtime behavior.

In typical implementations of OnCreateAutomationPeer, the *owner* is specified as **this** or **Me** because the method override is in the same scope as the rest of the control class definition.

The actual peer class definition can be done in the same code file as the control or in a separate code file. The peer definitions all exist in the Windows.UI.Xaml.Automation.Peers namespace that is a separate namespace from the controls that they provide peers for. You can choose to declare your peers in separate

namespaces also, as long as you reference the necessary namespaces for the OnCreateAutomationPeer method call.

Choosing the correct peer base class

Make sure that your AutomationPeer is derived from a base class that gives you the best match for the existing peer logic of the control class you are deriving from. In the case of the previous example, because `NumericUpDown` derives from RangeBase, there is a RangeBaseAutomationPeer class available that you should base your peer on. By using the closest matching peer class in parallel to how you derive the control itself, you can avoid overriding at least some of the IRangeValueProvider functionality because the base peer class already implements it.

The base Control class does not have a corresponding peer class. If you need a peer class to correspond to a custom control that derives from **Control**, derive the custom peer class from FrameworkElementAutomationPeer.

If you derive from ContentControl directly, that class has no default automation peer behavior because there is no OnCreateAutomationPeer implementation that references a peer class. So make sure either to implement **OnCreateAutomationPeer** to use your own peer, or to use FrameworkElementAutomationPeer as the peer if that level of accessibility support is adequate for your control.

> ⓘ **Note**
>
> You don't typically derive from AutomationPeer rather than FrameworkElementAutomationPeer. If you did derive directly from **AutomationPeer** you'll need to duplicate a lot of basic accessibility support that would otherwise come from **FrameworkElementAutomationPeer**.

Initialization of a custom peer class

The automation peer should define a type-safe constructor that uses an instance of the owner control for base initialization. In the next example, the implementation passes the *owner* value on to the RangeBaseAutomationPeer base, and ultimately it is the FrameworkElementAutomationPeer that actually uses *owner* to set FrameworkElementAutomationPeer.Owner.

```
C#
```

```
public NumericUpDownAutomationPeer(NumericUpDown owner): base(owner)
{}
```

VB

```
Public Sub New(owner As NumericUpDown)
    MyBase.New(owner)
End Sub
```

C++/WinRT

```
// NumericUpDownAutomationPeer.idl
import "NumericUpDown.idl";
namespace MyNamespace
{
    runtimeclass NumericUpDownAutomationPeer :
Windows.UI.Xaml.Automation.Peers.AutomationPeer
    {
        NumericUpDownAutomationPeer(NumericUpDown owner);
        Int32 MyProperty;
    }
}

// NumericUpDownAutomationPeer.h
...
struct NumericUpDownAutomationPeer :
NumericUpDownAutomationPeerT<NumericUpDownAutomationPeer>
{
    ...
    NumericUpDownAutomationPeer(MyNamespace::NumericUpDown const& owner);
};
```

C++

```
//.h
public ref class NumericUpDownAutomationPeer sealed :
Windows::UI::Xaml::Automation::Peers::RangeBaseAutomationPeer
//.cpp
public:    NumericUpDownAutomationPeer(NumericUpDown^ owner);
```

Core methods of AutomationPeer

For UWP infrastructure reasons, the overridable methods of an automation peer are part of a pair of methods: the public access method that the UI Automation provider uses as a forwarding point for UI Automation clients, and the protected "Core" customization method that a UWP class can override to influence the behavior. The method pair is

wired together by default in such a way that the call to the access method always invokes the parallel "Core" method that has the provider implementation, or as a fallback, invokes a default implementation from the base classes.

When implementing a peer for a custom control, override any of the "Core" methods from the base automation peer class where you want to expose behavior that is unique to your custom control. UI Automation code gets information about your control by calling public methods of the peer class. To provide information about your control, override each method with a name that ends with "Core" when your control implementation and design creates accessibility scenarios or other UI Automation scenarios that differ from what's supported by the base automation peer class.

At a minimum, whenever you define a new peer class, implement the GetClassNameCore method, as shown in the next example.

C#

```
protected override string GetClassNameCore()
{
    return "NumericUpDown";
}
```

> ⓘ Note
>
> You might want to store the strings as constants rather than directly in the method body, but that is up to you. For GetClassNameCore, you won't need to localize this string. The **LocalizedControlType** property is used any time a localized string is needed by a UI Automation client, not **ClassName**.

GetAutomationControlType

Some assistive technologies use the GetAutomationControlType value directly when reporting characteristics of the items in a UI Automation tree, as additional information beyond the UI Automation **Name**. If your control is significantly different from the control you are deriving from and you want to report a different control type from what is reported by the base peer class used by the control, you must implement a peer and override GetAutomationControlTypeCore in your peer implementation. This is particularly important if you derive from a generalized base class such as ItemsControl or ContentControl, where the base peer doesn't provide precise information about control type.

Your implementation of GetAutomationControlTypeCore describes your control by returning an AutomationControlType value. Although you can return **AutomationControlType.Custom**, you should return one of the more specific control types if it accurately describes your control's main scenarios. Here's an example.

```
C#

protected override AutomationControlType GetAutomationControlTypeCore()
{
    return AutomationControlType.Spinner;
}
```

> ⓘ Note
>
> Unless you specify AutomationControlType.Custom, you don't have to implement GetLocalizedControlTypeCore to provide a **LocalizedControlType** property value to clients. UI Automation common infrastructure provides translated strings for every possible **AutomationControlType** value other than **AutomationControlType.Custom**.

GetPattern and GetPatternCore

A peer's implementation of GetPatternCore returns the object that supports the pattern that is requested in the input parameter. Specifically, a UI Automation client calls a method that is forwarded to the provider's GetPattern method, and specifies a PatternInterface enumeration value that names the requested pattern. Your override of **GetPatternCore** should return the object that implements the specified pattern. That object is the peer itself, because the peer should implement the corresponding pattern interface any time that it reports that it supports a pattern. If your peer does not have a custom implementation of a pattern, but you know that the peer's base does implement the pattern, you can call the base type's implementation of **GetPatternCore** from your **GetPatternCore**. A peer's **GetPatternCore** should return **null** if a pattern is not supported by the peer. However, instead of returning **null** directly from your implementation, you would usually rely on the call to the base implementation to return **null** for any unsupported pattern.

When a pattern is supported, the GetPatternCore implementation can return **this** or **Me**. The expectation is that the UI Automation client will cast the GetPattern return value to the requested pattern interface whenever it is not **null**.

If a peer class inherits from another peer, and all necessary support and pattern reporting is already handled by the base class, implementing **GetPatternCore** isn't necessary. For example, if you are implementing a range control that derives from **RangeBase**, and your peer derives from **RangeBaseAutomationPeer**, that peer returns itself for **PatternInterface.RangeValue** and has working implementations of the **IRangeValueProvider** interface that supports the pattern.

Although it is not the literal code, this example approximates the implementation of **GetPatternCore** already present in **RangeBaseAutomationPeer**.

C#

```
protected override object GetPatternCore(PatternInterface patternInterface)
{
    if (patternInterface == PatternInterface.RangeValue)
    {
        return this;
    }
    return base.GetPatternCore(patternInterface);
}
```

If you are implementing a peer where you don't have all the support you need from a base peer class, or you want to change or add to the set of base-inherited patterns that your peer can support, then you should override **GetPatternCore** to enable UI Automation clients to use the patterns.

For a list of the provider patterns that are available in the UWP implementation of UI Automation support, see **Windows.UI.Xaml.Automation.Provider**. Each such pattern has a corresponding value of the **PatternInterface** enumeration, which is how UI Automation clients request the pattern in a **GetPattern** call.

A peer can report that it supports more than one pattern. If so, the override should include return path logic for each supported **PatternInterface** value and return the peer in each matching case. It is expected that the caller will request only one interface at a time, and it is up to the caller to cast to the expected interface.

Here's an example of a **GetPatternCore** override for a custom peer. It reports the support for two patterns, **IRangeValueProvider** and **IToggleProvider**. The control here is a media display control that can display as full-screen (the toggle mode) and that has a progress bar within which users can select a position (the range control). This code came from the XAML accessibility sample.

C#

```
protected override object GetPatternCore(PatternInterface patternInterface)
{
    if (patternInterface == PatternInterface.RangeValue)
    {
        return this;
    }
    else if (patternInterface == PatternInterface.Toggle)
    {
        return this;
    }
    return null;
}
```

Forwarding patterns from sub-elements

A **GetPatternCore** method implementation can also specify a sub-element or part as a pattern provider for its host. This example mimics how **ItemsControl** transfers scroll-pattern handling to the peer of its internal **ScrollViewer** control. To specify a sub-element for pattern handling, this code gets the sub-element object, creates a peer for the sub-element by using the **FrameworkElementAutomationPeer.CreatePeerForElement** method, and returns the new peer.

C#

```
protected override object GetPatternCore(PatternInterface patternInterface)
{
    if (patternInterface == PatternInterface.Scroll)
    {
        ItemsControl owner = (ItemsControl) base.Owner;
        UIElement itemsHost = owner.ItemsHost;
        ScrollViewer element = null;
        while (itemsHost != owner)
        {
            itemsHost = VisualTreeHelper.GetParent(itemsHost) as UIElement;
            element = itemsHost as ScrollViewer;
            if (element != null)
            {
                break;
            }
        }
        if (element != null)
        {
            AutomationPeer peer =
FrameworkElementAutomationPeer.CreatePeerForElement(element);
            if ((peer != null) && (peer is IScrollProvider))
            {
                return (IScrollProvider) peer;
            }
```

```
            }
        }
        return base.GetPatternCore(patternInterface);
    }
```

Other Core methods

Your control may need to support keyboard equivalents for primary scenarios; for more info about why this might be necessary, see Keyboard accessibility. Implementing the key support is necessarily part of the control code and not the peer code because that is part of a control's logic, but your peer class should override the **GetAcceleratorKeyCore** and **GetAccessKeyCore** methods to report to UI Automation clients which keys are used. Consider that the strings that report key information might need to be localized, and should therefore come from resources, not hard-coded strings.

If you are providing a peer for a class that supports a collection, it's best to derive from both functional classes and peer classes that already have that kind of collection support. If you can't do so, peers for controls that maintain child collections may have to override the collection-related peer method **GetChildrenCore** to properly report the parent-child relationships to the UI Automation tree.

Implement the **IsContentElementCore** and **IsControlElementCore** methods to indicate whether your control contains data content or fulfills an interactive role in the user interface (or both). By default, both methods return **true**. These settings improve the usability of assistive technologies such as screen readers, which may use these methods to filter the automation tree. If your **GetPatternCore** method transfers pattern handling to a sub-element peer, the sub-element peer's **IsControlElementCore** method can return **false** to hide the sub-element peer from the automation tree.

Some controls may support labeling scenarios, where a text label part supplies information for a non-text part, or a control is intended to be in a known labeling relationship with another control in the UI. If it's possible to provide a useful class-based behavior, you can override **GetLabeledByCore** to provide this behavior.

GetBoundingRectangleCore and **GetClickablePointCore** are used mainly for automated testing scenarios. If you want to support automated testing for your control, you might want to override these methods. This might be desired for range-type controls, where you can't suggest just a single point because where the user clicks in coordinate space has a different effect on a range. For example, the default **ScrollBar** automation peer overrides **GetClickablePointCore** to return a "not a number" **Point** value.

GetLiveSettingCore influences the control default for the **LiveSetting** value for UI Automation. You might want to override this if you want your control to return a value

other than AutomationLiveSetting.Off. For more info on what **LiveSetting** represents, see AutomationProperties.LiveSetting.

You might override GetOrientationCore if your control has a settable orientation property that can map to AutomationOrientation. The ScrollBarAutomationPeer and SliderAutomationPeer classes do this.

Base implementation in FrameworkElementAutomationPeer

The base implementation of FrameworkElementAutomationPeer provides some UI Automation information that can be interpreted from various layout and behavior properties that are defined at the framework level.

- GetBoundingRectangleCore: Returns a Rect structure based on the known layout characteristics. Returns a 0-value **Rect** if IsOffscreen is **true**.
- GetClickablePointCore: Returns a Point structure based on the known layout characteristics, as long as there is a nonzero **BoundingRectangle**.
- GetNameCore: More extensive behavior than can be summarized here; see GetNameCore. Basically, it attempts a string conversion on any known content of a ContentControl or related classes that have content. Also, if there is a value for LabeledBy, that item's **Name** value is used as the **Name**.
- HasKeyboardFocusCore: Evaluated based on the owner's FocusState and IsEnabled properties. Elements that aren't controls always return **false**.
- IsEnabledCore: Evaluated based on the owner's IsEnabled property if it is a Control. Elements that aren't controls always return **true**. This doesn't mean that the owner is enabled in the conventional interaction sense; it means that the peer is enabled despite the owner not having an **IsEnabled** property.
- IsKeyboardFocusableCore: Returns **true** if owner is a Control; otherwise it is **false**.
- IsOffscreenCore: A Visibility of Collapsed on the owner element or any of its parents equates to a **true** value for IsOffscreen. Exception: a Popup object can be visible even if its owner's parents are not.
- SetFocusCore: Calls Focus.
- GetParent: Calls FrameworkElement.Parent from the owner, and looks up the appropriate peer. This isn't an override pair with a "Core" method, so you can't change this behavior.

> ⓘ **Note**
>
> Default UWP peers implement a behavior by using internal native code that implements the UWP, not necessarily by using actual UWP code. You won't be able

to see the code or logic of the implementation through common language runtime (CLR) reflection or other techniques. You also won't see distinct reference pages for subclass-specific overrides of base peer behavior. For example, there might be additional behavior for GetNameCore of a TextBoxAutomationPeer, which won't be described on the **AutomationPeer.GetNameCore** reference page, and there is no reference page for **TextBoxAutomationPeer.GetNameCore**. There isn't even a **TextBoxAutomationPeer.GetNameCore** reference page. Instead, read the reference topic for the most immediate peer class, and look for implementation notes in the Remarks section.

Peers and AutomationProperties

Your automation peer should provide appropriate default values for your control's accessibility-related information. Note that any app code that uses the control can override some of that behavior by including **AutomationProperties** attached-property values on control instances. Callers can do this either for the default controls or for custom controls. For example, the following XAML creates a button that has two customized UI Automation properties: `<Button AutomationProperties.Name="Special" AutomationProperties.HelpText="This is a special button."/>`

For more info about **AutomationProperties** attached properties, see Basic accessibility information.

Some of the **AutomationPeer** methods exist because of the general contract of how UI Automation providers are expected to report information, but these methods are not typically implemented in control peers. This is because that info is expected to be provided by **AutomationProperties** values applied to the app code that uses the controls in a specific UI. For example, most apps would define the labeling relationship between two different controls in the UI by applying a **AutomationProperties.LabeledBy** value. However, **LabeledByCore** is implemented in certain peers that represent data or item relationships in a control, such as using a header part to label a data-field part, labeling items with their containers, or similar scenarios.

Implementing patterns

Let's look at how to write a peer for a control that implements an expand-collapse behavior by implementing the control pattern interface for expand-collapse. The peer should enable the accessibility for the expand-collapse behavior by returning itself whenever **GetPattern** is called with a value of **PatternInterface.ExpandCollapse**. The

peer should then inherit the provider interface for that pattern (**IExpandCollapseProvider**) and provide implementations for each of the members of that provider interface. In this case the interface has three members to override: **Expand**, **Collapse**, **ExpandCollapseState**.

It's helpful to plan ahead for accessibility in the API design of the class itself. Whenever you have a behavior that is potentially requested either as a result of typical interactions with a user who is working in the UI or through an automation provider pattern, provide a single method that either the UI response or the automation pattern can call. For example, if your control has button parts that have wired event handlers that can expand or collapse the control, and has keyboard equivalents for those actions, have these event handlers call the same method that you call from within the body of the **Expand** or **Collapse** implementations for **IExpandCollapseProvider** in the peer. Using a common logic method can also be a useful way to make sure that your control's visual states are updated to show logical state in a uniform way, regardless of how the behavior was invoked.

A typical implementation is that the provider APIs first call **Owner** for access to the control instance at run time. Then the necessary behavior methods can be called on that object.

C#

```
public class IndexCardAutomationPeer : FrameworkElementAutomationPeer,
IExpandCollapseProvider {
    private IndexCard ownerIndexCard;
    public IndexCardAutomationPeer(IndexCard owner) : base(owner)
    {
         ownerIndexCard = owner;
    }
}
```

An alternate implementation is that the control itself can reference its peer. This is a common pattern if you are raising automation events from the control, because the **RaiseAutomationEvent** method is a peer method.

UI Automation events

UI Automation events fall into the following categories.

Expand table

Event	Description
Property change	Fires when a property on a UI Automation element or control pattern changes. For example, if a client needs to monitor an app's check box control, it can register to listen for a property change event on the ToggleState property. When the check box control is checked or unchecked, the provider fires the event and the client can act as necessary.
Element action	Fires when a change in the UI results from user or programmatic activity; for example, when a button is clicked or invoked through the **Invoke** pattern.
Structure change	Fires when the structure of the UI Automation tree changes. The structure changes when new UI items become visible, hidden, or removed on the desktop.
Global change	Fires when actions of global interest to the client occur, such as when the focus shifts from one element to another, or when a child window closes. Some events do not necessarily mean that the state of the UI has changed. For example, if the user tabs to a text-entry field and then clicks a button to update the field, a TextChanged event fires even if the user did not actually change the text. When processing an event, it may be necessary for a client application to check whether anything has actually changed before taking action.

AutomationEvents identifiers

UI Automation events are identified by AutomationEvents values. The values of the enumeration uniquely identify the kind of event.

Raising events

UI Automation clients can subscribe to automation events. In the automation peer model, peers for custom controls must report changes to control state that are relevant to accessibility by calling the RaiseAutomationEvent method. Similarly, when a key UI Automation property value changes, custom control peers should call the RaisePropertyChangedEvent method.

The next code example shows how to get the peer object from within the control definition code and call a method to fire an event from that peer. As an optimization, the code determines whether there are any listeners for this event type. Firing the event and creating the peer object only when there are listeners avoids unnecessary overhead and helps the control remain responsive.

C#

```
if (AutomationPeer.ListenerExists(AutomationEvents.PropertyChanged))
{
    NumericUpDownAutomationPeer peer =
```

```
        FrameworkElementAutomationPeer.FromElement(nudCtrl) as
NumericUpDownAutomationPeer;
    if (peer != null)
    {
        peer.RaisePropertyChangedEvent(
            RangeValuePatternIdentifiers.ValueProperty,
            (double)oldValue,
            (double)newValue);
    }
}
```

Peer navigation

After locating an automation peer, a UI Automation client can navigate the peer structure of an app by calling the peer object's **GetChildren** and **GetParent** methods. Navigation among UI elements within a control is supported by the peer's implementation of the **GetChildrenCore** method. The UI Automation system calls this method to build up a tree of sub-elements contained within a control; for example, list items in a list box. The default **GetChildrenCore** method in **FrameworkElementAutomationPeer** traverses the visual tree of elements to build the tree of automation peers. Custom controls can override this method to expose a different representation of child elements to automation clients, returning the automation peers of elements that convey information or allow user interaction.

Native automation support for text patterns

Some of the default UWP app automation peers provide control pattern support for the text pattern (**PatternInterface.Text**). But they provide this support through native methods, and the peers involved won't note the **ITextProvider** interface in the (managed) inheritance. Still, if a managed or non-managed UI Automation client queries the peer for patterns, it will report support for the text pattern, and provide behavior for parts of the pattern when client APIs are called.

If you intend to derive from one of the UWP app text controls and also create a custom peer that derives from one of the text-related peers, check the Remarks sections for the peer to learn more about any native-level support for patterns. You can access the native base behavior in your custom peer if you call the base implementation from your managed provider interface implementations, but it's difficult to modify what the base implementation does because the native interfaces on both the peer and its owner control aren't exposed. Generally you should either use the base implementations as-is (call base only) or completely replace the functionality with your own managed code and don't call the base implementation. The latter is an advanced scenario, you'll need

good familiarity with the text services framework being used by your control in order to support the accessibility requirements when using that framework.

AutomationProperties.AccessibilityView

In addition to providing a custom peer, you can also adjust the tree view representation for any control instance, by setting **AutomationProperties.AccessibilityView** in XAML. This isn't implemented as part of a peer class, but we'll mention it here because it's germane to overall accessibility support either for custom controls or for templates you customize.

The main scenario for using **AutomationProperties.AccessibilityView** is to deliberately omit certain controls in a template from the UI Automation views, because they don't meaningfully contribute to the accessibility view of the entire control. To prevent this, set **AutomationProperties.AccessibilityView** to "Raw".

Throwing exceptions from automation peers

The APIs that you are implementing for your automation peer support are permitted to throw exceptions. It's expected any UI Automation clients that are listening are robust enough to continue on after most exceptions are thrown. In all likelihood that listener is looking at an all-up automation tree that includes apps other than your own, and it's an unacceptable client design to bring down the entire client just because one area of the tree threw a peer-based exception when the client called its APIs.

For parameters that are passed in to your peer, it's acceptable to validate the input, and for example throw **ArgumentNullException** if it was passed **null** and that's not a valid value for your implementation. However, if there are subsequent operations performed by your peer, remember that the peer's interactions with the hosting control have something of an asynchronous character to them. Anything a peer does won't necessarily block the UI thread in the control (and it probably shouldn't). So you could have situations where an object was available or had certain properties when the peer was created or when an automation peer method was first called, but in the meantime the control state has changed. For these cases, there are two dedicated exceptions that a provider can throw:

- Throw **ElementNotAvailableException** if you're unable to access either the peer's owner or a related peer element based on the original info your API was passed. For example, you might have a peer that's trying to run its methods but the owner has since been removed from the UI, such as a modal dialog that's been closed. For a non-.NET client, this maps to **UIA_E_ELEMENTNOTAVAILABLE**.

- Throw **ElementNotEnabledException** if there still is an owner, but that owner is in a mode such as **IsEnabled** = **false** that's blocking some of the specific programmatic changes that your peer is trying to accomplish. For a non-.NET client, this maps to **UIA_E_ELEMENTNOTENABLED**.

Beyond this, peers should be relatively conservative regarding exceptions that they throw from their peer support. Most clients won't be able to handle exceptions from peers and turn these into actionable choices that their users can make when interacting with the client. So sometimes a no-op, and catching exceptions without rethrowing within your peer implementations, is a better strategy than is throwing exceptions every time something the peer tries to do doesn't work. Consider also that most UI Automation clients aren't written in managed code. Most are written in COM and are just checking for **S_OK** in an **HRESULT** whenever they call a UI Automation client method that ends up accessing your peer.

Related topics

- Accessibility
- XAML accessibility sample
- **FrameworkElementAutomationPeer**
- **AutomationPeer**
- **OnCreateAutomationPeer**
- Control patterns and interfaces

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Control patterns and interfaces

Article • 01/04/2023

Lists the Microsoft UI Automation control patterns, the classes that clients use to access them, and the interfaces providers use to implement them.

The table in this topic describes the Microsoft UI Automation control patterns. The table also lists the classes used by UI Automation clients to access the control patterns and the interfaces used by UI Automation providers to implement them. The **Control pattern** column shows the pattern name from the UI Automation client perspective, as a constant value listed in **Control Pattern Availability Property Identifiers**. From the UI Automation provider perspective, each of these patterns is a **PatternInterface** constant name. The **Class provider interface** column shows the name of the interface that providers implement to provide this pattern for a custom XAML control.

For more info about how to implement custom automation peers that expose control patterns and implement the interfaces, see Custom automation peers.

When you implement a control pattern, you should also consult the UI Automation provider documentation that explains some of the expectations that clients will have of a control pattern regardless of which UI framework is used to implement it. Some of the info listed in the general UI Automation provider documentation will influence how you implement your peers and correctly support that pattern. See Implementing UI Automation Control Patterns, and view the page that documents the pattern you intend to implement.

Control pattern	Class provider interface	Description
Annotation	IAnnotationProvider	Used to expose the properties of an annotation in a document.
Dock	IDockProvider	Used for controls that can be docked in a docking container. For example, toolbars or tool palettes.
Drag	IDragProvider	Used to support draggable controls, or controls with draggable items.
DropTarget	IDropTargetProvider	Used to support controls that can be the target of a drag-and-drop operation.
ExpandCollapse	IExpandCollapseProvider	Used to support controls that visually expand to display more content and collapse to hide content.

Control pattern	Class provider interface	Description
Grid	IGridProvider	Used for controls that support grid functionality such as sizing and moving to a specified cell. Note that Grid itself does not implement this pattern because it provides layout but is not a control.
GridItem	IGridItemProvider	Used for controls that have cells within grids.
Invoke	IInvokeProvider	Used for controls that can be invoked, such as a Button.
ItemContainer	IItemContainerProvider	Enables applications to find an element in a container, such as a virtualized list.
MultipleView	IMultipleViewProvider	Used for controls that can switch between multiple representations of the same set of information, data, or children.
ObjectModel	IObjectModelProvider	Used to expose a pointer to the underlying object model of a document.
RangeValue	IRangeValueProvider	Used for controls that have a range of values that can be applied to the control. For example, a spinner control containing years might have a range of 1900 to the current year, while another spinner control presenting months would have a range of 1 to 12.
Scroll	IScrollProvider	Used for controls that can scroll. For example, a control that has scroll bars that are active when there is more information than can be displayed in the viewable area of the control.
ScrollItem	IScrollItemProvider	Used for controls that have individual items in a list that scrolls. For example, a list control that has individual items in the scroll list, such as a combo box control.
Selection	ISelectionProvider	Used for selection container controls. For example, ListBox and ComboBox.
SelectionItem	ISelectionItemProvider	Used for individual items in selection container controls, such as list boxes and combo boxes.
Spreadsheet	ISpreadsheetProvider	Used to expose the contents of a spreadsheet or other grid-based document.

Control pattern	Class provider interface	Description
SpreadsheetItem	ISpreadsheetItemProvider	Used to expose the properties of a cell in a spreadsheet or other grid-based document.
Styles	IStylesProvider	Used to describe a UI element that has a specific style, fill color, fill pattern, or shape.
SynchronizedInput	ISynchronizedInputProvider	Enables UI Automation client apps to direct the mouse or keyboard input to a specific UI element.
Table	ITableProvider	Used for controls that have a grid as well as header information. For example, a tabular calendar control.
TableItem	ITableItemProvider	Used for items in a table.
Text	ITextProvider	Used for edit controls and documents that expose textual information. See also ITextRangeProvider and ITextProvider2.
TextChild	ITextChildProvider	Used to access an element's nearest ancestor that supports the **Text** control pattern.
TextEdit	No managed class available	Provides access to a control that modifies text, for example a control that performs auto-correction or enables input composition through an Input Method Editor (IME).
TextRange	ITextRangeProvider	Provides access to a span of continuous text in a text container that implements ITextProvider. See also ITextRangeProvider2.
Toggle	IToggleProvider	Used for controls where the state can be toggled. For example, CheckBox and menu items that can be checked.
Transform	ITransformProvider	Used for controls that can be resized, moved, and rotated. Typical uses for the Transform control pattern are in designers, forms, graphical editors, and drawing applications.
Value	IValueProvider	Allows clients to get or set a value on controls that do not support a range of values.

Control pattern	Class provider interface	Description
VirtualizedItem	IVirtualizedItemProvider	Exposes items inside containers that are virtualized and need to be made fully accessible as UI Automation elements.
Window	IWindowProvider	Exposes information specific to windows, a fundamental concept to the Microsoft Windows operating system. Examples of controls that are windows are child windows and dialogs.

> **Note**
>
> You won't necessarily find implementations of all these patterns in existing XAML controls. Some of the patterns have interfaces solely to support parity with the general UI Automation framework definition of patterns, and to support automation peer scenarios that will require a purely custom implementation to support that pattern.

> **Note**
>
> Windows Phone Store apps do not support all the UI Automation control patterns listed here. **Annotation**, **Dock**, **Drag**, **DropTarget**, **ObjectModel** are some of the unsupported patterns.

Related topics

- Custom automation peers
- Accessibility

Guidelines for app settings

Article • 06/24/2021

App settings are the user-customizable portions of your Windows app accessed through an app settings page. For example, a news reader app might let the user specify which news sources to display or how many columns to display on the screen, while a weather app could let the user choose between Celsius and Fahrenheit. This article provides recommendations and best practices for creating and displaying app settings.

When to provide a settings page

Here are examples of app options that belong in an app settings page:

- Configuration options that affect the behavior of the app and don't require frequent readjustment, like choosing between Celsius or Fahrenheit as default units for temperature in a weather app, changing account settings for a mail app, settings for notifications, or accessibility options.
- Options that depend on the user's preferences, like music, sound effects, or color themes.
- App information that isn't accessed very often, such as privacy policy, help, app version, or copyright info.

Commands that are part of the typical app workflow (for example, changing the brush size in an art app) shouldn't be in a settings page. To learn more about command placement, see Command design basics.

General recommendations

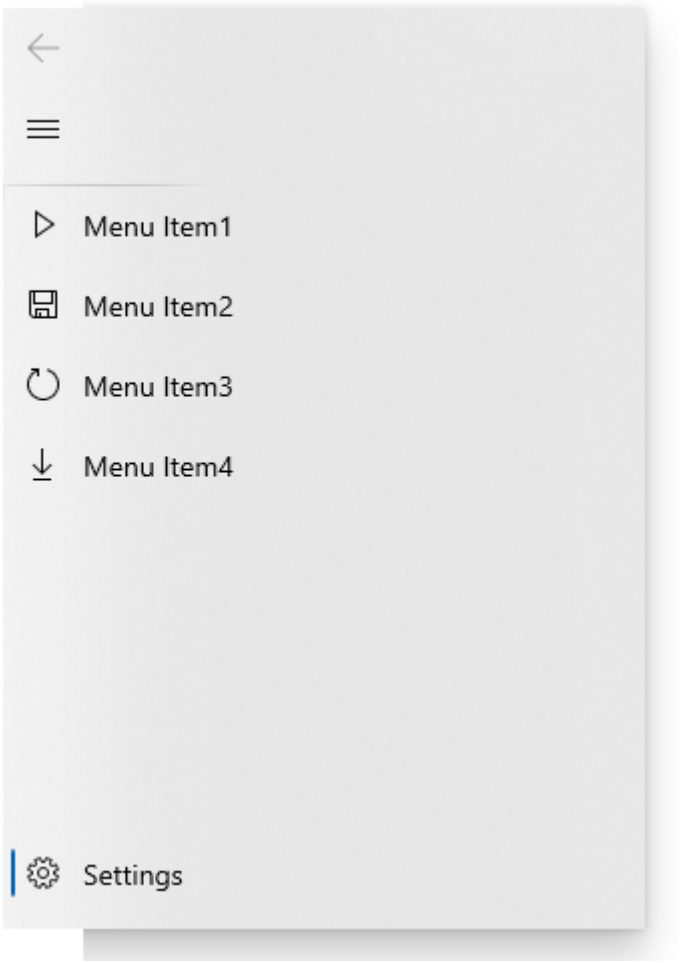
- Keep settings pages simple and make use of binary (on/off) controls. A toggle switch is usually the best control for a binary setting.
- For settings that let users choose one item from a set of up to 5 mutually exclusive, related options, use radio buttons.
- Create an entry point for all app settings in your app setting's page.
- Keep your settings simple. Define smart defaults and keep the number of settings to a minimum.
- When a user changes a setting, the app should immediately reflect the change.
- Don't include commands that are part of the common app workflow.

Entry point

The way that users get to your app settings page should be based on your app's layout.

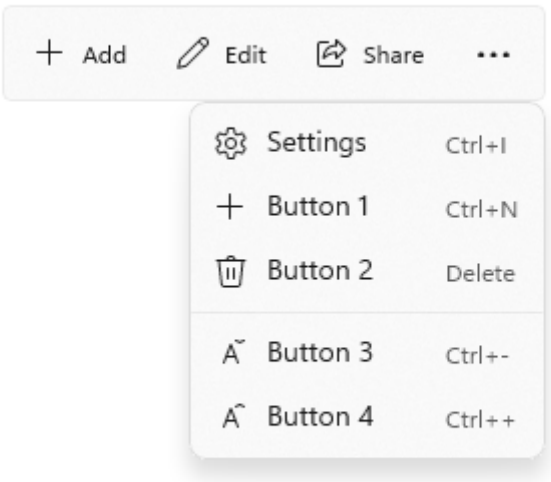
Navigation pane

For a nav pane layout, app settings should be the last item in the list of navigational choices and be pinned to the bottom:



Command bar

If you're using a command bar or tool bar, place the settings entry point as one of the last items in the "More" overflow menu. If greater discoverability for the settings entry point is important for your app, place the entry point directly on the command bar and not within the overflow.



Hub

If you're using a hub layout, the entry point for app settings should be placed inside the "More" overflow menu of a command bar.

Tabs/pivots

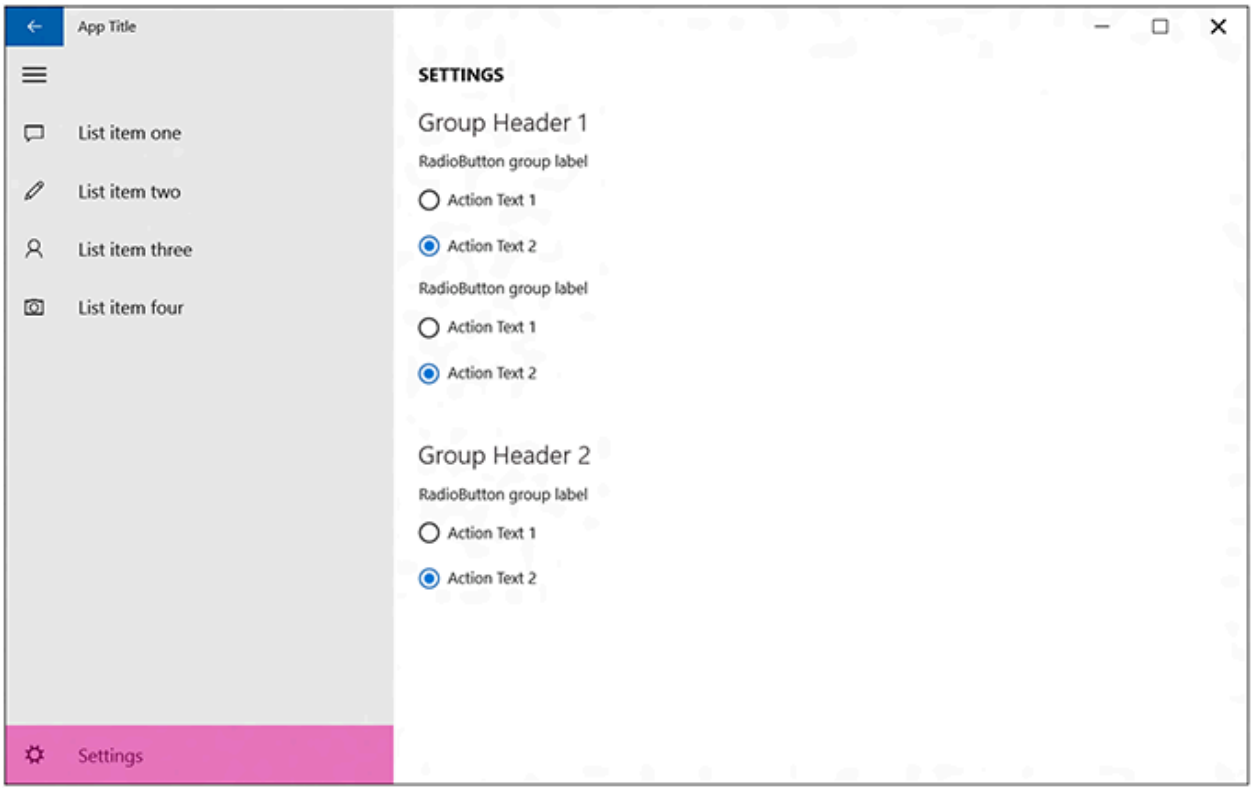
For a tabs or pivots layout, we don't recommend placing the app settings entry point as one of the top items within the navigation. Instead, the entry point for app settings should be placed inside the "More" overflow menu of a command bar.

List-details

Instead of burying the app settings entry point deeply within a list-details pane, make it the last pinned item on the top level of the list pane.

Layout

The app settings window should open full-screen and fill the whole window. If your app settings menu has up to four top-level groups, those groups should cascade down one column.

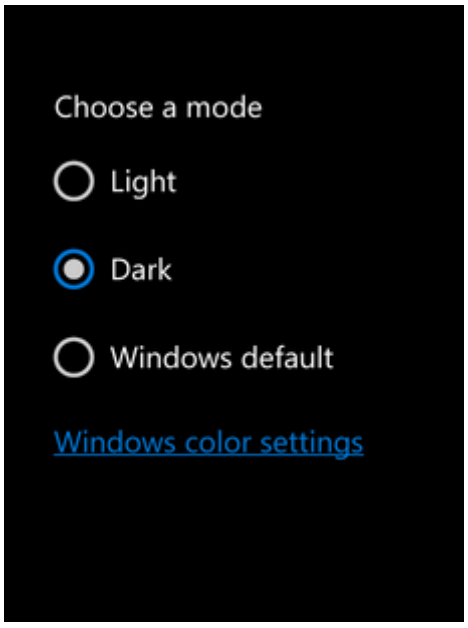


"Color mode" settings

If your app allows users to choose the app's color mode, present these options using radio buttons or a combo box with the header "Choose an app mode". The options should read

- Light
- Dark
- Windows default

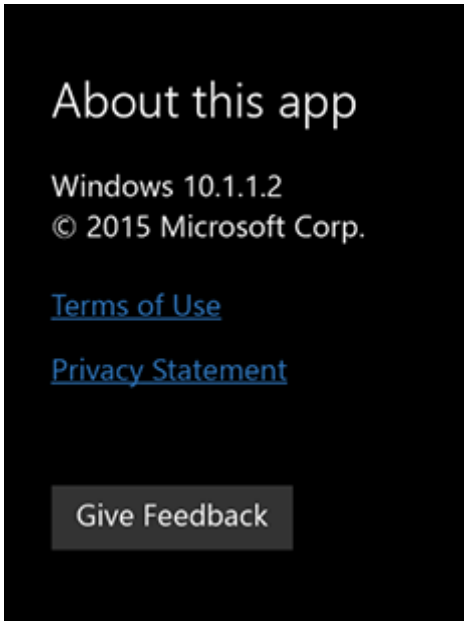
We also recommend adding a hyperlink to the Colors page of the Windows Settings app where users can access and modify the current default app mode. Use the string "Windows color settings" for the hyperlink text and `ms-settings:colors` for the URI.



About section and Feedback button

We recommend placing "About this App" section in your app either as a dedicated page or in its own section. If you want a "Send Feedback" button, place that toward the bottom of the "About this App" page.

Under a "Legal" subheader, place any "Terms of Use" and "Privacy Statement" (should be hyperlink buttons with wrapping text) as well as additional legal information, such as copyright.



Recommended page content

Once you have a list of items that you want to include in your app settings page, consider these guidelines:

- Group similar or related settings under one settings label.
- Try to keep the total number of settings to a maximum of four or five.
- Display the same settings regardless of the app context. If some settings aren't relevant in a certain context, disable those in the app settings flyout.
- Use descriptive, one-word labels for settings. For example, name the setting "Accounts" instead of "Account settings" for account-related settings. If you only want one option for your settings and the settings don't lend themselves to a descriptive label, use "Options" or "Defaults."
- If a setting directly links to the web instead of to a flyout, let the user know this with a visual clue, such as "Help (online)" or "Web forums" styled as a hyperlink. Consider grouping multiple links to the web into a flyout with a single setting. For example, an "About" setting could open a flyout with links to your terms of use, privacy policy, and app support.
- Combine less-used settings into a single entry so that more common settings can each have their own entry. Put content or links that only contain information in an "About" setting.
- Don't duplicate the functionality in the "Permissions" pane. Windows provides this pane by default and you can't modify it.
- Add settings content to Settings flyouts
- Present content from top to bottom in a single column, scrollable if necessary. Limit scrolling to a maximum of twice the screen height.
- Use the following controls for app settings:
  - Toggle switches: To let users set values on or off.
  - Radio buttons: To let users choose one item from a set of up to 5 mutually exclusive, related options.
  - Text input box: To let users enter text. Use the type of text input box that corresponds to the type of text you're getting from the user, such as an email or password.
  - Hyperlinks: To take the user to another page within the app or to an external website. When a user clicks a hyperlink, the Settings flyout will be dismissed.
  - Buttons: To let users initiate an immediate action without dismissing the current Settings flyout.
- Add a descriptive message if one of the controls is disabled. Place this message above the disabled control.

- Animate content and controls as a single block after the Settings flyout and header have animated. Animate content using the **enterPage** or **EntranceThemeTransition** animations with a 100px left offset.
- Use section headers, paragraphs, and labels to aid organize and clarify content, if necessary.
- If you need to repeat settings, use an additional level of UI or an expand/collapse model, but avoid hierarchies deeper than two levels. For example, a weather app that provides per-city settings could list the cities and let the user tap on the city to either open a new flyout or expand to show the settings options.
- If loading controls or web content takes time, use an indeterminate progress control to indicate to users that info is loading. For more info, see Guidelines for progress controls.
- Don't use buttons for navigation or to commit changes. Use hyperlinks to navigate to other pages, and instead of using a button to commit changes, automatically save changes to app settings when a user dismisses the Settings flyout.

Related articles

- Command design basics
- Guidelines for progress controls
- Store and retrieve app data
- **EntranceThemeTransition**

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Store and retrieve settings and other app data

Article • 06/16/2022

App data is mutable data that is created and managed by a specific app. It includes runtime state, app settings, user preferences, reference content (such as the dictionary definitions in a dictionary app), and other settings. App data is different from *user data*, data that the user creates and manages when using an app. User data includes document or media files, email or communication transcripts, or database records holding content created by the user. User data may be useful or meaningful to more than one app. Often, this is data that the user wants to manipulate or transmit as an entity independent of the app itself, such as a document.

Important note about app data: The lifetime of the app data is tied to the lifetime of the app. If the app is removed, all of the app data will be lost as a consequence. Don't use app data to store user data or anything that users might perceive as valuable and irreplaceable. We recommend that the user's libraries and Microsoft OneDrive be used to store this sort of information. App data is ideal for storing app-specific user preferences, settings, and favorites.

Types of app data

There are two types of app data: settings and files.

Settings

Use settings to store user preferences and application state info. The app data API enables you to easily create and retrieve settings (we'll show you some examples later in this article).

Here are data types you can use for app settings:

- **UInt8**, **Int16**, **UInt16**, **Int32**, **UInt32**, **Int64**, **UInt64**, **Single**, **Double**
- **Boolean**
- **Char16**, **String**
- DateTime, TimeSpan
  - For C#/.NET, use: System.DateTimeOffset, System.TimeSpan
- **GUID**, Point, Size, Rect
- ApplicationDataCompositeValue: A set of related app settings that must be serialized and deserialized atomically. Use composite settings to easily handle

atomic updates of interdependent settings. The system ensures the integrity of composite settings during concurrent access and roaming. Composite settings are optimized for small amounts of data, and performance can be poor if you use them for large data sets.

Files

Use files to store binary data or to enable your own, customized serialized types.

Storing app data in the app data stores

When an app is installed, the system gives it its own per-user data stores for settings and files. You don't need to know where or how this data exists, because the system is responsible for managing the physical storage, ensuring that the data is kept isolated from other apps and other users. The system also preserves the contents of these data stores when the user installs an update to your app and removes the contents of these data stores completely and cleanly when your app is uninstalled.

Within its app data store, each app has system-defined root directories: one for local files, one for roaming files, and one for temporary files. Your app can add new files and new containers to each of these root directories.

Local app data

Local app data should be used for any information that needs to be preserved between app sessions and is not suitable for roaming app data. Data that is not applicable on other devices should be stored here as well. There is no general size restriction on local data stored. Use the local app data store for data that it does not make sense to roam and for large data sets.

Retrieve the local app data store

Before you can read or write local app data, you must retrieve the local app data store. To retrieve the local app data store, use the **ApplicationData.LocalSettings** property to get the app's local settings as an **ApplicationDataContainer** object. Use the **ApplicationData.LocalFolder** property to get the files in a **StorageFolder** object. Use the **ApplicationData.LocalCacheFolder** property to get the folder in the local app data store where you can save files that are not included in backup and restore.

```
CSharp
```

```
Windows.Storage.ApplicationDataContainer localSettings =
    Windows.Storage.ApplicationData.Current.LocalSettings;
Windows.Storage.StorageFolder localFolder =
    Windows.Storage.ApplicationData.Current.LocalFolder;
```

Create and retrieve a simple local setting

To create or write a setting, use the ApplicationDataContainer.Values property to access the settings in the `localSettings` container we got in the previous step. This example creates a setting named `exampleSetting`.

CSharp

```
// Simple setting

localSettings.Values["exampleSetting"] = "Hello Windows";
```

To retrieve the setting, you use the same ApplicationDataContainer.Values property that you used to create the setting. This example shows how to retrieve the setting we just created.

CSharp

```
// Simple setting
Object value = localSettings.Values["exampleSetting"];
```

Create and retrieve a local composite value

To create or write a composite value, create an ApplicationDataCompositeValue object. This example creates a composite setting named `exampleCompositeSetting` and adds it to the `localSettings` container.

CSharp

```
// Composite setting

Windows.Storage.ApplicationDataCompositeValue composite =
    new Windows.Storage.ApplicationDataCompositeValue();
composite["intVal"] = 1;
composite["strVal"] = "string";

localSettings.Values["exampleCompositeSetting"] = composite;
```

This example shows how to retrieve the composite value we just created.

CSharp

```
// Composite setting

Windows.Storage.ApplicationDataCompositeValue composite =

(Windows.Storage.ApplicationDataCompositeValue)localSettings.Values["example
CompositeSetting"];

if (composite == null)
{
   // No data
}
else
{
   // Access data in composite["intVal"] and composite["strVal"]
}
```

Create and read a local file

To create and update a file in the local app data store, use the file APIs, such as Windows.Storage.StorageFolder.CreateFileAsync and Windows.Storage.FileIO.WriteTextAsync. This example creates a file named `dataFile.txt` in the `localFolder` container and writes the current date and time to the file. The **ReplaceExisting** value from the CreationCollisionOption enumeration indicates to replace the file if it already exists.

C#

```
async void WriteTimestamp()
{
   Windows.Globalization.DateTimeFormatting.DateTimeFormatter formatter =
       new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("longtime");

   StorageFile sampleFile = await
localFolder.CreateFileAsync("dataFile.txt",
       CreationCollisionOption.ReplaceExisting);
   await FileIO.WriteTextAsync(sampleFile,
formatter.Format(DateTimeOffset.Now));
}
```

To open and read a file in the local app data store, use the file APIs, such as Windows.Storage.StorageFolder.GetFileAsync, Windows.Storage.StorageFile.GetFileFromApplicationUriAsync, and

Windows.Storage.FileIO.ReadTextAsync. This example opens the `dataFile.txt` file created in the previous step and reads the date from the file. For details on loading file resources from various locations, see How to load file resources.

C#

```
async void ReadTimestamp()
{
   try
   {
      StorageFile sampleFile = await
localFolder.GetFileAsync("dataFile.txt");
      String timestamp = await FileIO.ReadTextAsync(sampleFile);
      // Data is contained in timestamp
   }
   catch (Exception)
   {
      // Timestamp not found
   }
}
```

Roaming data

> ⚠ **Warning**
>
> Roaming data and settings is no longer supported as of Windows 11. The recommended replacement is Azure App Service. Azure App Service is widely supported, well documented, reliable, and supports cross-platform/cross-ecosystem scenarios such as iOS, Android and web.
>
> The following documentation applies to Windows 10 versions 1909 and lower.

If you use roaming data in your app, your users can easily keep your app's app data in sync across multiple devices. If a user installs your app on multiple devices, the OS keeps the app data in sync, reducing the amount of setup work that the user needs to do for your app on their second device. Roaming also enables your users to continue a task, such as composing a list, right where they left off even on a different device. The OS replicates roaming data to the cloud when it is updated, and synchronizes the data to the other devices on which the app is installed.

The OS limits the size of the app data that each app may roam. See ApplicationData.RoamingStorageQuota. If the app hits this limit, none of the app's app data will be replicated to the cloud until the app's total roamed app data is less than the

limit again. For this reason, it is a best practice to use roaming data only for user preferences, links, and small data files.

Roaming data for an app is available in the cloud as long as it is accessed by the user from some device within the required time interval. If the user does not run an app for longer than this time interval, its roaming data is removed from the cloud. If a user uninstalls an app, its roaming data isn't automatically removed from the cloud, it's preserved. If the user reinstalls the app within the time interval, the roaming data is synchronized from the cloud.

Roaming data do's and don'ts

> See important note about roaming data.

- Use roaming for user preferences and customizations, links, and small data files. For example, use roaming to preserve a user's background color preference across all devices.
- Use roaming to let users continue a task across devices. For example, roam app data like the contents of a drafted email or the most recently viewed page in a reader app.
- Handle the **DataChanged** event by updating app data. This event occurs when app data has just finished syncing from the cloud.
- Roam references to content rather than raw data. For example, roam a URL rather than the content of an online article.
- For important, time critical settings, use the *HighPriority* setting associated with **RoamingSettings**.
- Don't roam app data that is specific to a device. Some info is only pertinent locally, such as a path name to a local file resource. If you do decide to roam local information, make sure that the app can recover if the info isn't valid on the secondary device.
- Don't roam large sets of app data. There's a limit to the amount of app data an app may roam; use **RoamingStorageQuota** property to get this maximum. If an app hits this limit, no data can roam until the size of the app data store no longer exceeds the limit. When you design your app, consider how to put a bound on larger data so as to not exceed the limit. For example, if saving a game state requires 10KB each, the app might only allow the user store up to 10 games.
- Don't use roaming for data that relies on instant syncing. Windows doesn't guarantee an instant sync; roaming could be significantly delayed if a user is offline or on a high latency network. Ensure that your UI doesn't depend on instant syncing.

- Don't use roaming for frequently changing data. For example, if your app tracks frequently changing info, such as the position in a song by second, don't store this as roaming app data. Instead, pick a less frequent representation that still provides a good user experience, like the currently playing song.

Roaming pre-requisites

> See important note about roaming data.

Any user can benefit from roaming app data if they use a Microsoft account to log on to their device. However, users and group policy administrators can switch off roaming app data on a device at any time. If a user chooses not to use a Microsoft account or disables roaming data capabilities, she will still be able to use your app, but app data will be local to each device.

Data stored in the **PasswordVault** will only transition if a user has made a device "trusted". If a device isn't trusted, data secured in this vault will not roam.

Conflict resolution

> See important note about roaming data.

Roaming app data is not intended for simultaneous use on more than one device at a time. If a conflict arises during synchronization because a particular data unit was changed on two devices, the system will always favor the value that was written last. This ensures that the app utilizes the most up-to-date information. If the data unit is a setting composite, conflict resolution will still occur on the level of the setting unit, which means that the composite with the latest change will be synchronized.

When to write data

> See important note about roaming data.

Depending on the expected lifetime of the setting, data should be written at different times. Infrequently or slowly changing app data should be written immediately. However, app data that changes frequently should only be written periodically at regular intervals (such as once every 5 minutes), as well as when the app is suspended. For example, a music app might write the "current song" settings whenever a new song starts to play, however, the actual position in the song should only be written on suspend.

Excessive usage protection

> See important note about roaming data.

The system has various protection mechanisms in place to avoid inappropriate use of resources. If app data does not transition as expected, it is likely that the device has been temporarily restricted. Waiting for some time will usually resolve this situation automatically and no action is required.

Versioning

> See important note about roaming data.

App data can utilize versioning to upgrade from one data structure to another. The version number is different from the app version and can be set at will. Although not enforced, it is highly recommended that you use increasing version numbers, since undesirable complications (including data loss)could occur if you try to transition to a lower data version number that represents newer data.

App data only roams between installed apps with the same version number. For example, devices on version 2 will transition data between each other and devices on version 3 will do the same, but no roaming will occur between a device running version 2 and a device running version 3. If you install a new app that utilized various version numbers on other devices, the newly installed app will sync the app data associated with the highest version number.

Testing and tools

> See important note about roaming data.

Developers can lock their device in order to trigger a synchronization of roaming app data. If it seems that the app data does not transition within a certain time frame, please check the following items and make sure that:

- Your roaming data does not exceed the maximum size (see **RoamingStorageQuota** for details).
- Your files are closed and released properly.
- There are at least two devices running the same version of the app.

Register to receive notification when roaming data changes

> See important note about roaming data.

To use roaming app data, you need to register for roaming data changes and retrieve the roaming data containers so you can read and write settings.

1. Register to receive notification when roaming data changes.

   The **DataChanged** event notifies you when roaming data changes. This example sets `DataChangeHandler` as the handler for roaming data changes.

```
C#
```

```
void InitHandlers()
    {
       Windows.Storage.ApplicationData.Current.DataChanged +=
          new TypedEventHandler<ApplicationData, object>(DataChangeHandler);
    }

    void DataChangeHandler(Windows.Storage.ApplicationData appData, object
o)
    {
       // TODO: Refresh your data
    }
```

2. Get the containers for the app's settings and files.

   Use the **ApplicationData.RoamingSettings** property to get the settings and the **ApplicationData.RoamingFolder** property to get the files.

```
C#
```

```
Windows.Storage.ApplicationDataContainer roamingSettings =
        Windows.Storage.ApplicationData.Current.RoamingSettings;
    Windows.Storage.StorageFolder roamingFolder =
        Windows.Storage.ApplicationData.Current.RoamingFolder;
```

Create and retrieve roaming settings

> See important note about roaming data.

Use the ApplicationDataContainer.Values property to access the settings in the `roamingSettings` container we got in the previous section. This example creates a simple setting named `exampleSetting` and a composite value named `composite`.

C#

```
// Simple setting

roamingSettings.Values["exampleSetting"] = "Hello World";
// High Priority setting, for example, last page position in book reader app
roamingSettings.values["HighPriority"] = "65";

// Composite setting

Windows.Storage.ApplicationDataCompositeValue composite =
    new Windows.Storage.ApplicationDataCompositeValue();
composite["intVal"] = 1;
composite["strVal"] = "string";

roamingSettings.Values["exampleCompositeSetting"] = composite;
```

This example retrieves the settings we just created.

C#

```
// Simple setting

Object value = roamingSettings.Values["exampleSetting"];

// Composite setting

Windows.Storage.ApplicationDataCompositeValue composite =

(Windows.Storage.ApplicationDataCompositeValue)roamingSettings.Values["examp
leCompositeSetting"];

if (composite == null)
{
   // No data
}
else
{
   // Access data in composite["intVal"] and composite["strVal"]
}
```

Create and retrieve roaming files

> See important note about roaming data.

To create and update a file in the roaming app data store, use the file APIs, such as Windows.Storage.StorageFolder.CreateFileAsync and Windows.Storage.FileIO.WriteTextAsync. This example creates a file named `dataFile.txt` in the `roamingFolder` container and writes the current date and time to the file. The **ReplaceExisting** value from the CreationCollisionOption enumeration indicates to replace the file if it already exists.

C#

```
async void WriteTimestamp()
{
   Windows.Globalization.DateTimeFormatting.DateTimeFormatter formatter =
       new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("longtime");

   StorageFile sampleFile = await
roamingFolder.CreateFileAsync("dataFile.txt",
       CreationCollisionOption.ReplaceExisting);
   await FileIO.WriteTextAsync(sampleFile,
formatter.Format(DateTimeOffset.Now));
}
```

To open and read a file in the roaming app data store, use the file APIs, such as Windows.Storage.StorageFolder.GetFileAsync, Windows.Storage.StorageFile.GetFileFromApplicationUriAsync, and Windows.Storage.FileIO.ReadTextAsync. This example opens the `dataFile.txt` file created in the previous section and reads the date from the file. For details on loading file resources from various locations, see How to load file resources.

C#

```
async void ReadTimestamp()
{
   try
   {
      StorageFile sampleFile = await
roamingFolder.GetFileAsync("dataFile.txt");
      String timestamp = await FileIO.ReadTextAsync(sampleFile);
      // Data is contained in timestamp
   }
   catch (Exception)
   {
      // Timestamp not found
   }
}
```

Temporary app data

The temporary app data store works like a cache. Its files do not roam and could be removed at any time. The System Maintenance task can automatically delete data stored at this location at any time. The user can also clear files from the temporary data store using Disk Cleanup. Temporary app data can be used for storing temporary information during an app session. There is no guarantee that this data will persist beyond the end of the app session as the system might reclaim the used space if needed. The location is available via the **temporaryFolder** property.

Retrieve the temporary data container

Use the **ApplicationData.TemporaryFolder** property to get the files. The next steps use the `temporaryFolder` variable from this step.

C#

```
Windows.Storage.StorageFolder temporaryFolder =
ApplicationData.Current.TemporaryFolder;
```

Create and read temporary files

To create and update a file in the temporary app data store, use the file APIs, such as **Windows.Storage.StorageFolder.CreateFileAsync** and **Windows.Storage.FileIO.WriteTextAsync**. This example creates a file named `dataFile.txt` in the `temporaryFolder` container and writes the current date and time to the file. The **ReplaceExisting** value from the **CreationCollisionOption** enumeration indicates to replace the file if it already exists.

C#

```
async void WriteTimestamp()
{
   Windows.Globalization.DateTimeFormatting.DateTimeFormatter formatter =
       new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("longtime");

   StorageFile sampleFile = await
temporaryFolder.CreateFileAsync("dataFile.txt",
       CreateCollisionOption.ReplaceExisting);
   await FileIO.WriteTextAsync(sampleFile,
formatter.Format(DateTimeOffset.Now));
}
```

To open and read a file in the temporary app data store, use the file APIs, such as Windows.Storage.StorageFolder.GetFileAsync, Windows.Storage.StorageFile.GetFileFromApplicationUriAsync, and Windows.Storage.FileIO.ReadTextAsync. This example opens the `dataFile.txt` file created in the previous step and reads the date from the file. For details on loading file resources from various locations, see How to load file resources.

C#

```
async void ReadTimestamp()
{
   try
   {
      StorageFile sampleFile = await
temporaryFolder.GetFileAsync("dataFile.txt");
      String timestamp = await FileIO.ReadTextAsync(sampleFile);
      // Data is contained in timestamp
   }
   catch (Exception)
   {
      // Timestamp not found
   }
}
```

Organize app data with containers

To help you organize your app data settings and files, you create containers (represented by ApplicationDataContainer objects) instead of working directly with directories. You can add containers to the local, roaming, and temporary app data stores. Containers can be nested up to 32 levels deep.

To create a settings container, call the ApplicationDataContainer.CreateContainer method. This example creates a local settings container named `exampleContainer` and adds a setting named `exampleSetting`. The **Always** value from the ApplicationDataCreateDisposition enumeration indicates that the container is created if it doesn't already exist.

C#

```
Windows.Storage.ApplicationDataContainer localSettings =
    Windows.Storage.ApplicationData.Current.LocalSettings;
Windows.Storage.StorageFolder localFolder =
    Windows.Storage.ApplicationData.Current.LocalFolder;

// Setting in a container
Windows.Storage.ApplicationDataContainer container =
```

```
    localSettings.CreateContainer("exampleContainer",
Windows.Storage.ApplicationDataCreateDisposition.Always);

if (localSettings.Containers.ContainsKey("exampleContainer"))
{
    localSettings.Containers["exampleContainer"].Values["exampleSetting"] =
"Hello Windows";
}
```

Delete app settings and containers

To delete a simple setting that your app no longer needs, use the ApplicationDataContainerSettings.Remove method. This example deletes the `exampleSetting` local setting that we created earlier.

C#

```
Windows.Storage.ApplicationDataContainer localSettings =
    Windows.Storage.ApplicationData.Current.LocalSettings;
Windows.Storage.StorageFolder localFolder =
    Windows.Storage.ApplicationData.Current.LocalFolder;

// Delete simple setting

localSettings.Values.Remove("exampleSetting");
```

To delete a composite setting, use the ApplicationDataCompositeValue.Remove method. This example deletes the local `exampleCompositeSetting` composite setting we created in an earlier example.

C#

```
Windows.Storage.ApplicationDataContainer localSettings =
    Windows.Storage.ApplicationData.Current.LocalSettings;
Windows.Storage.StorageFolder localFolder =
    Windows.Storage.ApplicationData.Current.LocalFolder;

// Delete composite setting

localSettings.Values.Remove("exampleCompositeSetting");
```

To delete a container, call the ApplicationDataContainer.DeleteContainer method. This example deletes the local `exampleContainer` settings container we created earlier.

C#

```
Windows.Storage.ApplicationDataContainer localSettings =
    Windows.Storage.ApplicationData.Current.LocalSettings;
Windows.Storage.StorageFolder localFolder =
    Windows.Storage.ApplicationData.Current.LocalFolder;

// Delete container

localSettings.DeleteContainer("exampleContainer");
```

Versioning your app data

You can optionally version the app data for your app. This would enable you to create a future version of your app that changes the format of its app data without causing compatibility problems with the previous version of your app. The app checks the version of the app data in the data store, and if the version is less than the version the app expects, the app should update the app data to the new format and update the version. For more info, see the**Application.Version** property and the **ApplicationData.SetVersionAsync** method.

Related articles

- **Windows.Storage.ApplicationData**
- **Windows.Storage.ApplicationData.RoamingSettings**
- **Windows.Storage.ApplicationData.RoamingFolder**
- **Windows.Storage.ApplicationData.RoamingStorageQuota**
- **Windows.Storage.ApplicationDataCompositeValue**

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Globalization and localization

Article • 12/15/2021

Windows is used worldwide by audiences that are diverse in terms of language, region, and culture. Your users speak a variety of different languages and in a variety of different countries and regions. Some users speak more than one language. So, your app runs on configurations that involve many permutations of system settings for language, region, and culture. You can increase the potential market for your app by designing it to be readily adaptable, using *globalization* and *localization*.

For a short video that demonstrates how to prepare your app for the world, see Introduction to globalization and localization.

Globalization is the process of designing and developing your app in such a way that it functions appropriately in different global markets (on systems with different language and culture configurations) without requiring culture-specific changes or customization.

- Take culture into account when manipulating strings, for example don't change the case of strings before comparing them.
- Use calendars that are appropriate for the current culture.
- Use globalization APIs to display data that are formatted appropriately for the country or region, such as numbers, dates, times, and currencies.
- Take into account that different cultures have different rules for collating (sorting) text and other data.

Your code needs to function equally well in any of the cultures that you've determined that your app will support. Ideally, your code will function equally well in the context of *any* language, region, or culture. The most efficient way to globalize your app's functions is to use the concept of cultures/locales. A culture/locale is a set of rules and a set of data that are specific to a given language and geographic area. These rules and data include information about the following.

- Character classification
- Writing system
- Date and time formatting
- Numeric, currency, weight, and measure conventions
- Sorting rules

> **Note**

For a list of supported locale names by Windows operating system version, see the Language tag column of the table in **Appendix A: Product Behavior** in the **Windows Language Code Identifier (LCID) Reference**.

Localizability is the process of preparing a globalized app for localization and/or verifying that the app is ready for localization. Correctly making an app localizable means that the later localization process will not uncover any functional defects in the app. The most essential property of a localizable app is that its executable code has been cleanly separated from the app's localizable resources.

- Strings translated into different languages can vary greatly in length. So, design your UI to accommodate different text lengths and font sizes for labels and text input controls.
- Try to avoid text and/or culturally-sensitive material in images.
- Don't hard-code strings and culture-dependent images in your app's code and markup. Instead, store them as string and image resources so that they can be adapted to different local markets independently of your app's built binaries.
- Pseudo-localize your app to disclose any localizability issues.

Localization is the process of adapting or translating your app's localizable resources to meet the language, cultural, and political requirements of the specific local markets that the app is intended to support. By the time you reach the stage of localizing your app, if your app is localizable then you will not have to modify any logic during this process.

- Translate the string resources and other assets of the app for the new market.
- Modify any culture-dependent images as necessary.
- Files can also vary depending on a user's region, separate from their language. For example, a map may have different borders depending on the user's location, but the labels should follow the user's preferred language.

Most localization teams use special tools to aid the process. For example, by recycling translations of recurring text.

Article	Description
Guidelines for globalization	Design and develop your app in such a way that it functions appropriately on systems with different language and culture configurations.
Understand user profile languages and app manifest languages	This topic defines the terms "user profile language list", "app manifest language list", and "app runtime language list". We'll be using these terms in this topic and other topics in this feature area, so it's important to know what they mean.

Article	Description
Globalize your date/time/number formats	Design your app to be global-ready by appropriately formatting dates, times, numbers, phone numbers, and currencies. You'll then be able later to adapt your app for additional cultures, regions, and languages in the global market.
Use templates and patterns to format dates and times	Use classes in the **Windows.Globalization.DateTimeFormatting** namespace with custom templates and patterns to display dates and times in exactly the format you wish.
Adjust layout and fonts, and support RTL	Design your app to support the layouts and fonts of multiple languages, including RTL (right-to-left) flow direction.
NumeralSystem values	This topic lists the values available to the **NumeralSystem** property of various classes in the **Windows.Globalization** namespace.
Make your app localizable	A localized app is one that can be localized to other markets, languages, or regions without uncovering any functional defects in the app. The most essential property of a localizable app is that its executable code has been cleanly separated from its localizable resources.
International fonts	This topic lists the fonts available for Windows apps that are localized into languages other than U.S. English.
Design your app for bidirectional text	Design your app to provide bi-directional text support (BiDi) so that you can combine script from left-to-right and right-to-left writing systems.
Use the Multilingual App Toolkit 4.0	The Multilingual App Toolkit (MAT) 4.0 integrates with Microsoft Visual Studio 2017 and later to provide Windows apps with translation support, translation file management, and editor tools.
Multilingual App Toolkit 4.0 FAQ & troubleshooting	This topic provides answers to frequently-asked questions and issues related to the Multilingual App Toolkit (MAT) 4.0.
Use the UTF-8 code page	UTF-8 is the universal code page for internationalization.
Prepare your application for the Japanese era change	Learn about the May 2019 Japanese era change and how to prepare your application.

Guidelines for globalization

Article • 06/24/2021

Design and develop your app in such a way that it functions appropriately on systems with different language and culture configurations. Use **Globalization** APIs to format data; and avoid assumptions in your code about language, region, character classification, writing system, date/time formatting, numbers, currencies, weights, and sorting rules.

Expand table

Recommendation	Description
Take culture into account when manipulating and comparing strings.	For example, don't change the case of strings before comparing them. See Recommendations for String Usage.
When collating (sorting) strings and other data, don't assume that it's is always done alphabetically.	For languages that don't use Latin script, collation is based on factors such as pronunciation, or number of pen strokes. Even languages that do use Latin script don't always use alphabetic sorting. For example, in some cultures, a phone book might not be sorted alphabetically. Windows can handle sorting for you, but if you create your own sorting algorithm then be sure to take into account the sorting methods used in your target markets.
Appropriately format numbers, dates, times, addresses, and phone numbers.	These formats vary between cultures, regions, languages, and markets. If you're displaying these data then use **Globalization** APIs to get the format appropriate for a particular audience. See Globalize your date/time/number formats. The order in which family and given names are displayed, and the format of addresses, can differ as well. Use standard date, time, and number displays. Use standard date and time picker controls. Use standard address information.
Support international units of measurement and currency.	Different units and scales are used in different countries, although the most popular are the metric system and the imperial system. Be sure to support the correct system measurement if you deal with measurements such as length, temperature, and area. Use the **GeographicRegion.CurrenciesInUse** property to get the set of currencies in use in a region.
Use Unicode for character encoding.	By default, Microsoft Visual Studio uses Unicode character encoding for all documents. If you're using a different editor, be sure to save source files in the appropriate Unicode character encodings. All Windows Runtime APIs return UTF-16 encoded strings.

Recommendation	Description
Support international paper sizes.	The most common paper sizes differ between countries/regions, so if you include features that depend on paper size, such as printing, be sure to support and test common international sizes.
Record the language of the keyboard or IME.	When your app asks the user for text input, record the language tag for the currently enabled keyboard layout or Input Method Editor (IME). This ensures that when the input is displayed later it's presented to the user with the appropriate formatting. Use the **Language.CurrentInputMethodLanguageTag** property to get the current input language.
Don't use language to assume a user's region; and don't use region to assume a user's language.	Language and region are separate concepts. A user can speak a particular regional variant of a language, like en-GB for English as spoken in the UK, but the user might be in an entirely different country or region. Consider whether your app requires knowledge about the user's language (for UI text, for example), or region (for licensing, for example). For more info, see Understand user profile languages and app manifest languages.
The rules for comparing language tags are non-trivial.	BCP-47 language tags are complex. There are a number of issues when comparing language tags, including issues with matching script information, legacy tags, and multiple regional variants. The Resource Management System in Windows takes care of matching for you. You can specify a set of resources in any languages, and the system chooses the appropriate one for the user and the app. See App resources and the Resource Management System and How the Resource Management System matches language tags.
Design your UI to accommodate different text lengths and font sizes for labels and text input controls.	Strings translated into different languages can vary greatly in length, so you'll need your UI controls to dynamically size to their content. Common characters in other languages include marks above or below what is typically used in English (such as Å or Ṇ). Use the standard font sizes and line heights to provide adequate vertical space. Be aware that fonts for other languages may require larger minimum font sizes to remain legible. See the classes in the Windows.Globalization.Fonts namespace.
Support the mirroring of reading order.	Text alignment and reading order can be left to right (as in English, for example), or right to left (RTL) (as in Arabic or Hebrew, for example). If you are localizing your product into languages that use a different reading order than your own, then be sure that the layout of your UI elements supports mirroring. Even items such as back buttons, UI transition effects, and images might need to be mirrored. For more info, see Adjust layout and fonts, and support RTL.
Display text and fonts correctly.	The ideal font, font size, and direction of text varies between different markets. For more info, see **Adjust layout and fonts, and support RTL** and International fonts.

Important APIs

- Globalization
- GeographicRegion.CurrenciesInUse
- Language.CurrentInputMethodLanguageTag
- Windows.Globalization.Fonts

Related topics

- Recommendations for String Usage
- Globalize your date/time/number formats
- Understand user profile languages and app manifest languages
- BCP-47 language tags
- App resources and the Resource Management System
- How the Resource Management System matches language tags
- Adjust layout and fonts, and support RTL
- International fonts
- Make your app localizable

Samples

- Globalization preferences sample

Understand user profile languages and app manifest languages

Article • 01/22/2023

A Windows user can use **Settings** > **Time & Language** > **Region & language** to configure an ordered list of preferred display languages, or just a single preferred display language. A language can have a regional variant. For example, you can select Spanish as spoken in Spain, Spanish as spoken in Mexico, Spanish as spoken in the United States, among others.

Also in **Settings** > **Time & Language** > **Region & language**, but separate from language, the user can specify their location (known as region) in the world. Note that the display language (and regional variant) setting isn't a determiner of the region setting, and vice versa. For example, a user might be currently living in France but choose a preferred Windows display language of Español (México).

For Windows apps, a language is represented as a BCP-47 language tag. For example, the BCP-47 language tag "en-US" corresponds to English (United States) in **Settings**. Appropriate Windows Runtime APIs accept and return string representations of BCP-47 language tags.

Also see the IANA language subtag registry.

The following three sections define the terms "user profile language list", "app manifest language list", and "app runtime language list". We'll be using these terms in this topic and other topics in this feature area, so it's important to know what they mean.

User profile language list

The user profile language list is the name of the list that's configured by the user in **Settings** > **Time & Language** > **Region & language** > **Languages**. In code you can use the **GlobalizationPreferences.Languages** property to access the user profile language list as a read-only list of strings, where each string is a single BCP-47 language tag such as "en-US" or "ja-JP".

C#

```
    IReadOnlyList<string> userLanguages =
Windows.System.UserProfile.GlobalizationPreferences.Languages;
```

App manifest language list

The app manifest language list is the list of languages for which your app declares (or will declare) support. This list grows as you progress your app through the development lifecycle all the way to localization.

The list is determined at compile time, but you have two options for controlling exactly how that happens. One option is to let Visual Studio determine the list from the files in your project. To do that, first set your app's **Default language** on the **Application** tab in your app package manifest source file (`Package.appxmanifest`). Then, confirm that the same file contains this configuration (which it does by default).

XML

```
<Resources>
  <Resource Language="x-generate" />
</Resources>
```

Each time Visual Studio produces your built app package manifest file (`AppxManifest.xml`), it expands that single `Resource` element in the source file into a union of all the language qualifiers that it finds in your project (see Tailor your resources for language, scale, high contrast, and other qualifiers). For example, if you've begun localizing and you have string, image, and/or file resources whose folder or file names include "en-US", "ja-JP", and "fr-FR", then your built `AppxManifest.xml` file will contain the following (the first entry in the list is the default language that you set).

XML

```
<Resources>
  <Resource Language="EN-US" />
  <Resource Language="JA-JP" />
  <Resource Language="FR-FR" />
</Resources>
```

The other option is to replace that single "x-generate" `<Resource>` element in your app package manifest source file (`Package.appxmanifest`) with the expanded list of `<Resource>` elements (being careful to list the default language first). That option involves more maintenance work for you, but it might be an appropriate option for you if you use a custom build system.

To begin with, your app manifest language list will only contain one language. Perhaps that's en-US. But eventually—as you either manually configure your manifest, or as you add translated resources to your project—that list will grow.

When your app is in the Microsoft Store, the languages in the app manifest language list are the ones that are displayed to customers. For a list of BCP-47 language tags specifically supported by the Microsoft Store, see Supported languages.

In code you can use the **ApplicationLanguages.ManifestLanguages** property to access the app manifest language list as a read-only list of strings, where each string is a single BCP-47 language tag.

```
C#
```

```
    IReadOnlyList<string> userLanguages =
Windows.Globalization.ApplicationLanguages.ManifestLanguages;
```

App runtime language list

The third language list of interest is the intersection between the two lists that we've just described. At runtime, the list of languages for which your app has declared support (the app manifest language list) is compared with the list of languages for which the user has declared a preference (the user profile language list). The app runtime language list is set to this intersection (if the intersection is not empty), or to just the app's default language (if the intersection is empty).

More specifically, the app runtime language list is made up of these items.

1. **(Optional) Primary Language Override**. The **PrimaryLanguageOverride** is a simple override setting for apps that give users their own independent language choice, or apps that have some strong reason to override the default language choices. To learn more, see the Application resources and localization sample .
2. **The user's languages that are supported by the app**. This is the user profile language list filtered by the app manifest language list. Filtering the user's languages by those supported by the app maintains consistency among software development kits (SDKs), class libraries, dependent framework packages, and the app.
3. **If 1 and 2 are empty, then the default or first language supported by the app**. If the user profile language list doesn't contain any languages that the app supports, then the app runtime language is the first language supported by the app.

In code you can use the ResourceContext.QualifierValues property to access the app runtime language list in the form of a string containing a semicolon-delimited list of BCP-47 language tags.

```
C#
```

```
    string runtimeLanguages =
Windows.ApplicationModel.Resources.Core.ResourceContext.GetForCurrentView().
QualifierValues["Language"];
```

You can also access it as a read-only list of strings, each containing a single BCP-47 language tag. You can use the **ResourceContext.Languages** property or the **ApplicationLanguages.Languages** property to do this.

C#

```
    IReadOnlyList<string> runtimeLanguages =
Windows.ApplicationModel.Resources.Core.ResourceContext.GetForCurrentView().
Languages;

    runtimeLanguages = Windows.Globalization.ApplicationLanguages.Languages;
```

The app runtime language list determines the resources that Windows loads for your app and also the language(s) used to format dates, times, numbers, and other components. See Globalize your date/time/number formats.

Note If the user profile language and the app manifest language are regional variants of one another, then the user's regional variant is used as the app runtime language. For example, if the user prefers en-GB and the app supports en-US, then the app runtime language is en-GB. This ensures that dates, times, and numbers are formatted more closely to the user's expectations (en-GB), but localized resources are still loaded (due to language matching) in the app's supported language (en-US).

Qualify resource files with their language

Name your resource files, or their folders, with language resource qualifiers. To learn more about resource qualifiers, see Tailor your resources for language, scale, high contrast, and other qualifiers. A resource file can be an image (or other asset), or it can be a resource container file, such as a *.resw* that contains text strings.

Note Even resources in your app's default language must specify the language qualifier. For example, if your app's default language is English (United States), then qualify your assets as `\Assets\Images\en-US\logo.png`.

- Windows performs complex matching, including across regional variants such as en-US and en-GB. So include the region sub-tag as appropriate. See How the Resource Management System matches language tags.

- Specify a language script sub-tag in the qualifier when there is no Suppress-Script value defined for the language. For example, instead of zh-CN or zh-TW, use zh-Hant, zh-Hant-TW, or zh-Hans (for more detail, see the IANA language subtag registry).
- For languages that have a single standard dialect, there is no need to include the region qualifier. For example, use ja instead of ja-JP.
- Some tools and other components such as machine translators might find specific language tags, such as regional dialect info, helpful in understanding the data.

Not all resources need to be localized

Localization might not be required for all resources.

- At a minimum, ensure all resources exist in the default language.
- A subset of some resources might suffice for a closely related language (partial localization). For example, you might not localize all of your app's UI into Catalan if your app has a full set of resources in Spanish. For users who speak Catalan and then Spanish, the resources that are not available in Catalan appear in Spanish.
- Some resources might require exceptions for specific languages, while the majority of other resources map to a common resource. In this case, mark the resource intended to be used for all languages with the undetermined language tag 'und'. Windows interprets the 'und' language tag as a wildcard (similar to '*') in that it matches the top app language after any other specific match. For example, if a few resources are different for Finnish, but the rest of the resources are the same for all languages, then the Finnish resource should be marked with the Finnish language tag, and the rest should be marked with 'und'.
- For resources that are based on a language script, such as a font or height of text, use the undetermined language tag with a specified script: 'und-<script>'. For example, for Latin fonts use `und-Latn\\fonts.css` and for Cyrillic fonts use `und-Cryl\\fonts.css`.

Set the HTTP Accept-Language request header

Consider whether the web services that you call have the same extent of localization as your app does. HTTP requests made from Windows apps in typical web requests, and XMLHttpRequest (XHR), use the standard HTTP Accept-Language request header. By default, the HTTP header is set to the user profile language list. Each language in the list is further expanded to include neutrals of the language and a weighting (q). For example, a user's language list of fr-FR and en-US results in an HTTP Accept-Language request header of fr-FR, fr, en-US, en ("fr-FR,fr;q=0.8,en-US;q=0.5,en;q=0.3"). But if your

weather app (for example) is displaying a UI in French (France), but the user's top language in their preference list is German, then you'll need to explicitly request French (France) from the service in order to remain consistent within your app.

APIs in the Windows.Globalization namespace

Typically, the APIs in the **Windows.Globalization** namespace use the app runtime language list to determine the language. If none of the languages has a matching format, then the user locale is used. This is the same locale that is used for the system clock. The user locale is available from **Settings** > **Time & Language** > **Region & language** > **Additional date, time, & regional settings** > **Region: Change date, time, or number formats**. The **Windows.Globalization** APIs also have overrides to specify a list of languages to use, instead of the app runtime language list.

Using the **Language** class, you can inspect details about a particular language, such as the script of the language, the display name, and the native name.

Use geographic region when appropriate

In **Settings** > **Time & Language** > **Region & language** > **Country or region**, the user can specify their location in the world. You can use this settings, instead of language, for choosing what content to display to the user. For example, a news app might default to displaying content from this region.

In code, you can access this setting by using the **GlobalizationPreferences.HomeGeographicRegion** property.

Using the **GeographicRegion** class, you can inspect details about a particular region, such as its display name, native name, and currencies in use.

Examples

The following table contains examples of what the user would see in your app's UI under various language and region settings.

Expand table

App manifest language list	User profile language list	App's primary language override (optional)	App runtime language list	What the user sees in the app
English (GB) (default); German (Germany)	English (GB)	none	English (GB)	UI: English (GB) Dates/Times/Numbers: English (GB)
German (Germany) (default); French (France); Italian (Italy)	French (Austria)	none	French (Austria)	UI: French (France) (fallback from French (Austria)) Dates/Times/Numbers: French (Austria)
English (US) (default); French (France); English (GB)	English (Canada); French (Canada)	none	English (Canada); French (Canada)	UI: English (US) (fallback from English (Canada)) Dates/Times/Numbers: English (Canada)
Spanish (Spain) (default); Spanish (Mexico); Spanish (Latin America); Portuguese (Brazil)	English (US)	none	Spanish (Spain)	UI: Spanish (Spain) (uses default since no fallback available for English) Dates/Times/Numbers Spanish (Spain)
Catalan (default); Spanish (Spain); French (France)	Catalan; French (France)	none	Catalan; French (France)	UI: Mostly Catalan and some French (France) because not all the strings are in Catalan Dates/Times/Numbers: Catalan
English (GB) (default); French (France); German (Germany)	German (Germany); English (GB)	English (GB) (chosen by user in app's UI)	English (GB); German (Germany)	UI: English (GB) (language override) Dates/Times/Numbers English (GB)

> ⓘ **Note**
>
> For a list of standard country/region codes used by Microsoft, see the Official Country/Region List.

Important APIs

- GlobalizationPreferences.Languages
- ApplicationLanguages.ManifestLanguages
- PrimaryLanguageOverride
- ResourceContext.QualifierValues
- ResourceContext.Languages
- ApplicationLanguages.Languages
- Windows.Globalization
- Language
- GlobalizationPreferences.HomeGeographicRegion
- GeographicRegion

Related topics

- BCP-47 language tag
- IANA language subtag registry
- Tailor your resources for language, scale, high contrast, and other qualifiers
- Supported languages
- Globalize your date/time/number formats
- How the Resource Management System matches language tags

Samples

- Application resources and localization sample

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Globalize your date/time/number formats

Article • 06/24/2021

Design your app to be global-ready by appropriately formatting dates, times, numbers, phone numbers, and currencies. You'll then be able later to adapt your app for additional cultures, regions, and languages in the global market.

Introduction

When creating your app, if you think more broadly than a single language and culture then you'll have fewer (if any) unexpected issues when your app grows into new markets. For example, dates, times, numbers, calendars, currency, telephone numbers, units of measurement, and paper sizes are all items that can be displayed differently in different cultures or languages.

Different regions and cultures use different date and time formats. These include conventions for the order of day and month in the date, for the separation of hours and minutes in the time, and even for what punctuation is used as a separator. In addition, dates may be displayed in various long formats ("Wednesday, March 28, 2012") or short formats ("3/28/12"), which vary across cultures. And, of course, the names and abbreviations for the days of the week and months of the year differ between languages.

You can preview the formats used for different languages. Go to **Settings** > **Time & Language** > **Region & language**, and click **Additional date, time, & regional settings** > **Change date, time, or number formats**. On the **Formats** tab, select a language from the **Format** drop-down and preview the formats in **Examples**.

This topic uses the terms "user profile language list", "app manifest language list", and "app runtime language list". For details on exactly what those terms mean and how to access their values, see Understand user profile languages and app manifest languages.

Format dates and times for the app runtime language list

If you need to allow users to choose a date, or to select a time, then use the standard calendar, date, and time controls. These automatically use the best date and time format for the app runtime language list.

If you need to display dates or times yourself then you can use the **DateTimeFormatter** class. By default, **DateTimeFormatter** automatically uses the best date and time format for the app runtime language list. So, the code below formats a given **DateTime** in the best way for that list. As an example, assume that your app manifest language list includes English (United States), which is also your default, and German (Germany). If the current date is Nov 6 2017 and the user profile language list contains German (Germany) first, then the formatter gives "06.11.2017". If the user profile language list contains English (United States) first (or if it contains neither English nor German), then the formatter gives "11/6/2017" (since "en-US" matches, or is used as the default).

C#

```
    // Use the DateTimeFormatter class to display dates and times using
basic formatters.

    var shortDateFormatter = new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("shortdate");
    var shortTimeFormatter = new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("shorttime");

    var dateTimeToFormat = DateTime.Now;

    var shortDate = shortDateFormatter.Format(dateTimeToFormat);
    var shortTime = shortTimeFormatter.Format(dateTimeToFormat);

    var results = "Short Date: " + shortDate + "\n" +
                  "Short Time: " + shortTime;
```

You can test the code above on your own PC like this.

- Make sure that you have resource files in your project qualified for both "en-US" and "de-DE" (see Tailor your resources for language, scale, high contrast, and other qualifiers).
- Change your user profile language list in **Settings** > **Time & Language** > **Region & language** > **Languages**. Add German (Germany), make it the default, and run the code again.

Format dates and times for the user profile language list

Remember that, by default, **DateTimeFormatter** matches the app runtime language list. That way, if you display strings such as "The date is <date>", then the language will match the date format.

If for whatever reason you want to format dates and/or times only according to the user profile language list, then you can do that using code like the example below. But if you do so then understand that the user can choose a language for which your app doesn't have translated strings. For example, if your app is not localized into German (Germany), but the user chooses that as their preferred language, then that could result in the display of arguably odd-looking strings such as "The date is 06.11.2017".

C#

```
    // Use the DateTimeFormatter class to display dates and times using
basic formatters.

    var userLanguages =
Windows.System.UserProfile.GlobalizationPreferences.Languages;

    var shortDateFormatter = new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("shortdate",
userLanguages);

    var results = "Short Date: " + shortDateFormatter.Format(DateTime.Now);
```

Format numbers and currencies appropriately

Different cultures format numbers differently. Format differences may include how many decimal digits to display, what characters to use as decimal separators, and what currency symbol to use. Use classes in the **NumberFormatting** namespace to display decimal, percent, or permille numbers, and currencies. Most of the time, you will want these formatter classes to use the best format for the user profile. But you may use the formatters to display a currency for any region or format.

This example shows how to display currencies both per the user profile, and for a specific given currency system.

C#

```
    // This scenario uses the CurrencyFormatter class to format a number as
a currency.

    var userCurrency =
Windows.System.UserProfile.GlobalizationPreferences.Currencies[0];

    var valueToBeFormatted = 12345.67;

    var userCurrencyFormatter = new
Windows.Globalization.NumberFormatting.CurrencyFormatter(userCurrency);
    var userCurrencyValue =
userCurrencyFormatter.Format(valueToBeFormatted);
```

```
    // Create a formatter initialized to a specific currency,
    // in this case US Dollar (specified as an ISO 4217 code)
    // but with the default number formatting for the current user.
    var currencyFormatUSD = new
Windows.Globalization.NumberFormatting.CurrencyFormatter("USD");
    var currencyValueUSD = currencyFormatUSD.Format(valueToBeFormatted);

    // Create a formatter initialized to a specific currency.
    // In this case it's the Euro with the default number formatting for
France.
    var currencyFormatEuroFR = new
Windows.Globalization.NumberFormatting.CurrencyFormatter("EUR", new[] { "fr-
FR" }, "FR");
    var currencyValueEuroFR =
currencyFormatEuroFR.Format(valueToBeFormatted);

    // Results for display.
    var results = "Fixed number (" + valueToBeFormatted + ")\n" +
                  "With user's default currency: " + userCurrencyValue +
"\n" +
                  "Formatted US Dollar: " + currencyValueUSD + "\n" +
                  "Formatted Euro (fr-FR defaults): " +
currencyValueEuroFR;
```

You can test the code above on your own PC by changing the country or region in **Settings** > **Time & Language** > **Region & language** > **Country or region**. Choose a country or region (perhaps Iceland), and run the code again.

Use a culturally appropriate calendar

The calendar differs across regions and languages. The Gregorian calendar is not the default for every region. Users in some regions may choose alternate calendars, such as the Japanese era calendar, or Arabic lunar calendars. Dates and times on the calendar are also sensitive to different time zones and daylight-saving time.

To ensure that the preferred calendar format is used, you can use the standard calendar, date, and time controls. For more complex scenarios, where working directly with operations on calendar dates may be required, **Windows.Globalization** provides a **Calendar** class that gives an appropriate calendar representation for the given culture, region, and calendar type.

Format phone numbers appropriately

Phone numbers are formatted differently across regions. The number of digits, how the digits are grouped, and the significance of certain parts of the phone number vary

between countries/regions. Starting in Windows 10, version 1607, you can use classes in the **PhoneNumberFormatting** namespace to format phone numbers appropriately for the current region.

PhoneNumberInfo parses a string of digits and allows you to: determine whether the digits are a valid phone number in the current region; compare two numbers for equality; and to extract the different functional parts of the phone number, such as country code or geographical area code.

PhoneNumberFormatter formats a string of digits or a **PhoneNumberInfo** for display, even when the string of digits represents a partial phone number. You can use this partial number formatting to format a number as a user is entering the number.

The example below shows how to use **PhoneNumberFormatter** to format a phone number as it is being entered. Each time text changes in a **TextBox** named phoneNumberInputTextBox, the contents of the text box are formatted using the current default region and displayed in a **TextBlock** named phoneNumberOutputTextBlock. For demonstration purposes, the string is also formatted using the region for New Zealand, and displayed in a TextBlock named phoneNumberOutputTextBlockNZ.

C#

```
    using Windows.Globalization.PhoneNumberFormatting;

    PhoneNumberFormatter currentFormatter, NZFormatter;

    public MainPage()
    {
        this.InitializeComponent();

        // Use the default formatter for the current region
        this.currentFormatter = new PhoneNumberFormatter();

        // Create an explicit formatter for New Zealand.
        PhoneNumberFormatter.TryCreate("NZ", out this.NZFormatter);
    }

    private void phoneNumberInputTextBox_TextChanged(object sender,
TextChangedEventArgs e)
    {
        // Format for the default region.
        this.phoneNumberOutputTextBlock.Text =
currentFormatter.FormatPartialString(this.phoneNumberInputTextBox.Text);

        // If the NZFormatter was created successfully, format the partial
string for the NZ TextBlock.
        if(this.NZFormatter != null)
        {
            this.phoneNumberOutputTextBlockNZ.Text =
```

```
this.NZFormatter.FormatPartialString(this.phoneNumberInputTextBox.Text);
        }
    }
```

You can test the code above on your own PC by changing the country or region in **Settings** > **Time & Language** > **Region & language** > **Country or region**. Choose a country or region (perhaps New Zealand to confirm that the formats match), and run the code again. For test data, you can do a web search for the phone number of a business in New Zealand.

The user's language and cultural preferences

For scenarios where you wish to provide different functionality based solely on the user's language, region, or cultural preferences, Windows gives you a way to access those preferences, through **Windows.System.UserProfile.GlobalizationPreferences**. When needed, use the **GlobalizationPreferences** class to get the value of the user's current geographic region, preferred languages, preferred currencies, and so on. But remember that if your app's strings/images aren't localized for the user's preferred language then dates and times and other data formatted for that preferred language won't match the strings that you display.

Important APIs

- DateTimeFormatter
- NumberFormatting
- Calendar
- PhoneNumberFormatting
- GlobalizationPreferences

Related topics

- Calendar, date, and time controls
- Understand user profile languages and app manifest languages
- Tailor your resources for language, scale, high contrast, and other qualifiers

Samples

- Calendar details and math sample
- Date and time formatting sample
- Globalization preferences sample

- Number formatting and parsing sample

Use templates and patterns to format dates and times

Article • 06/24/2021

Use classes in the Windows.Globalization.DateTimeFormatting namespace with custom templates and patterns to display dates and times in exactly the format you wish.

Introduction

The DateTimeFormatter class provides various ways to properly format dates and times for languages and regions around the world. You can use standard formats for year, month, day, and so on. Or you can pass a format template to the *formatTemplate* argument of the **DateTimeFormatter** constructor, such as "longdate" or "month day".

But when you want even more control over the order and format of the components of the DateTime object you wish to display, you can pass a format pattern to the *formatTemplate* argument of the constructor. A format pattern uses a special syntax, which allows you to obtain individual components of a **DateTime** object—just the month name, or just the year value, for example—in order to display them in whatever custom format you choose. Furthermore, the pattern can be localized to adapt to other languages and regions.

Note This is only an overview of format patterns. For a more complete discussion of format templates and format patterns see the Remarks section of the DateTimeFormatter class.

The difference between format templates and format patterns

A format template is a culture-agnostic format string. So, if you construct a **DateTimeFormatter** using a format template, then the formatter displays your format components in the right order for the current language. Conversely, a format pattern is culture-specific. If you construct a **DateTimeFormatter** using a format pattern, then the formatter will use the pattern exactly as given. Consequently, a pattern isn't necessarily valid across cultures.

Let's illustrate this distinction with an example. We'll pass a simple format template (not a pattern) to the **DateTimeFormatter** constructor. This is the format template "month day".

C#

```
var dateFormatter = new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("month day");
```

This creates a formatter based on the language and region value of the current context. The order of the components in a format template doesn't matter; the formatter displays them in the right order for the current language. So, it displays "January 1" for English (United States), "1 janvier" for French (France), and "1月1日" for Japanese.

On the other hand, a format pattern is culture-specific. Let's access the format pattern for our format template.

C#

```
IReadOnlyList<string> monthDayPatterns = dateFormatter.Patterns;
```

This yields different results depending on the runtime language and region. Different regions might use different components, in different orders, with or without additional characters and spacing.

syntax

```
En-US: "{month.full} {day.integer}"
Fr-FR: "{day.integer} {month.full}"
Ja-JP: "{month.integer}月{day.integer}日"
```

In the example above, we inputted a culture-agnostic format string, and we got back a culture-specific format string (which was a function of the language and region that happened to be in effect when we called `dateFormatter.Patterns`). It follows therefore that if you construct a **DateTimeFormatter** from a culture-specific format pattern, then it will only be valid for specific languages/regions.

C#

```
var dateFormatter = new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("{month.full}
{day.integer}");
```

The formatter above returns culture-specific values for the individual components inside the brackets {}. But the order of components in a format pattern is invariant. You get precisely what you ask for, which may or may not be culturally appropriate. This

formatter is valid for English (United States), but not for French (France) nor for Japanese.

syntax

```
En-US: January 1
Fr-FR: janvier 1 (inappropriate for France; non-standard order)
Ja-JP: 1月1 (inappropriate for Japan; the day symbol 日 is missing)
```

Furthermore, a pattern that's correct today might not be correct in the future. Countries or regions might change their calendar systems, which alters a format template. Windows updates the output of formatters based on format templates to accommodate such changes. Therefore, you should only use the pattern syntax under one or more of these conditions.

- You are not dependent on a particular output for a format.
- You do not need the format to follow some culture-specific standard.
- You specifically intend the pattern to be invariant across cultures.
- You intend to localize the actual format pattern string itself.

Here's a summary of the distinction between format templates and format patterns.

Format templates, such as "month day"

- Abstracted representation of a DateTime format that includes values for the month, day, etc., in any order.
- Guaranteed to return a valid standard format across all language-region values supported by Windows.
- Guaranteed to give you a culturally-appropriate formatted string for the given language-region.
- Not all combinations of components are valid. For example, "dayofweek day" is not valid.

Format patterns, such as "{month.full} {day.integer}"

- Explicitly ordered string that expresses the full month name, followed by a space, followed by the day integer, in that order, or whatever specific format pattern you specify.
- May not correspond to a valid standard format for any language-region pair.
- Not guaranteed to be culturally appropriate.
- Any combination of components may be specified, in any order.

Examples

Suppose you wish to display the current month and day together with the current time, in a specific format. For example, you would like US English users to see something like this:

syntax

```
June 25 | 1:38 PM
```

The date part corresponds to the "month day" format template, and the time part corresponds to the "hour minute" format template. So, you can construct formatters for the relevant date and time format templates, and then concatenate their output together using a localizable format string.

C#

```
var dateToFormat = System.DateTime.Now;
var resourceLoader =
Windows.ApplicationModel.Resources.ResourceLoader.GetForCurrentView();

var dateFormatter = new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("month day");
var timeFormatter = new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("hour minute");

var date = dateFormatter.Format(dateToFormat);
var time = timeFormatter.Format(dateToFormat);

string output =
string.Format(resourceLoader.GetString("CustomDateTimeFormatString"), date,
time);
```

`CustomDateTimeFormatString` is a resource identifier referring to a localizable resource in a Resources File (.resw). For a default language of English (United States), this would be set to a value of "{0} | {1}" along with a comment indicating that "{0}" is the date and "{1}" is the time. That way, translators can adjust the format items as needed. For example, they can change the order of the items if it seems more natural in some language or region to have the time precede the date. Or, they can replace "|" with some other separator character.

Another way to implement this example is to query the two formatters for their format patterns, concatenate those together, and then construct a third formatter from the resultant format pattern.

C#

```
var resourceLoader =
Windows.ApplicationModel.Resources.ResourceLoader.GetForCurrentView();

var dateFormatter = new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("month day");
var timeFormatter = new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter("hour minute");

string dateFormatterPattern = dateFormatter.Patterns[0];
string timeFormatterPattern = timeFormatter.Patterns[0];

string pattern =
string.Format(resourceLoader.GetString("CustomDateTimeFormatString"),
dateFormatterPattern, timeFormatterPattern);

var patternFormatter = new
Windows.Globalization.DateTimeFormatting.DateTimeFormatter(pattern);

string output = patternFormatter.Format(System.DateTime.Now);
```

Important APIs

- Windows.Globalization.DateTimeFormatting
- DateTimeFormatter
- DateTime

Related topics

- Date and time formatting sample

Adjust layout and fonts, and support RTL

Article • 06/24/2021

Design your app to support the layouts and fonts of multiple languages, including RTL (right-to-left) flow direction. Flow direction is the direction in which script is written and displayed, and the UI elements on the page are scanned by the eye.

Layout guidelines

Languages such as German and Finnish typically use more characters than English does. East Asian fonts typically require more height. And languages such as Arabic and Hebrew require that layout panels and text elements be laid out in right-to-left (RTL) reading order.

Because of these variations in the metrics of translated text, we recommend that you don't bake absolute positioning, fixed widths, or fixed heights into your user interface (UI). Instead, take advantage of the dynamic layout mechanisms that are built into the Windows UI elements. For example, content controls (such as buttons), items controls (such as grid views and list views), and layout panels (such as grids and stackpanels) automatically resize and reflow by default to fit their content. Pseudo-localize your app to uncover any problematic edge cases where your UI elements don't size to content properly.

Dynamic layout is the recommended technique, and you'll be able to use it in the majority of cases. Less preferable, but still better than baking sizes into your UI markup, is to set layout values as a function of language. Here's an example of how you can expose a Width property in your app as a resource that localizers can set appropriately per language. First, in your app's Resources File (.resw), add a property identifier with the name "TitleText.Width" (instead of "TitleText", you can use any name you like). Then, use an **x:Uid** to associate one or more UI elements with this property identifier.

XAML

```
<TextBlock x:Uid="TitleText">
```

For more info about Resources Files (.resw), property identifiers, and **x:Uid**, see Localize strings in your UI and app package manifest.

Fonts

Use the **LanguageFont** font-mapping class for programmatic access to the recommended font family, size, weight, and style for a particular language. The **LanguageFont** class provides access to the correct font info for various categories of content including UI headers, notifications, body text, and user-editable document body fonts.

Mirroring images

If your app has images that must be mirrored (that is, the same image can be flipped) for RTL, then you can use **FlowDirection**.

XAML

```
<!-- en-US\localized.xaml -->
<Image ... FlowDirection="LeftToRight" />

<!-- ar-SA\localized.xaml -->
<Image ... FlowDirection="RightToLeft" />
```

If your app requires a different image to flip the image correctly, then you can use the resource management system with the `LayoutDirection` qualifier (see the LayoutDirection section of Tailor your resources for language, scale, and other qualifiers). The system chooses an image named `file.layoutdir-rtl.png` when the app runtime language (see Understand user profile languages and app manifest languages) is set to an RTL language. This approach may be necessary when some part of the image is flipped, but another part isn't.

Handling right-to-left (RTL) languages

When your app is localized for right-to-left (RTL) languages, use the **FrameworkElement.FlowDirection** property, and set symmetrical padding and margins. Layout panels such as **Grid** scale and flip automatically with the value of **FlowDirection** that you set.

Set **FlowDirection** on the root layout panel (or frame) of your Page, or on the Page itself. This causes all of the controls contained within to inherit that property.

> ⓘ **Important**

However, **FlowDirection** is *not* set automatically based on the user's selected display language in Windows settings; nor does it change dynamically in response to the user switching display language. If the user switches Windows settings from English to Arabic, for example, then the **FlowDirection** property will *not* automatically change from left-to-right to right-to-left. As the app developer, you have to set **FlowDirection** appropriately for the language that you are currently displaying.

The programmatic technique is to use the `LayoutDirection` property of the preferred user display language to set the **FlowDirection** property (see the code example below). Most controls included in Windows use **FlowDirection** already. If you're implementing a custom control, it should use **FlowDirection** to make appropriate layout changes for RTL and LTR languages.

C#

```
this.languageTag = Windows.Globalization.ApplicationLanguages.Languages[0];

// For bidirectional languages, determine flow direction for the root layout
panel, and all contained UI.

var flowDirectionSetting =
Windows.ApplicationModel.Resources.Core.ResourceContext.GetForCurrentView().
QualifierValues["LayoutDirection"];
if (flowDirectionSetting == "LTR")
{
    this.layoutRoot.FlowDirection =
Windows.UI.Xaml.FlowDirection.LeftToRight;
}
else
{
    this.layoutRoot.FlowDirection =
Windows.UI.Xaml.FlowDirection.RightToLeft;
}
```

C++/WinRT

```
#include <winrt/Windows.ApplicationModel.Resources.Core.h>
#include <winrt/Windows.Globalization.h>
...

m_languageTag =
Windows::Globalization::ApplicationLanguages::Languages().GetAt(0);

// For bidirectional languages, determine flow direction for the root layout
panel, and all contained UI.

auto flowDirectionSetting =
```

```
Windows::ApplicationModel::Resources::Core::ResourceContext::GetForCurrentVi
ew().QualifierValues().Lookup(L"LayoutDirection");
if (flowDirectionSetting == L"LTR")
{

layoutRoot().FlowDirection(Windows::UI::Xaml::FlowDirection::LeftToRight);
}
else
{

layoutRoot().FlowDirection(Windows::UI::Xaml::FlowDirection::RightToLeft);
}
```

C++

```
this->languageTag = Windows::Globalization::ApplicationLanguages::Languages-
>GetAt(0);

// For bidirectional languages, determine flow direction for the root layout
panel, and all contained UI.

auto flowDirectionSetting =
Windows::ApplicationModel::Resources::Core::ResourceContext::GetForCurrentVi
ew()->QualifierValues->Lookup("LayoutDirection");
if (flowDirectionSetting == "LTR")
{
    this->layoutRoot->FlowDirection =
Windows::UI::Xaml::FlowDirection::LeftToRight;
}
else
{
    this->layoutRoot->FlowDirection =
Windows::UI::Xaml::FlowDirection::RightToLeft;
}
```

The technique above makes **FlowDirection** a function of the `LayoutDirection` property of the preferred user display language. If for whatever reason you don't want that logic, then you can expose a FlowDirection property in your app as a resource that localizers can set appropriately for each language they translate into.

First, in your app's Resources File (.resw), add a property identifier with the name "MainPage.FlowDirection" (instead of "MainPage", you can use any name you like). Then, use an **x:Uid** to associate your main **Page** element (or its root layout panel or frame) with this property identifier.

XAML

```
<Page x:Uid="MainPage">
```

Instead of a single line of code for all languages, this depends on the translator "translating" this property resource correctly for each translated language; so be aware that there's that extra opportunity for human error when you use this technique.

Important APIs

- FrameworkElement.FlowDirection
- LanguageFont

Related topics

- Localize strings in your UI and app package manifest
- Tailor your resources for language, scale, and other qualifiers
- Understand user profile languages and app manifest languages

NumeralSystem values

Article • 06/24/2021

This topic lists the values available to the **NumeralSystem** property of various classes in the Windows.Globalization namespace. These classes contain a **NumeralSystem** property.

- Windows.Globalization.Calendar
- Windows.Globalization.NumberFormatting.CurrencyFormatter
- Windows.Globalization.NumberFormatting.DecimalFormatter
- Windows.Globalization.NumberFormatting.NumeralSystemTranslator
- Windows.Globalization.NumberFormatting.PercentFormatter
- Windows.Globalization.NumberFormatting.PermilleFormatter
- Windows.Globalization.NumberFormatting.INumberFormatterOptions

The **NumeralSystem** property indicates the numeral system used by the class. The property is a string such as "Latn" for the Latin numeral system (0123456789), or "Arab" for the Arabic-Indic numeral system (٠١٢٣٤٥٦٧٨٩).

These are the values for **NumeralSystem** that are supported in Windows.

NumeralSystem value	Description	Unicode code points
Arab	Arabic-Indic	U+0660, U+0661, U+0662, U+0663, U+0664, U+0665, U+0666, U+0667, U+0668, U+0669
ArabExt	Extended Arabic-Indic	U+06F0, U+06F1, U+06F2, U+06F3, U+06F4, U+06F5, U+06F6, U+06F7, U+06F8, U+06F9
Bali	Balinese	U+1B50, U+1B51, U+1B52, U+1B53, U+1B54, U+1B55, U+1B56, U+1B57, U+1B58, U+1B59
Beng	Bengali	U+09E6, U+09E7, U+09E8, U+09E9, U+09EA, U+09EB, U+09EC, U+09ED, U+09EE, U+09EF
Cham	Cham	U+AA50, U+AA51, U+AA52, U+AA53, U+AA54, U+AA55, U+AA56, U+AA57, U+AA58, U+AA59
Deva	Devanagari	U+0966, U+0967, U+0968, U+0969, U+096A, U+096B, U+096C, U+096D, U+096E, U+096F
FullWide	Full width	U+FF10, U+FF11, U+FF12, U+FF13, U+FF14, U+FF15, U+FF16, U+FF17, U+FF18, U+FF19

NumeralSystem value	Description	Unicode code points
Gujr	Gujarati	U+0AE6, U+0AE7, U+0AE8, U+0AE9, U+0AEA, U+0AEB, U+0AEC, U+0AED, U+0AEE, U+0AEF
Guru	Gurmukhi	U+0A66, U+0A67, U+0A68, U+0A69, U+0A6A, U+0A6B, U+0A6C, U+0A6D, U+0A6E, U+0A6F
Java	Javanese	U+A9D0, U+A9D1, U+A9D2, U+A9D3, U+A9D4, U+A9D5, U+A9D6, U+A9D7, U+A9D8, U+A9D9
Kali	Kayah Li	U+A900, U+A901, U+A902, U+A903, U+A904, U+A905, U+A906, U+A907, U+A908, U+A909
Khmr	Khmer	U+17E0, U+17E1, U+17E2, U+17E3, U+17E4, U+17E5, U+17E6, U+17E7, U+17E8, U+17E9
Knda	Kannada	U+0CE6, U+0CE7, U+0CE8, U+0CE9, U+0CEA, U+0CEB, U+0CEC, U+0CED, U+0CEE, U+0CEF
Lana	Tai Tham Hora	U+1A80, U+1A81, U+1A82, U+1A83, U+1A84, U+1A85, U+1A86, U+1A87, U+1A88, U+1A89
LanaTham	Tai Tham Tham	U+1A90, U+1A91, U+1A92, U+1A93, U+1A94, U+1A95, U+1A96, U+1A97, U+1A98, U+1A99
Laoo	Lao	U+0ED0, U+0ED1, U+0ED2, U+0ED3, U+0ED4, U+0ED5, U+0ED6, U+0ED7, U+0ED8, U+0ED9
Latn	Latin	U+0030, U+0031, U+0032, U+0033, U+0034, U+0035, U+0036, U+0037, U+0038, U+0039
Lepc	Lepcha	U+1C40, U+1C41, U+1C42, U+1C43, U+1C44, U+1C45, U+1C46, U+1C47, U+1C48, U+1C49
Limb	Limbu	U+1946, U+1947, U+1948, U+1949, U+194A, U+194B, U+194C, U+194D, U+194E, U+194F
Mlym	Malayalam	U+0D66, U+0D67, U+0D68, U+0D69, U+0D6A, U+0D6B, U+0D6C, U+0D6D, U+0D6E, U+0D6F
Mong	Mongolian	U+1810, U+1811, U+1812, U+1813, U+1814, U+1815, U+1816, U+1817, U+1818, U+1819
Mtei	Meetei Mayek	U+ABF0, U+ABF1, U+ABF2, U+ABF3, U+ABF4, U+ABF5, U+ABF6, U+ABF7, U+ABF8, U+ABF9
Mymr	Myanmar	U+1040, U+1041, U+1042, U+1043, U+1044, U+1045, U+1046, U+1047, U+1048, U+1049

NumeralSystem value	Description	Unicode code points
MymrShan	Myanmar Shan	U+1090, U+1091, U+1092, U+1093, U+1094, U+1095, U+1096, U+1097, U+1098, U+1099
Nkoo	Nko	U+07C0, U+07C1, U+07C2, U+07C3, U+07C4, U+07C5, U+07C6, U+07C7, U+07C8, U+07C9
Olck	Ol Chiki	U+1C50, U+1C51, U+1C52, U+1C53, U+1C54, U+1C55, U+1C56, U+1C57, U+1C58, U+1C59
Orya	Odia	U+0B66, U+0B67, U+0B68, U+0B69, U+0B6A, U+0B6B, U+0B6C, U+0B6D, U+0B6E, U+0B6F
Saur	Saurashtra	U+A8D0, U+A8D1, U+A8D2, U+A8D3, U+A8D4, U+A8D5, U+A8D6, U+A8D7, U+A8D8, U+A8D9
Sund	Sundanese	U+1BB0, U+1BB1, U+1BB2, U+1BB3, U+1BB4, U+1BB5, U+1BB6, U+1BB7, U+1BB8, U+1BB9
Talu	New Tai Lue	U+19D0, U+19D1, U+19D2, U+19D3, U+19D4, U+19D5, U+19D6, U+19D7, U+19D8, U+19D9
TamlDec	Tamil	U+0BE6, U+0BE7, U+0BE8, U+0BE9, U+0BEA, U+0BEB, U+0BEC, U+0BED, U+0BEE, U+0BEF
Telu	Telugu	U+0C66, U+0C67, U+0C68, U+0C69, U+0C6A, U+0C6B, U+0C6C, U+0C6D, U+0C6E, U+0C6F
Thai	Thai	U+0E50, U+0E51, U+0E52, U+0E53, U+0E54, U+0E55, U+0E56, U+0E57, U+0E58, U+0E59
Tibt	Tibetan	U+0F20, U+0F21, U+0F22, U+0F23, U+0F24, U+0F25, U+0F26, U+0F27, U+0F28, U+0F29
Vaii	Vai	U+A620, U+A621, U+A622, U+A623, U+A624, U+A625, U+A626, U+A627, U+A628, U+A629

Make your app localizable

Article • 06/24/2021

A localized app is one that can be localized for other markets, languages, or regions without uncovering any functional defects in the app. The most essential property of a localizable app is that its executable code has been cleanly separated from its localizable resources. So, you should determine which of your app's resources need to be localized. Ask yourself what needs to change if your app is to be localized for other markets.

We also recommend that you become familiar with the guidelines for globalization.

Put your strings into Resources Files (.resw)

Don't hard-code string literals in your imperative code, XAML markup, nor in your app package manifest. Instead, put your strings into Resources Files (.resw) so that they can be adapted to different local markets independently of your app's built binaries. For details, see Localize strings in your UI and app package manifest.

That topic also shows you how to add comments to your default Resources File (.resw). For example, if you are adopting an informal voice or tone then be sure to explain that in comments. Also, to minimize expense, confirm that only the strings that need to be translated are provided to translators.

Set the default language for your app appropriately in your app package manifest source file (the `Package.appxmanifest` file). The default language determines the language that's used when the user's preferred languages don't match any of the supported languages of your app. Mark all of your resources with their language (even the ones in your default language, for example `\Assets\en-us\Logo.png`) so that the system can tell which language the resource is in and how it's used in particular situations.

Tailor your images and other file resources for language

Ideally, you will be able to globalize your images—that is, make them culture-independent. For any images and other file resources where that's not possible, create as many different variants of them as you need and put the appropriate language qualifiers into their file or folder names. To learn more, see Tailor your resources for language, scale, high contrast, and other qualifiers.

To minimize localization costs, don't put text nor culturally-sensitive material into images to begin with. An image that's appropriate in your own culture might be offensive or misinterpreted in other cultures. Avoid the use of culture-specific images such as mailboxes, which are not common around the world. Avoid religious symbols, animals, political, or gender-specific images. The display of flesh, body parts, or hand gestures can also be a sensitive topic. If you can't avoid all of these, then your images will need to be thoughtfully localized. If you're localizing to a language with a different reading direction than your own, using symmetrical images and effects make it easier to support mirroring.

Also avoid the use of text in images, and speech in audio/video files.

The use of color in your app

Be mindful when using color. Using color combinations that are associated with national flags or political movements can be problematic. Color choices may need to be reviewed by culture experts. There is also an accessibility issues with using color. If you use color to convey meaning then you should also convey that same information by some other means, such as size, shape, or a label.

Consider factoring your strings into sentences

Use appropriately-sized strings. Short strings are easier to translate, and they enable translation recycling (which saves expense because the same string isn't sent to the localizer more than once). Also, extremely long strings might not be supported by localization tools.

But in tension with this guideline is the risk of re-using a string in different contexts. Even simple words such as "on" and "off" might be translated differently, depending on the context. In the English language, "on" and "off" can be used for a toggle for Flight Mode, Bluetooth, and devices. But in Italian, the translation depends on the context of what is being turned on and off. You would need to create a pair of strings for each context. You can reuse strings if the two contexts are the same. For instance, you can reuse the string "Volume" for both sound effect volume and music volume because both refer to intensity of sound. You should not reuse that same string when referring to a hard disk volume because the context and meaning are different, and the word might be translated differently.

Additionally, a string like "text" or "fax" could be used as both a verb and a noun in the English language, which can confuse the translation process. Instead, create a separate

string for both the verb and noun format. When you're not sure whether the contexts are the same, err on the safe side and use a distinct string.

In short, factor your strings into pieces that work in all contexts. There will be cases where a string will need to be an entire sentence.

Consider the following string: "The {0} could not be synchronized."

A variety of words could replace {0}, such as "appointment", "task", or "document". While this example works for the English language, it will not work in all cases for the corresponding sentence in, for example, German. Notice that in the following German sentences, some of the words in the template string ("Der", "Die", "Das") need to match the parameterized word:

English	German
The appointment could not be synchronized.	Der Termin konnte nicht synchronisiert werden.
The task could not be synchronized.	Die Aufgabe konnte nicht synchronisiert werden.
The document could not be synchronized.	Das Dokument konnte nicht synchronisiert werden.

As another example, consider the sentence "Remind me in {0} minute(s)." Using "minute(s)" works for the English language, but other languages might use different terms. For example, the Polish language uses "minuta", "minuty", or "minut" depending on the context.

To solve this problem, localize the entire sentence, rather than a single word. Doing this may seem like extra work and an inelegant solution, but it is the best solution because:

- A grammatically correct message will be displayed for all languages.
- Your translator will not need to ask about what the strings will be replaced with.
- You will not need to implement a costly code fix when a problem like this surfaces after your app is completed.

Other considerations for strings

Avoid colloquialisms and metaphors in the strings that you author in your default language. Language that's specific to a demographic group, such as culture and age, can be hard to understand or translate because only people in that demographic group use that language. Similarly, metaphors might make sense to one person but mean nothing to someone else. For example, a "bluebird" means something specific to those

who are part of skiing culture, but those who aren't part of that culture don't understand the reference.

Don't use technical jargon, abbreviations, or acronyms. Technical language is less likely to be understood by non-technical audiences or people from other cultures or regions, and it's difficult to translate. People don't use these kinds of words in everyday conversations. Technical language often appears in error messages to identify hardware and software issues, but you should strings to be technical *only if the user needs that level of information, and can either action it or find someone who can*.

Using an informal voice or tone in your strings is a valid choice. You can use comments in your default Resources File (.resw) to indicate that intention.

Pseudo-localization

Pseudo-localize your app to uncover any localizability issues. Pseudo-localization is a kind of localization dry-run, or disclosure test. You produce a set of resources that are not really translated; they only look that way. Your strings are approximately 40% longer than in the default language, for example, and they have delimiters in them so that you can see at a glance whether they have been truncated in the UI.

Deployment Considerations

When you install an app that contains localized language data, you might find that only the default language is available for the app even though you initially included resources for multiple languages. This is because the installation process is optimized to only install language resources that match the current language and culture of the device. Therefore, if your device is configured for en-US, only the en-US language resources are installed with your app.

> **Note**
>
> It is not possible to install additional language support for your app after the initial installation. If you change the default language after installing an app, the app continues to use only the original language resources.

If you want to ensure all language resources are available after installation, create a configuration file for the app package that specifies that certain resources are required during installation (including language resources). This optimized installation feature is automatically enabled when your application's .appxbundle is generated during

packaging. For more information, see Ensure that resources are installed on a device regardless of whether a device requires them.

Optionally, to ensure all resources are installed (not just a subset), you can disable .appxbundle generation when you package your app. This is not recommended however as it can increase the installation time of your app.

Disable automatic generation of the .appxbundle by setting the "Generate App Bundle" attribute to “never”:

1. In Visual Studio, right-click the project name
2. Select **Store** -> **Create app packages...**
3. In the **Create Your Packages** dialog, select **I want to create packages to upload to the Microsoft Store using a new app name** and then click **Next**.
4. In the **Select an app name** dialog, select/create an app name for your package.
5. In the **Select and Configure Packages** dialog, set **Generate app bundle** to **Never**.

Geopolitical awareness

Avoid political offense in maps or when referring to regions. Maps might include controversial regional or national boundaries, and they're a frequent source of political offense. Be careful that any UI used for selecting a nation refers to it as a "country/region". Listing a disputed territory in a list labeled "countries"—such as in an address form—might offend some users.

Language- and region-changed events

Subscribe to events that are raised when the system's language and region settings change. Do this so that you can re-load resources, if appropriate. For details, see Updating strings in response to qualifier value change events and Updating images in response to qualifier value change events.

Ensure the correct parameter order when formatting strings

Don't assume that all languages express parameters in the same order. For example, consider this format.

```
C#
```

```
    string.Format("Every {0} {1}", monthName, dayNumber); // For example,
"Every April 1".
```

The format string in this example works for English (United States). But it is not appropriate for German (Germany), for example, where the day and month are displayed in the reverse order. Ensure that the translator knows the intent of each of the parameters so that they can reverse the order of the format items in the format string (for example, "{1} {0}") as appropriate for the target language.

Don’t over-localize

Only submit natural language to translators; not programming language nor markup. A `<link>` tag is not natural language. Consider these examples.

Don't localize this	Localize this
<link>terms of use</link>	terms of use
<link>privacy policy</link>	privacy policy

Including the `<link>` tag in your Resources File (.resw) means that it, too, is likely to be translated. That would render the tag invalid. If you have long strings that need to include markup in order to maintain context and ensure ordering, then make it clear in comments what not to translate.

Choose an appropriate translation approach

After strings are separated into resource files, they can be translated. The ideal time to translate strings is after the strings in your project are finalized, which usually happens toward the end of a project. You can approach the translation process in number of ways. This may depend on the volume of strings to be translated, the number of languages to be translated, and how the translation will be done (such as in-house versus hiring an external vendor).

Consider these options.

- **The resource files can be translated by opening them directly in the project.** This approach works well for a project that has a small volume of strings that need to be translated into two or three languages. It could be suitable for a scenario where a developer speaks more than one language and is willing to handle the translation process. This approach benefits from being quick, requires no tools,

and minimizes the risk of mistranslations. But it is not scalable. In particular, the resources in different languages can easily get out of sync, causing bad user experiences and maintenance headaches.

- **The string resource files are in XML or ResJSON text format, so could be handed off for translation using any text editor. The translated files would then be copied back into the project.** This approach carries a risk of translators accidentally editing the XML tags, but it lets translation work take place outside of the Microsoft Visual Studio project. This approach could work well for projects that need to be translated into a small number of languages. The XLIFF format is an XML format specifically designed for use in localization, and should be well supported by some localization vendors or localization tools. You can use the Multilingual App Toolkit to generate XLIFF files from other resource files, such as .resw or .resjson.

> **Note**
>
> Localization might also be necessary for other assets, including images and audio files.

You should also consider the following:

- **Localization tools** A number of localization tools are available for parsing resource files and allowing only the translatable strings to be edited by translators. This approach reduces the risk of a translator accidentally editing the XML tags. But it has the drawback of introducing a new tool and process to the localization process. A localization tool is good for projects with a large volume of strings but a small number of languages. To learn more, see How to use the Multilingual App Toolkit.
- **Localization vendors** Consider using a localization vendor if your application contains extensive strings that need to be translated into a large number of languages. A localization vendor can give advice about tools and processes, as well as translating your resource files. This is an ideal solution, but is also the most costly option, and may increase the turnaround time for your translated content.

Keep access keys and labels consistent

It is a challenge to "synchronize" the access keys used in accessibility with the display of the localized access keys, because the two string resources are categorized in two separate sections. Be sure to provide comments for the label string such as: `Make sure that the emphasized shortcut key is synchronized with the access key.`

Support furigana for Japanese strings that can be sorted

Japanese kanji characters have the property of having more than one reading (pronunciation) depending on the word in which they are used. This leads to problems when you try to sort Japanese named objects, such as application names, files, songs, and so on. Japanese kanji have, in the past, usually been sorted in a machine-understandable order called XJIS. Unfortunately, because this sorting order is not phonetic it is not very useful for humans.

Furigana works around this problem by allowing the user or creator to specify the phonetics for the characters they are using. If you use the following procedure to add furigana to your app name, you can ensure that it is sorted in the proper location in the app list. If your app name contains kanji characters and furigana is not provided when the user's UI language or the sort order is set to Japanese, Windows makes its best effort to generate the appropriate pronunciation. However, there is a possibility for app names containing rare or unique readings to be sorted under a more common reading instead. Therefore, the best practice for Japanese applications (especially those containing kanji characters in their names) is to provide a furigana version of their app name as part of the Japanese localization process.

1. Add "ms-resource:Appname" as the Package Display Name and the Application Display Name.

2. Create a ja-JP folder under strings, and add two resource files as follows:

   syntax

   ```
   strings\
       en-us\
       ja-jp\
           Resources.altform-msft-phonetic.resw
           Resources.resw
   ```

3. In Resources.resw for general ja-JP: Add a string resource for Appname "希蒼"

4. In Resources.altform-msft-phonetic.resw for Japanese furigana resources: Add furigana value for AppName "のあ"

The user can search for the app name "希蒼" using both the furigana value "のあ" (noa), and the phonetic value (using the **GetPhonetic** function from the Input Method Editor (IME)) "まれあお" (mare-ao).

Sorting follows the **Regional Control Panel** format:

- Under a Japanese user locale,
  - If furigana is enabled, then "希蒼" is sorted under "の".
  - If furigana is missing, then "希蒼" is sorted under "ま".
- Under a non-Japanese user locale,
  - If furigana is enabled, then "希蒼" is sorted under "の".
  - If furigana is missing, then "希蒼" is sorted under "漢字".

Related topics

- Guidelines for globalization
- Localize strings in your UI and app package manifest
- Tailor your resources for language, scale, high contrast, and other qualifiers
- Adjust layout and fonts, and support RTL
- Updating images in response to qualifier value change events

Samples

- Application resources and localization sample

International fonts

Article • 06/23/2023

This topic lists the fonts available for Windows apps that are localized into languages other than U.S. English.

In the following table, the "Language tag" column lists the BCP-47 language tag of the language. To see a list of the language tags specifically supported by the Microsoft Store, see Supported languages.

Expand table

Language	Language tag	Font used in Windows
Afrikaans	af-ZA	Segoe UI
Albanian	sq-AL	Segoe UI
Amharic	am-ET	Ebrima
Arabic	ar-SA	Segoe UI
Armenian	hy-AM	Segoe UI
Assamese	as-IN	Nirmala UI
Azerbaijani (Latin)	az-Latn-AZ	Segoe UI
Bangla (Bangladesh)	bn-BD	Nirmala UI
Bangla (India)	bn-IN	Nirmala UI
Basque (Basque)	eu-ES	Segoe UI
Belarusian	be-BY	Segoe UI
Bosnian (Latin)	bs-Latn-BA	Segoe UI
Bulgarian	bg-BG	Segoe UI
Catalan	ca-ES	Segoe UI
Central Kurdish (Arabic)	ku-ARAB-IQ	Segoe UI
Cherokee (Cherokee)	chr-CHER-US	Gadugi
Chinese Simplified	zh-CN	Microsoft YaHei UI

Language	Language tag	Font used in Windows
Chinese Traditional (Hong Kong Special Administrative Region)	zh-HK	Microsoft JhengHei UI
Chinese Traditional (Taiwan)	zh-TW	Microsoft JhengHei UI
Croatian	hr-HR	Segoe UI
Czech	cs-CZ	Segoe UI
Danish	da-DK	Segoe UI
Dari	prs-AF	Segoe UI
Dutch	nl-NL	Segoe UI
English (United Kingdom)	en-GB	Segoe UI
Estonian	et-EE	Segoe UI
Filipino	fil-PH	Segoe UI
Finnish	fi-FI	Segoe UI
French	fr-FR	Segoe UI
Galician	gl-ES	Segoe UI
Georgian	ka-GE	Segoe UI
German	de-DE	Segoe UI
Greek	el-GR	Segoe UI
Gujarati	gu-IN	Nirmala UI
Hausa (Latin)	ha-Latn-NG	Segoe UI
Hebrew	he-IL	Segoe UI
Hindi	hi-IN	Nirmala UI
Hungarian	hu-HU	Segoe UI
Icelandic	is-IS	Segoe UI
Igbo	ig-NG	Segoe UI
Indonesian	id-ID	Segoe UI

Language	Language tag	Font used in Windows
Irish	ga-IE	Segoe UI
isiXhosa	xh-ZA	Segoe UI
isiZulu	zu-ZA	Segoe UI
Italian	it-IT	Segoe UI
Japanese	ja-JP	Yu Gothic UI
Kannada	kn-IN	Nirmala UI
Kazakh	kk-KZ	Segoe UI
Khmer	km-KH	Leelawadee UI
K'iche'	qut-GT	Segoe UI
Kinyarwanda	rw-RW	Segoe UI
KiSwahili	sw-KE	Segoe UI
Konkani	kok-IN	Nirmala UI
Korean	ko-KR	Malgun Gothic
Kyrgyz	ky-KG	Segoe UI
Latvian	lv-LV	Segoe UI
Lithuanian	lt-LT	Segoe UI
Luxembourgish	lb-LU	Segoe UI
Macedonian	mk-MK	Segoe UI
Malay	ms-MY	Segoe UI
Malayalam	ml-IN	Nirmala UI
Maltese	mt-MT	Segoe UI
Maori	mi-NZ	Segoe UI
Marathi	mr-IN	Nirmala UI
Mongolian (Cyrillic)	mn-MN	Segoe UI
Nepali	ne-NP	Nirmala UI
Norwegian (Bokmal)	nb-NO	Segoe UI

Language	Language tag	Font used in Windows
Norwegian (Nynorsk)	nn-NO	Segoe UI
Odia	or-IN	Nirmala UI
Persian	fa-IR	Segoe UI
Polish	pl-PL	Segoe UI
Portuguese (Brazil)	pt-BR	Segoe UI
Portuguese (Portugal)	pt-PT	Segoe UI
Punjabi	pa-IN	Nirmala UI
Punjabi (Arabic)	pa-Arab-PK	Segoe UI
Quechua	quz-PE	Segoe UI
Romanian	ro-RO	Segoe UI
Russian	ru-RU	Segoe UI
Scottish Gaelic	gd-GB	Segoe UI
Serbian (Cyrillic, Bosnia and Herzegovina)	sr-Cyrl-BA	Segoe UI
Serbian (Cyrillic, Serbia)	sr-Cyrl-CS	Segoe UI
Serbian (Latin)	sr-Latn-CS	Segoe UI
Sesotho sa Leboa	nso-ZA	Segoe UI
Setswana	tn-ZA	Segoe UI
Sindhi (Arabic)	sd-Arab-PK	Segoe UI
Sinhala	si-LK	Nirmala UI
Slovak	sk-SK	Segoe UI
Slovenian	sl-SI	Segoe UI
Spanish	es-ES	Segoe UI
Swedish	sv-SE	Segoe UI
Tajik (Cyrillic)	tg-Cyrl-TJ	Segoe UI
Tamil	ta-IN	Nirmala UI
Tatar	tt-RU	Segoe UI

Language	Language tag	Font used in Windows
Telugu	te-IN	Nirmala UI
Thai	th-TH	Leelawadee UI
Tigrinya	ti-ET	Ebrima
Turkish	tr-TR	Segoe UI
Turkmen	tk-TM	Segoe UI
Ukrainian	uk-UA	Segoe UI
Urdu	ur-PK	Segoe UI
Uyghur	ug-CN	Segoe UI
Uzbek (Latin)	uz-Latn-UZ	Segoe UI
Valencian	ca-ES-valencia	Segoe UI
Vietnamese	vi-VN	Segoe UI
Welsh	cy-GB	Segoe UI
Wolof	wo-SN	Segoe UI
Yoruba	yo-NG	Segoe UI

Related topics

- BCP-47 language tag
- Supported languages

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Design your app for bidirectional text

Article • 05/13/2022

Design your app to provide bidirectional text support (BiDi) so that you can combine script from right to left (RTL) and left to right (LTR) writing systems, which generally contain different types of alphabets.

Right-to-left writing systems, such as those used in the Middle East, Central and South Asia, and in Africa, have unique design requirements. These writing systems require bidirectional text support (BiDi). BiDi support is the ability to input and display text layout in either right to left (RTL) or left to right (LTR) order.

A total of nine BiDi languages are included with Windows.

- Two fully localized languages. Arabic, and Hebrew.
- Seven Language Interface Packs for emerging markets. Persian, Urdu, Dari, Central Kurdish, Sindhi, Punjabi (Pakistan), and Uyghur.

This topic contains the Windows BiDi design philosophy, and case studies that show BiDi design considerations.

Bidi design elements

Four elements influence BiDi design decisions in Windows.

- **User interface (UI) mirroring**. User interface (UI) flow allows right-to-left content to be presented in its native layout. UI design feels local to BiDi markets.
- **Consistency in user experience**. The design feels natural in right-to-left orientation. UI elements share a consistent layout direction and appear as the user expects them.
- **Touch optimization**. Similar to touch UI in non-mirrored UI, elements are easy to reach, and they natural to touch interaction.
- **Mixed text support**. Text directionality support enables great mixed text presentation (English text on BiDi builds, and vice versa).

Feature design overview

Windows supports the four BiDi design elements. Let's look at some of the major relevant features of (prior versions of) Windows, and provide some context around how they affect your app.

Navigate in the direction that feels natural

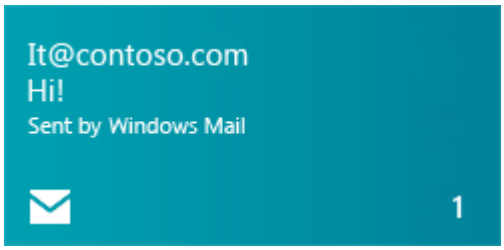
Windows adjusts the direction of the typographic grid so that it flows from right to left, meaning that the first tile on the grid is placed at the top right corner, and the last tile at the bottom left. This matches the RTL pattern of printed publications such as books and magazines, where the reading pattern always starts at the top right corner and progresses to the left.

To preserve a consistent UI flow, content on tiles retain a right-to-left layout, meaning that the app name and logo are positioned at the bottom right corner of the tile regardless of the app UI language.

BiDi tile

1 Notification Region
2 Notification Badge
3 App Name/Logo

English tile

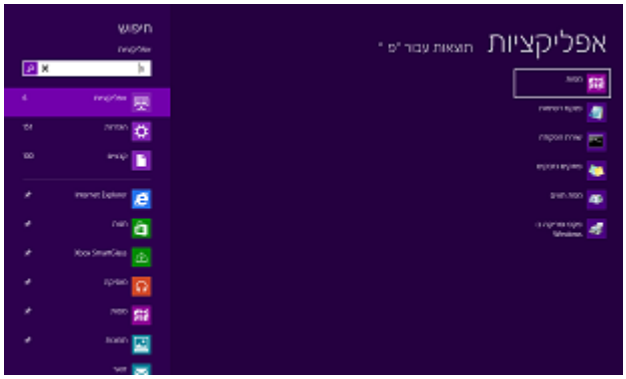


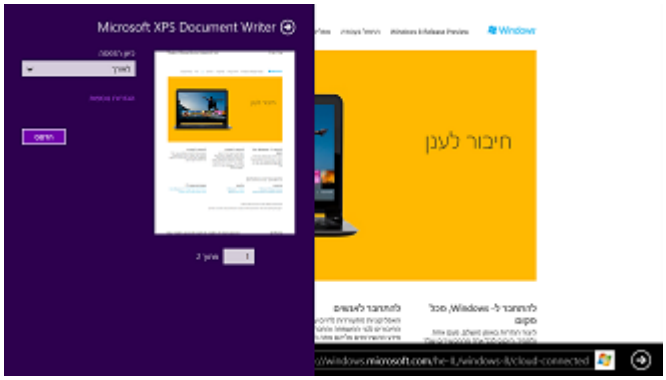
Get tile notifications that read correctly

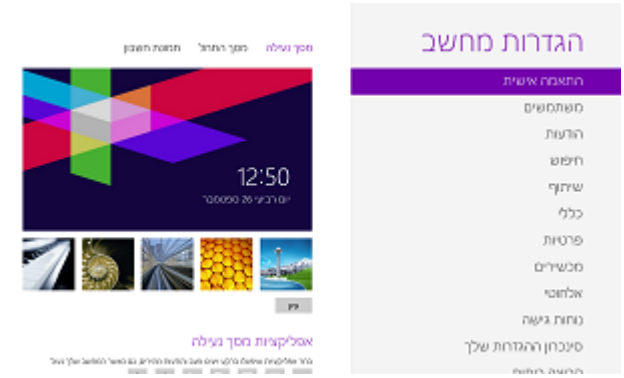
Tiles have mixed text support. The notification region has built-in flexibility to adjust the text alignment based on the notification language. When an app sends Arabic, Hebrew, or other BiDi language notifications, the text is aligned to the right. And when an English (or other LTR) notification arrives, it will align to the left.

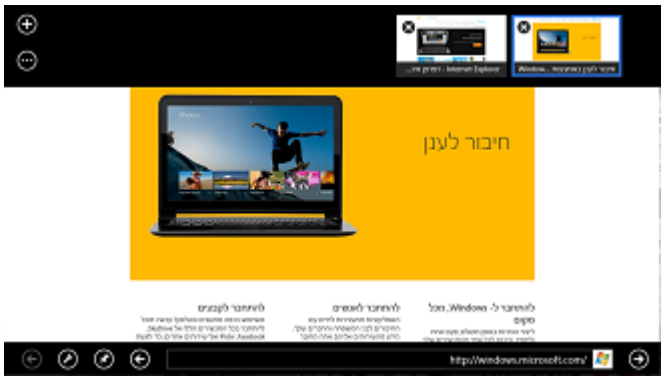
A consistent, easy-to-touch RTL user experience

Every element in the Windows UI fits into the RTL orientation. Charms and flyouts have been positioned on the left edge of the screen so that they don't overlap search results or degrade touch optimization. They can be easily reached with the thumbs.



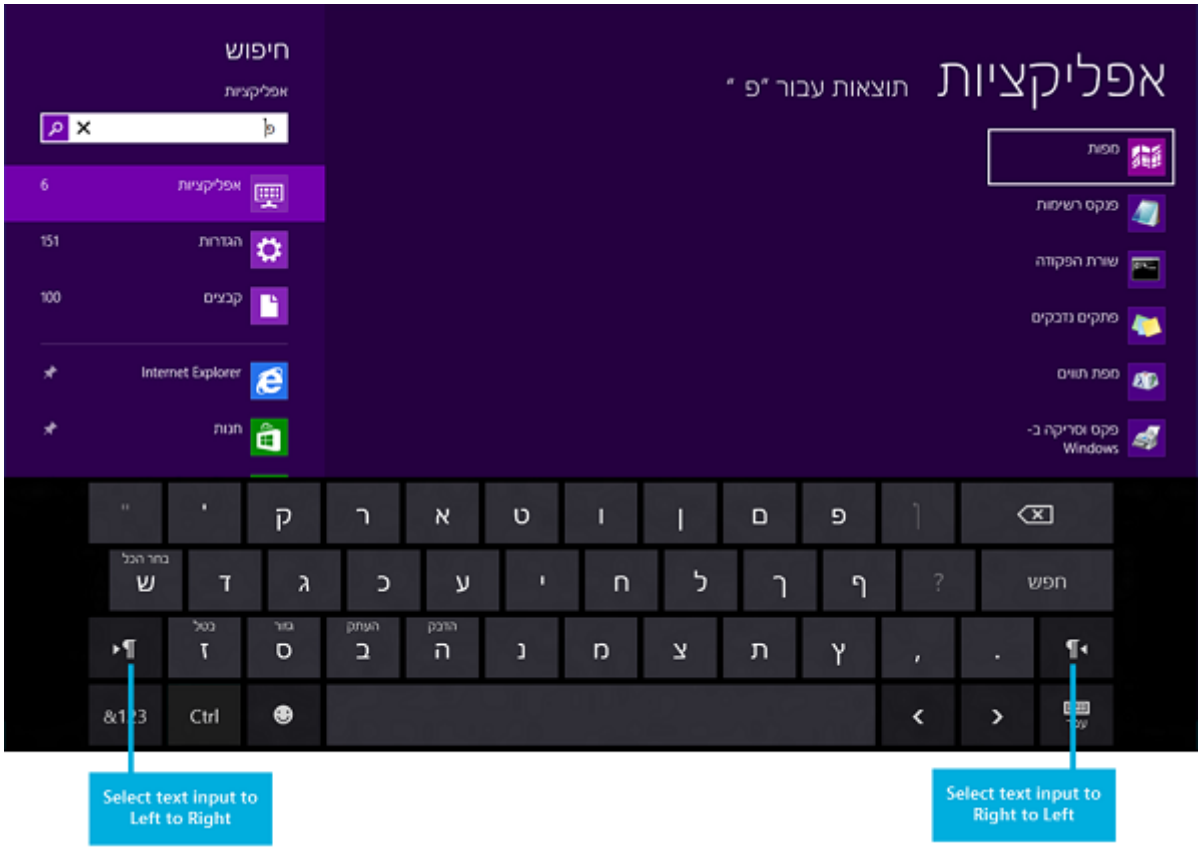






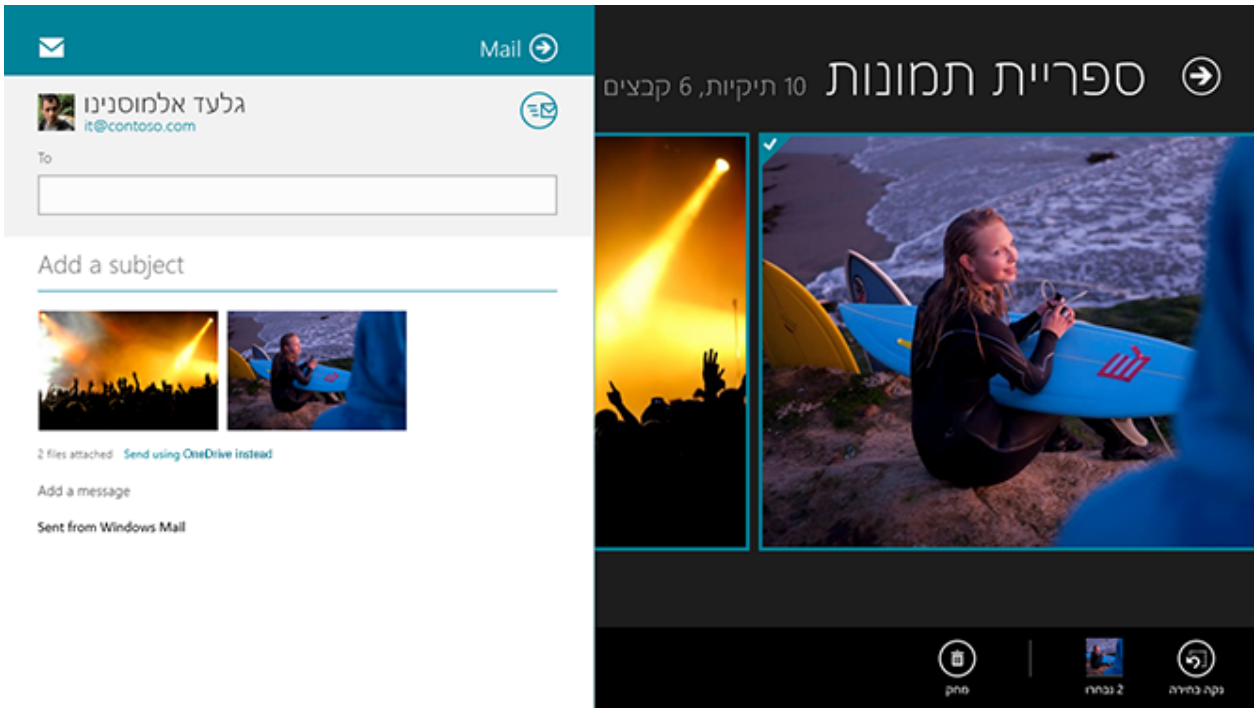
Text input in any direction

Windows offers an on-screen touch keyboard that is clean and clutter-free. For BiDi languages, there is a text direction control key so that the text input direction can be switched as needed.



Use any app in any language

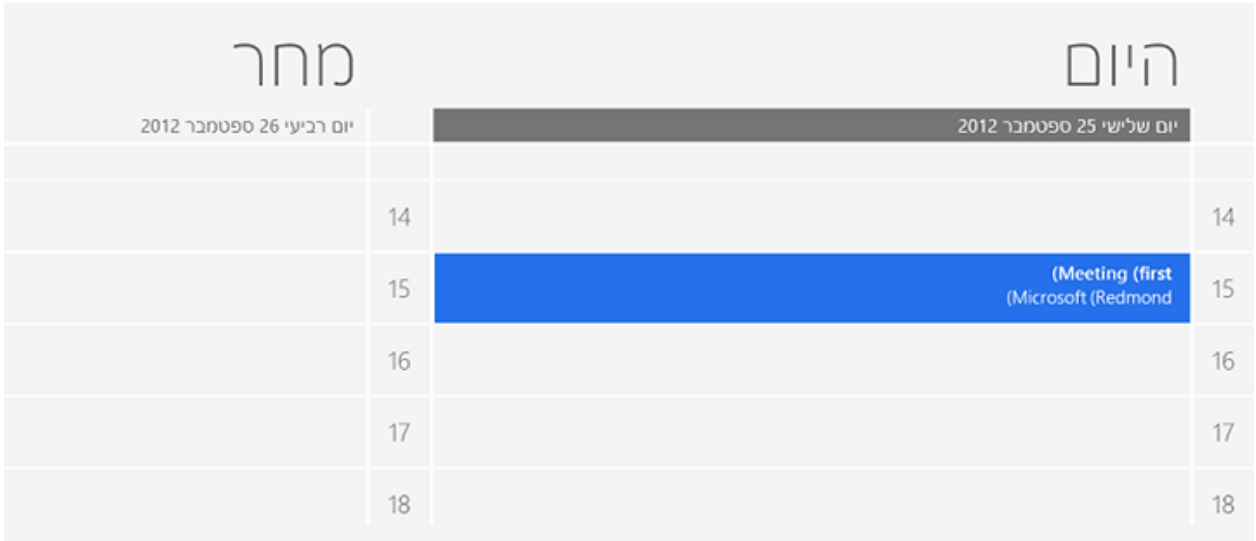
Install and use your favorite apps in any language. The apps appear and function as they would on non-BiDi versions of Windows. Elements within apps are always placed in a consistent and predictable position.



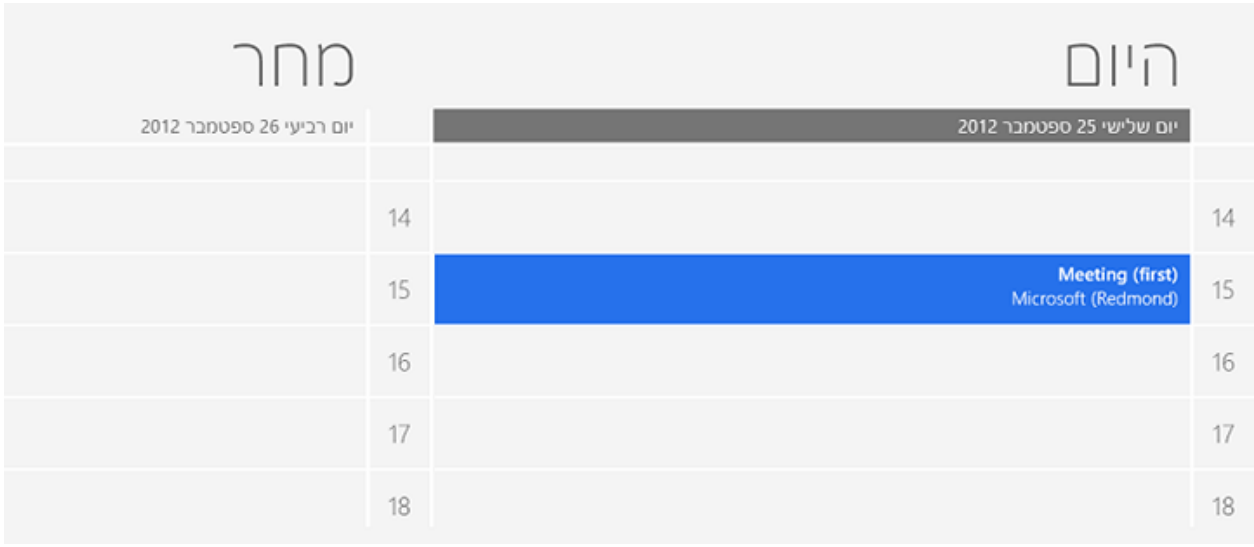
Display parentheses correctly

With the introduction of the BiDi Parenthesis Algorithm (BPA), paired parentheses always display properly regardless of language or text alignment properties.

Incorrect parentheses



Correct parentheses



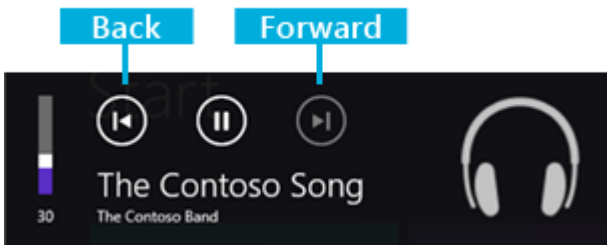
Typography

Windows uses the Segoe UI font for all BiDi languages. This font is shaped and scaled for the Windows UI.

Case study #1: A BiDi music app

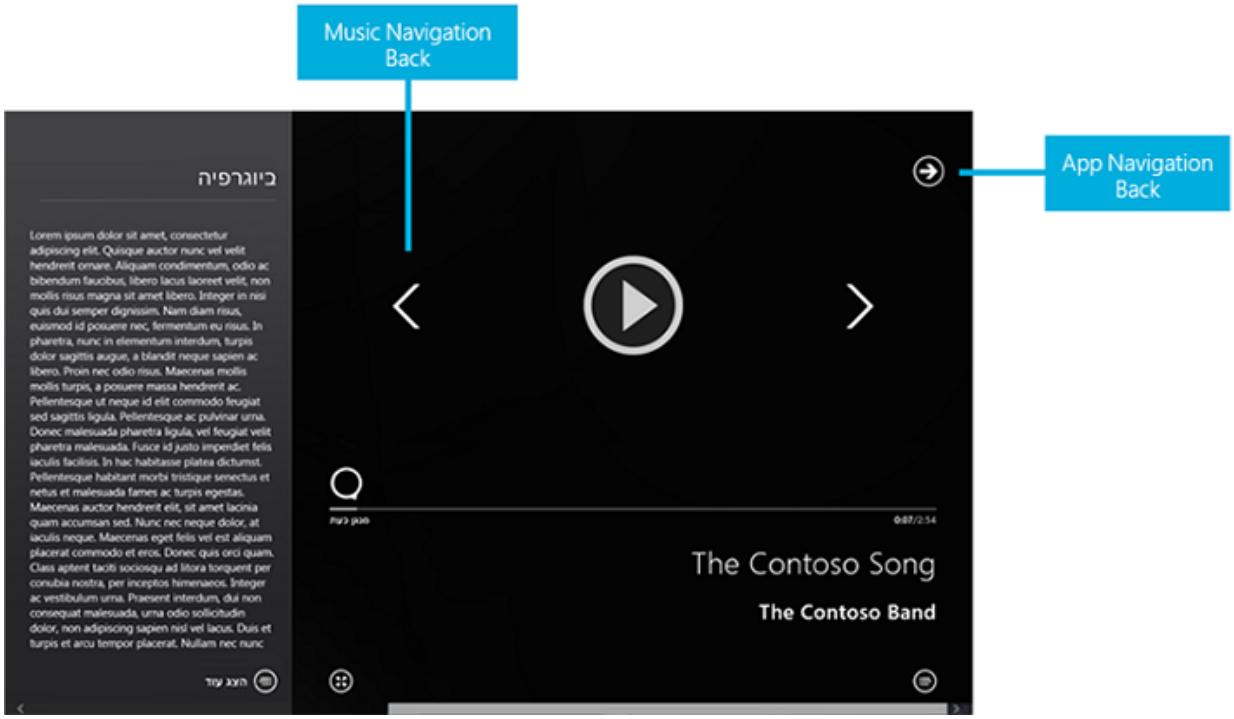
Overview

Multimedia apps make for a very interesting design challenge, because media controls are generally expected to have a left-to-right layout similar to that of non-BiDi languages.

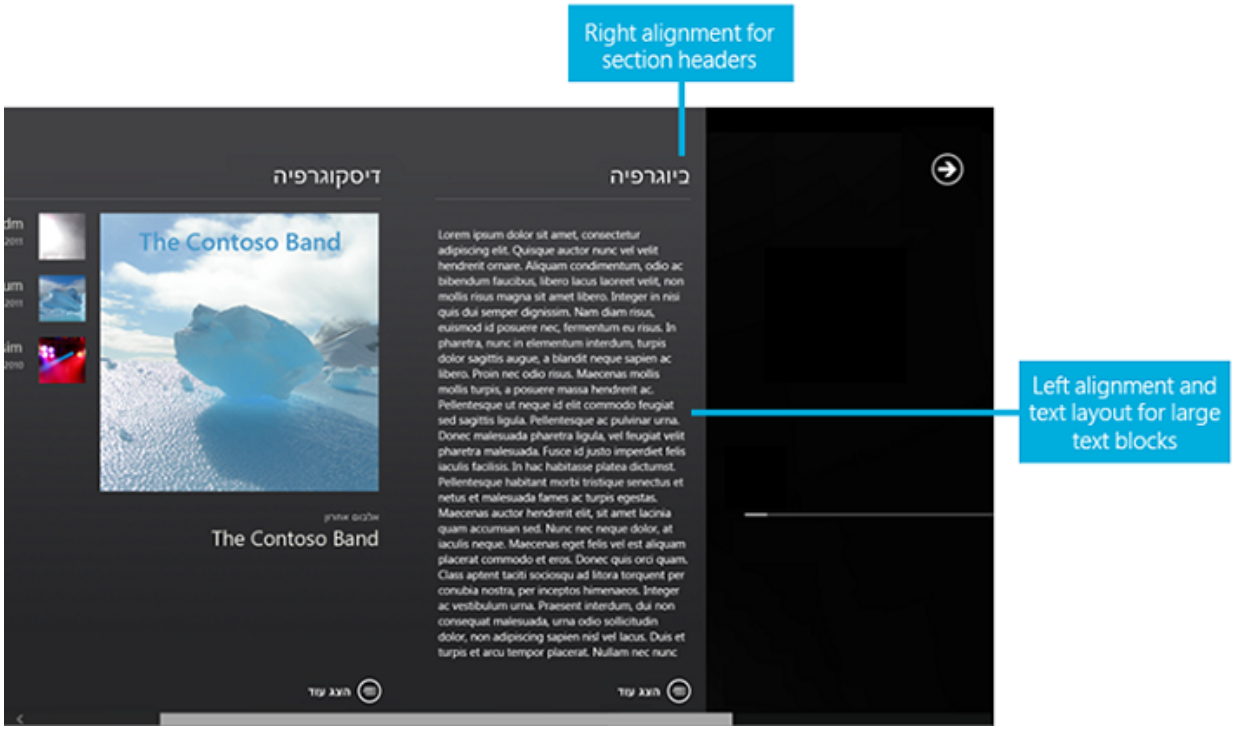


Establishing UI directionality

Retaining the right-to-left UI flow is important for consistent design for BiDi markets. Adding elements that have left-to-right flow within this context is difficult, because some navigational elements such as the back button may contradict the directional orientation of the back button in the audio controls.



This music app retains a right-to-left-oriented grid. This gives the app a very natural feel for users who already navigate in this direction across the Windows UI. The flow is retained by ensuring that the main elements are not just ordered from right to left, but also aligned properly in the section headers to help maintain the UI flow.



Text handling

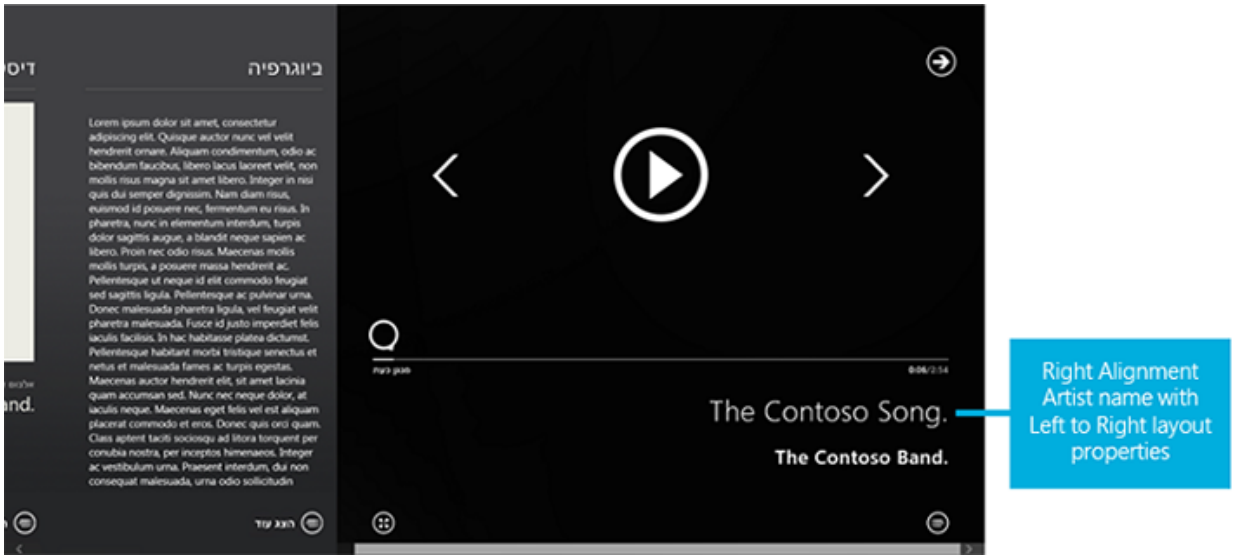
The artist bio in the screenshot above is left-aligned, while other artist-related text pieces such as album and track names preserve right alignment. The bio field is a fairly large text element, which reads poorly when aligned to the right simply because it's hard to track between the lines while reading a wider text block. In general, any text element with more than two or three lines containing five or more words should be considered for similar alignment exceptions, where the text block alignment is opposite to that of the overall app directional layout.

Manipulating the alignment across the app can look simple, but it often exposes some of the boundaries and limitations of the rendering engines in terms of neutral character placement across BiDi strings. For example, the following string can display differently based on alignment.

	English String (LTR)	**Hebrew String (RTL)**
Left-alignment	Hello, World!	!בוקר טוב
Right-alignment	!Hello, World	!בוקר טוב

To ensure that artist information is properly displayed across the music app, the development team separated text layout properties from alignment. In other words, the artist info might be displayed as right-aligned in many of the cases, but the string layout adjustment is set based on customized background processing. The background

processing determines the best directional layout setting based on the content of the string.



For example, without custom string layout processing, the artist name "The Contoso Band." would appear as ".The Contoso Band".

Specialized string direction preprocessing

When the app contacts the server for media metadata, it preprocesses each string prior to displaying it to the user. During this preprocessing, the app also does a transformation to make the text direction consistent. To do this, it checks whether there are neutral characters on the ends of the string. Also, if the text direction of the string is opposite to the app direction set by the Windows language settings, then it appends (and sometimes prepends) Unicode direction markers. The transformation function looks like this.

C#

```
string NormalizeTextDirection(string data)
{
    if (data.Length > 0) {
        var lastCharacterDirection =
DetectCharacterDirection(data[data.Length - 1]);

        // If the last character has strong directionality (direction is not
null), then the text direction for the string is already consistent.
        if (!lastCharacterDirection) {
            // If the last character has no directionality (neutral
character, direction is null), then we may need to add a direction marker to
            // ensure that the last character doesn't inherit directionality
from the outside context.
            var appTextDirection = GetAppTextDirection(); // checks the
<html> element's "dir" attribute.
            var dataTextDirection = DetectStringDirection(data); // Run
```

```
through the string until a non-neutral character is encountered,
                                                                    // which
determines the text direction.

            if (appTextDirection != dataTextDirection) {
                // Add a direction marker only if the data text runs
opposite to the directionality of the app as a whole,
                // which would cause the neutral characters at the ends to
flip.
                var directionMarkerCharacter =
                    dataTextDirection == TextDirections.RightToLeft ?
                        UnicodeDirectionMarkers.RightToLeftDirectionMarker :
// "\u200F"
                        UnicodeDirectionMarkers.LeftToRightDirectionMarker;
// "\u200E"

                data += directionMarkerCharacter;

                // Prepend the direction marker if the data text begins with
a neutral character.
                var firstCharacterDirection =
DetectCharacterDirection(data[0]);
                if (!firstCharacterDirection) {
                    data = directionMarkerCharacter + data;
                }
            }
        }
    }

    return data;
}
```

The added Unicode characters are zero-width, so they don't impact the spacing of the strings. This code carries a potential performance penalty, since detecting the direction of a string requires running through the string until a non-neutral character is encountered. Each character that's checked for neutrality is first compared against several Unicode ranges as well, so it isn't a trivial check.

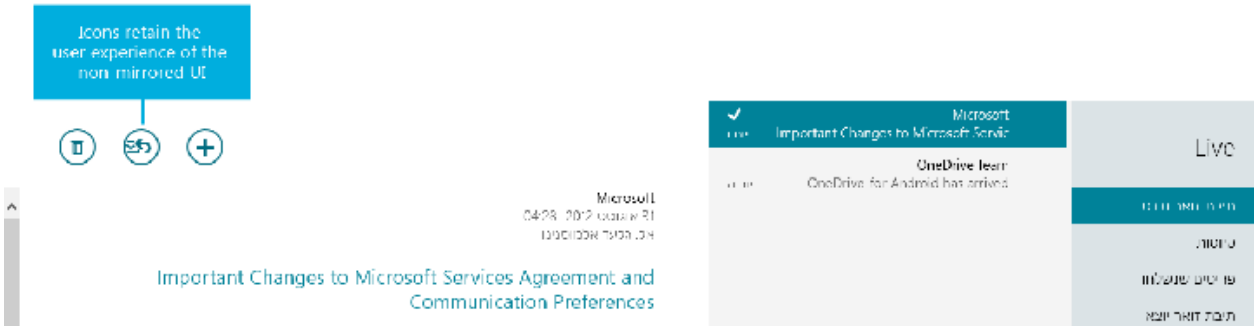
Case study #2: A BiDi mail app

Overview

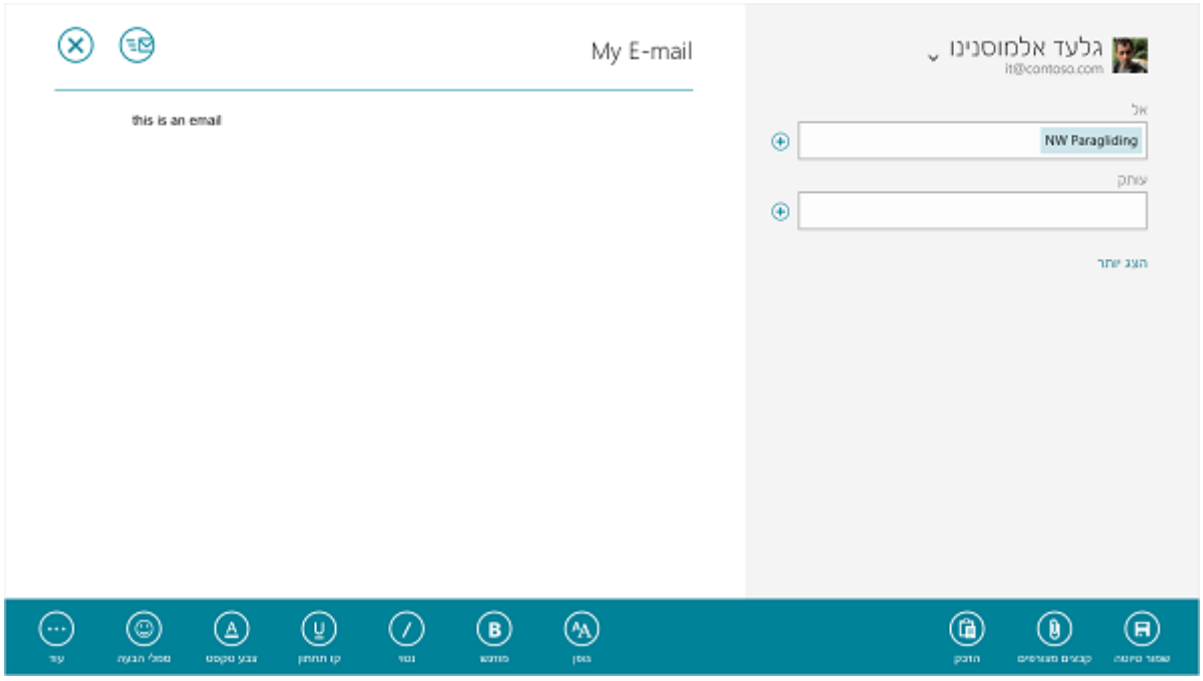
In terms of UI layout requirements, a mail client is fairly simple to design. The Mail app in Windows is mirrored by default. From a text-handling perspective the mail app is required to have more robust text display and composition capabilities in order to accommodate mixed text scenarios.

Establishing UI directionality

The UI layout for the Mail app is mirrored. The three panes have been reoriented so that the folder pane is positioned on the right edge of the screen, followed by the mail item list pane to the left, and then the email composition pane.



Additional items have been reoriented to match the overall UI flow, and touch optimization. These include the app bar and the compose, reply, and delete icons.



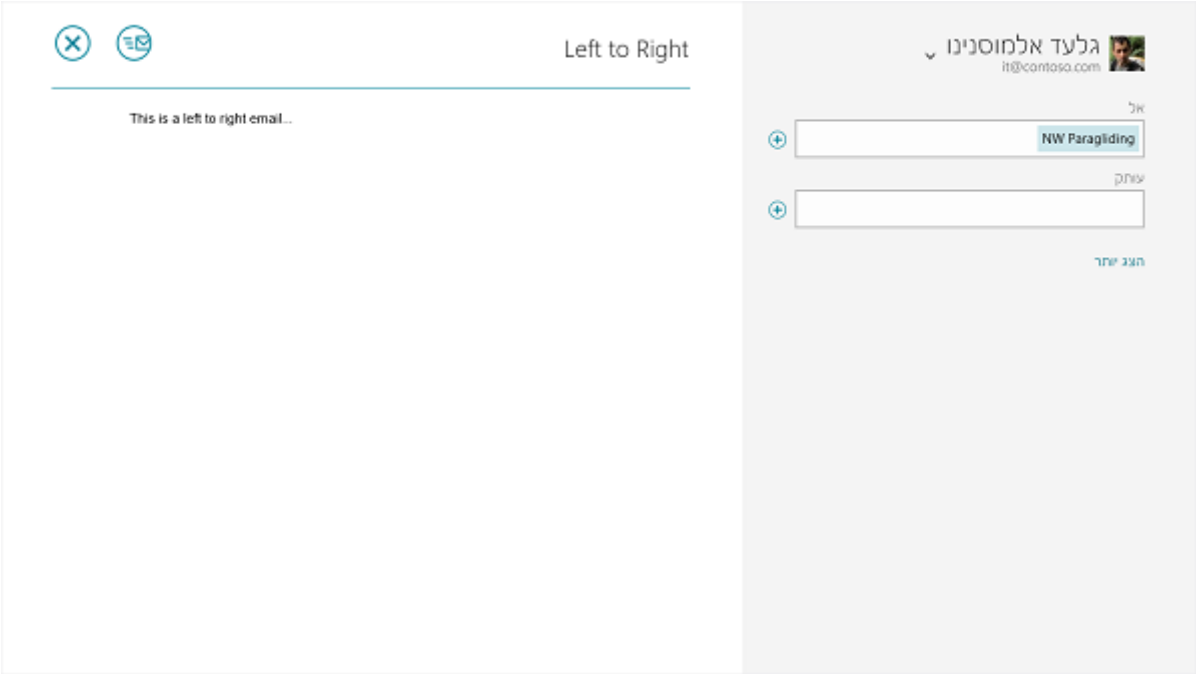
Text Handling

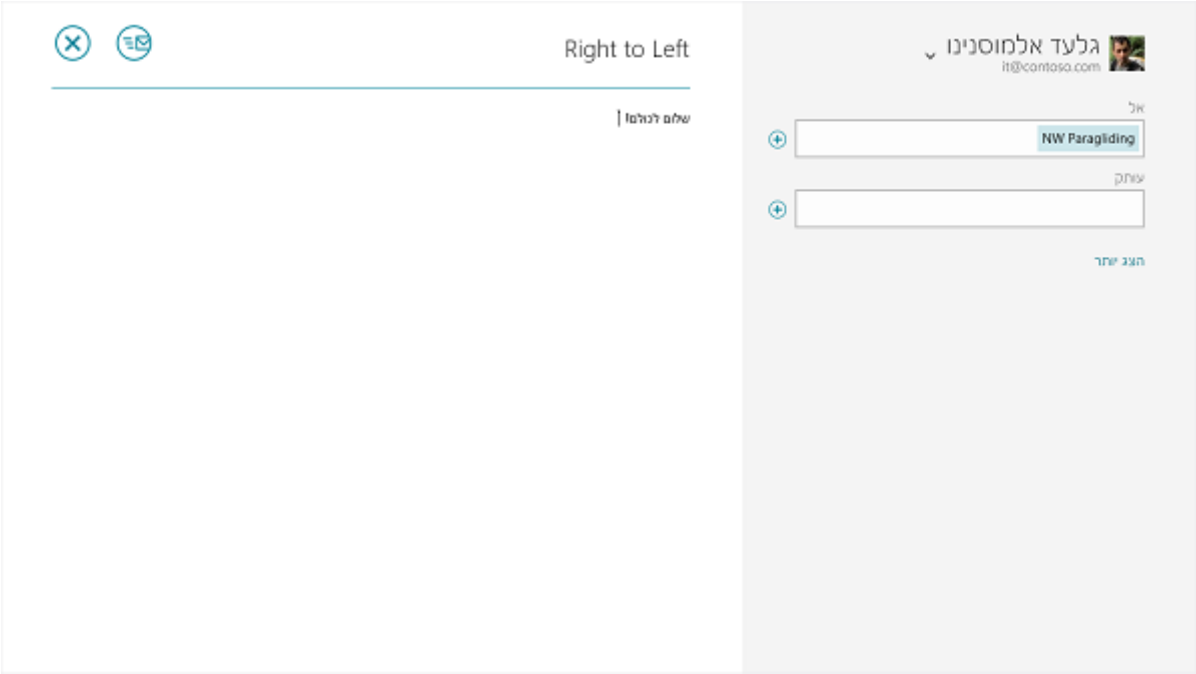
UI

Text alignment across the UI is usually right-aligned. This includes the folder pane and items pane. The item pane is limited to two lines of text (Address, and Title). This is important for retaining the right-to-left alignment, without introducing a block of text that would be difficult to read when the content direction is opposite to the UI direction flow.

Text Editing

Text editing requires the ability to compose in both right-to-left and left-to-right form. In addition, the composition layout must be preserved by using a format—such as rich text—that has the ability to save directional information.





Use UTF-8 code pages in Windows apps

Article • 08/20/2024

Use UTF-8 character encoding for optimal compatibility between web apps and other *nix-based platforms (Unix, Linux, and variants), minimize localization bugs, and reduce testing overhead.

UTF-8 is the universal code page for internationalization and is able to encode the entire Unicode character set. It is used pervasively on the web, and is the default for *nix-based platforms.

Set a process code page to UTF-8

As of Windows Version 1903 (May 2019 Update), you can use the ActiveCodePage property in the appxmanifest for packaged apps, or the fusion manifest for unpackaged apps, to force a process to use UTF-8 as the process code page.

> **Note**
>
> GDI doesn't currently support setting the ActiveCodePage property per process. Instead, GDI defaults to the active system codepage. To configure your app to render UTF-8 text via GDI, go to Windows **Settings** > **Time & language** > **Language & region** > **Administrative language settings** > **Change system locale**, and check **Beta: Use Unicode UTF-8 for worldwide language support**. Then reboot the PC for the change to take effect.

You can declare the ActiveCodePage property, and target/run on earlier Windows builds, but you must handle legacy code page detection and conversion as usual. With a minimum target version of Windows Version 1903, the process code page will always be UTF-8, so legacy code page detection and conversion can be avoided.

> **Note**
>
> In UTF-8, an encoded character is represented by a sequence of 1 to 4 bytes. (See definition D92 in Chapter 3 of The Unicode Standard for the formal specification.)

Examples

Appx manifest for a packaged app:

XAML

```
<?xml version="1.0" encoding="utf-8"?>
<Package
xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"
         ...

xmlns:uap7="http://schemas.microsoft.com/appx/manifest/uap/windows10/7"

xmlns:uap8="http://schemas.microsoft.com/appx/manifest/uap/windows10/8"
         ...
         IgnorableNamespaces="... uap7 uap8 ...">

  <Applications>
    <Application ...>
      <uap7:Properties>
        <uap8:ActiveCodePage>UTF-8</uap8:ActiveCodePage>
      </uap7:Properties>
    </Application>
  </Applications>
</Package>
```

Fusion manifest for an unpackaged Win32 app:

XAML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
  <assemblyIdentity type="win32" name="..." version="6.0.0.0"/>
  <application>
    <windowsSettings>
      <activeCodePage
xmlns="http://schemas.microsoft.com/SMI/2019/WindowsSettings">UTF-
8</activeCodePage>
    </windowsSettings>
  </application>
</assembly>
```

> **Note**
>
> Add a manifest to an existing executable from the command line with `mt.exe -manifest <MANIFEST> -outputresource:<EXE>;#1`.

-A vs. -W APIs

Win32 APIs often support both -A and -W variants.

-A variants recognize the ANSI code page configured on the system and support `char*`, while -W variants operate in UTF-16 and support `WCHAR`.

Until recently, Windows has emphasized "Unicode" -W variants over -A APIs. However, recent releases have used the ANSI code page and -A APIs as a means to introduce UTF-8 support to apps. If the ANSI code page is configured for UTF-8, then -A APIs typically operate in UTF-8. This model has the benefit of supporting existing code built with -A APIs without any code changes.

Code page conversion

Because Windows operates natively in UTF-16 (`WCHAR`), you might need to convert UTF-8 data to UTF-16 (or vice versa) to interoperate with Windows APIs.

MultiByteToWideChar and WideCharToMultiByte let you convert between UTF-8 and UTF-16 (`WCHAR`) (and other code pages). This is particularly useful when a legacy Win32 API might only understand `WCHAR`. These functions allow you to convert UTF-8 input to `WCHAR` to pass into a -W API and then convert any results back if necessary.

Use `dwFlags` of either `0` or `MB_ERR_INVALID_CHARS` when using these functions with `CodePage` set to `CP_UTF8` (otherwise an `ERROR_INVALID_FLAGS` occurs).

> **Note**
>
> `CP_ACP` equates to `CP_UTF8` only if running on Windows Version 1903 (May 2019 Update) or above and the ActiveCodePage property described above is set to UTF-8. Otherwise, it honors the legacy system code page. We recommend using `CP_UTF8` explicitly.

Related topics

- Code pages
- Code Page Identifiers

Prepare your application for the Japanese era change

Article • 05/13/2022

> **Note**
>
> On April 1, 2019, the new era name was announced: Reiwa (令和). On April 25, Microsoft released packages for different Windows operating systems containing the updated registry key with the new era name. Update your device and check your registry to see if it has the new key, and then test your application. Check **this support article** to make sure your operating system should have received the updated registry key.

The Japanese calendar is divided into eras, and for most of the modern age of computing, we've been in the Heisei era; however, on May 1, 2019, a new era will begin. Because this is the first time in decades for an era to change, software that supports the Japanese calendar will need to be tested to ensure it will function properly when the new era begins.

In the following sections, you will learn what you can do to prepare and test your application for the upcoming new era.

> **Note**
>
> We recommend that you use a test machine for this, because the changes you make will impact the behavior of the entire machine.

Add a registry key for the new era

> **Note**
>
> The following instructions are meant for devices that have not yet been updated with the new registry key. First check if your device contains the new registry key, and if not, test using the following instructions.

It is important to test for compatibility problems before the era has changed, and you can do so now using the new era name. To do this, add a registry key for the new era using **Registry Editor**:

1. Navigate to **Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Nls\Calendars\Japanese\Eras**.
2. Select **Edit** > **New** > **String Value**, and give it the name **2019 05 01**.
3. Right-click the key and select **Modify**.
4. In the **Value data** field, enter **令和_令_Reiwa_R** (you can copy and paste from here to make it easier).

See Era Handling for the Japanese Calendar to read more about the format for these registry keys.

On April 1, 2019, the new era name was announced. On April 25, an update with a new registry key for supported Windows versions containing the name was released, allowing you to validate that your application handles it properly. This update is being propagated to supported earlier releases of Windows 10, as well as Windows 8 and Windows 7.

You can delete your placeholder registry key once you're finished testing your application. This will ensure it doesn't interfere with the new registry key that will be added when Windows is updated.

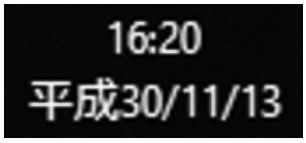
Change your device's calendar format

Once you've added the registry key for the new era, you need to configure your device to use the Japanese calendar. You don't need to have a Japanese-language device to do this. For thorough testing, you may want to install the Japanese language pack as well, but it isn't required for basic testing.

To configure your device to use the Japanese calendar:

1. Open **intl.cpl** (search for it from the Windows search bar).
2. From the **Format** dropdown, select **Japanese (Japan)**.
3. Select **Additional settings**.
4. Select the **Date** tab.
5. From the **Calendar type** dropdown, select **和暦** (*wareki*, the Japanese calendar). It should be the second option.
6. Click **OK**.
7. Click **OK** in the **Region** window.

Your device should now be configured to use the Japanese calendar, and it will reflect whichever eras are in the registry. Below is an example of what you might see now in the lower-right corner of the screen:



Adjust your device's clock

On Windows 10

1. Right-click the date and time area in the lower-right corner of the screen.
2. Select **Adjust date/time**.
3. In the Settings app, under **Change date and time**, select **Change**.
4. Change the date to on or after May 1, 2019.

On Windows 11

1. Right-click the date and time area in the lower-right corner of the screen.
2. Select **Adjust date/time**.

3. In the Settings app, under **Set the date and time manually**, select **Change**.
4. Change the date to on or after May 1, 2019.

> ⓘ **Note**
>
> You may not be able to change the date and time, when "Set time automatically" is "on".

> ⓘ **Note**
>
> You may not be able to change the date based on organization settings; if this is the case, talk to your admin. Alternatively, you can edit your placeholder registry key to have a date that has already passed.

Test your application

Now, test out how your application handles the new era. Check places where the date is displayed, such as timestamps and date pickers. Make sure that the era is correct before May 1, 2019 (Heisei, 平成) and after (Reiwa, 令和).

Gannen (元年)

The format for the Japanese calendar is generally **<Era name> <Year of era>**. For example, the year 2018 is **Heisei 30** (平成30年). However, the first year of an era is special; instead of being **<Era name> 1**, it is **<Era name> 元年** (*gannen*). So, the first year of the Heisei era would be 平成元年 (*Heisei gannen*). Make sure that your application properly handles the first year of the new era, and correctly outputs 令和元年.

Related APIs

There are several WinRT, .NET, and Win32 APIs that will be updated to handle the era change, so if you use them, you shouldn't have to worry too much. However, even if you do rely entirely on these APIs, it's still a good idea to test your application and make sure you get the desired behavior, especially if you are doing anything special with them like parsing.

You can follow along with updates to the OS and SDKs at Updates for May 2019 Japan Era Change.

The following APIs will be impacted:

WinRT

- Windows.Globalization Namespace
  - Calendar Class
    - AddDays Method
    - AddEras Method
    - AddHours Method

    - AddMinutes Method
    - AddMonths Method
    - AddNanoseconds Method
    - AddPeriods Method
    - AddSeconds Method
    - AddWeeks Method
    - AddYears Method
    - Era Property
    - EraAsString Method
    - FirstYearInThisEra Property
    - LastEra Property
    - LastYearInThisEra Property
    - NumberOfYearsInThisEra Property
- Windows.Globalization.DateTimeFormatting Namespace
  - DateTimeFormatter Class
    - Format Method

.NET

- System Namespace
  - DateTime Struct
  - DateTimeOffset Struct
- System.Globalization Namespace
  - Calendar Class
  - DateTimeFormatInfo Class
  - JapaneseCalendar Class
  - JapaneseLunisolarCalendar Class

Win32

- datetimeapi.h header
  - GetDateFormatA function
  - GetDateFormatEx function
  - GetDateFormatW function
- winnls.h header
  - EnumDateFormatsA function
  - EnumDateFormatsExA function
  - EnumDateFormatsExEx function
  - EnumDateFormatsExW function
  - EnumDateFormatsW function
  - GetCalendarInfoA function
  - GetCalendarInfoEx function
  - GetCalendarInfoW function

See also

- Era Handling for the Japanese Calendar
- The Japanese Calendar's Y2K Moment
- Using the Registry to Test the New Japanese Era on Windows
- Gannen vs Ichinen
- Updates for May 2019 Japan Era Change
- Handling a new era in the Japanese calendar in .NET

Announcements

Article • 04/16/2024

Learn about important Multilingual App Toolkit (MAT) news and updates here.

Multilingual App Toolkit 4.1

The Multilingual App Toolkit 4.1 update released.

MAT 4.1 audience

This extension is provided free to any Visual Studio developers interested in localizing their supported project types (such as UWP, WPF, VSIX, .NET for iOS/Mac/Android, and others).

What's new in MAT 4.1

- Visual Studio 2022 support.

MAT 4.1 Features

- **IDE and build integration**: End-to-end resource file localization. Translated resources are synchronized during builds.
- **Machine Translation**: Use Microsoft Translator and Language portal for translation.
- **Import/Export**: Recycle existing translations by importing .xlf files, or export machine translations for human review.

For more details, see Use the Multilingual App Toolkit.

Download MAT 4.1

Download the Multilingual App Toolkit 4.1 (VS 2022+) from the Visual Studio Marketplace.

Multilingual App Toolkit 4.1 Editor

Multilingual App Toolkit 4.1 Editor updates released.

MAT 4.1 Editor audience

The Multilingual App Toolkit Editor is a dedicated localization editor for human translators that, through its support for XLIFF, is not limited to any product or business category.

Although this editor is a part of the Multilingual App Toolkit and can be incorporated into workflows using the MAT Visual Studio extension, the use of Visual Studio is not a requirement. For example, this editor can be leveraged by Microsoft Dynamics 365 Translation Service (DTS) users to revise the output of machine translations produced by DTS.

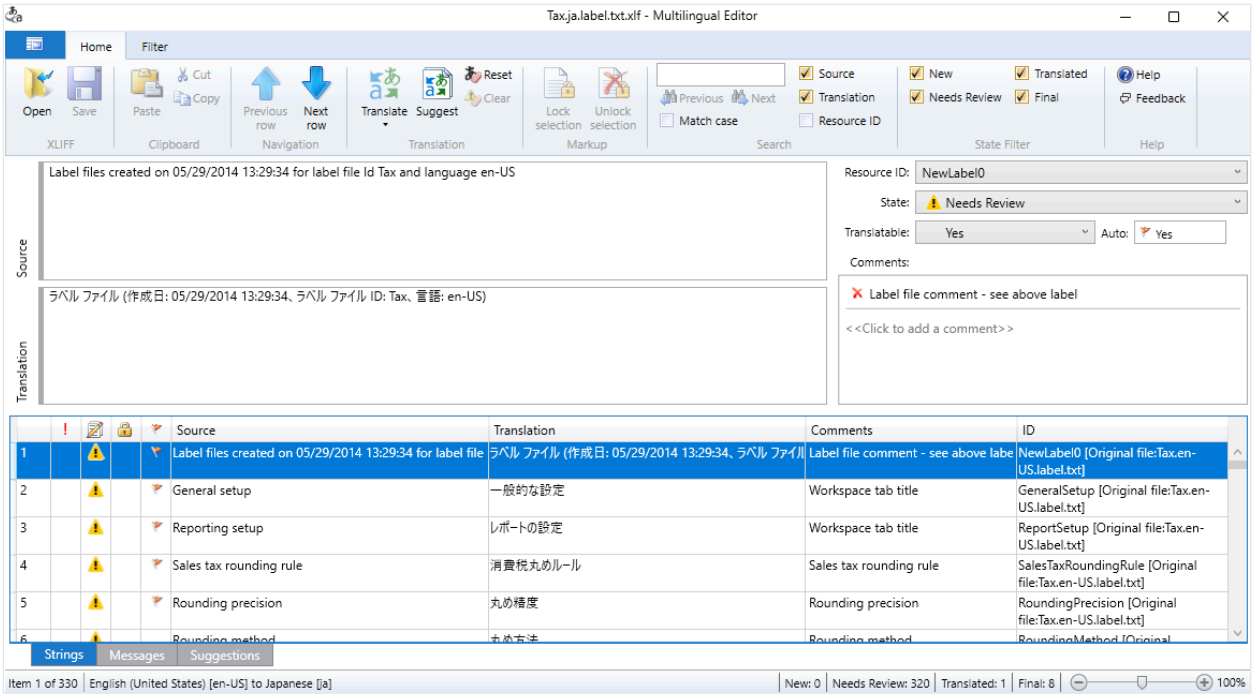
What's new in the MAT 4.1 Editor

- Allow installation without Visual Studio as a pre-requisite.
- Allow users to configure MS Translator provider from the UI.
- Re-enable Language Portal as a translation provider.
- Redesign of telemetry collection to satisfy current compliance requirements.

For more details, see Multilingual App Toolkit Editor.

Download MAT 4.1 Editor

See Multilingual App Toolkit 4.1 Editor for language downloads.



See also

See the (Multilingual-App-Toolkit) GitHub repo for a variety of samples and to provide feedback, ask questions, and report issues or bugs.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Use the Multilingual App Toolkit

Article • 11/14/2022

The Multilingual App Toolkit (MAT) integrates with Microsoft Visual Studio to provide Windows apps with translation support, translation file management, and editor tools. Here are some of the value propositions of the toolkit.

- Helps you manage resource changes and translation status during development.
- Provides a UI for choosing languages based on configured translation providers.
- Supports the localization industry-standard XLIFF file format.
- Provides a pseudo-language engine to help identify translation issues during development.
- Connects with the Microsoft Language Portal to easily access translated strings and terminology.
- Connects with the Microsoft Translator for quick translation suggestions.

How to use the toolkit

Step 1. Design your app for globalization and localization

Before you can use the MAT effectively, your app needs to be localizable. Specifically, your project should contain one or more Resources Files (.resw) containing your app's strings in the default language. For details, see Localize strings in your UI and app package manifest. Once you've done that, the toolkit makes adding additional languages quick and easy.

For the value proposition of globalization and localization—as well as definitions of the terms **globalization**, **localizability**, and **localization**—see Globalization and localization.

Also see Guidelines for globalization and Make your app localizable.

Step 2. Download and install the Multilingual App Toolkit

There are two parts to the Multilingual App Toolkit (MAT), each with its own installer.

- Multilingual App Toolkit Extension for Visual Studio 2017-2019. This contains the MAT extension for Visual Studio 2019, in the form of a .vsix installer.
- Multilingual App Toolkit 4.1 Extension for Visual Studio 2022 and later. This contains the MAT 4.1 extension for Visual Studio 2022, in the form of a .vsix installer.

- Multilingual app toolkit Editor. This contains the MAT standalone Multilingual Editor tool, in the form of an .msi installer. It also includes the MAT extension for Visual Studio 2015 and for Visual Studio 2013.

If you use Visual Studio 2017 or later, then download and run both installers, one after the other. If you use Visual Studio 2015 or Visual Studio 2013, then download and run the .msi installer.

Step 3. Enable the Multilingual App Toolkit for your project

The MAT must be enabled for your project before you can begin to localize the app. Here's how to enable the toolkit.

- Open the project solution in Visual Studio.
- Select the desired project in Solution Explorer.
- On the **Tools** menu, select **Multilingual App Toolkit** > **Enable selection**.

In the Output window (showing output from Multilingual App Toolkit), watch for the message `Project '<project-name>' was enabled. The project's source culture is '<language-tag>' <language-name>`. If this message appears, then the MAT is ready to use.

Step 4. Add languages to your project

Follow these steps to add languages to your project.

1. Right-click the project node in Solution Explorer.
2. Click **Multilingual App Toolkit** > **Add translation languages...**.
3. In the Translation Languages dialog, select the language(s) you want to support, and click OK.

The toolkit does these things in response.

- For each language you added, a new folder is created named for the BCP-47 language tag of the language. Inside that folder, new Resources File(s) (.resw) are created to match the one(s) that contain the default language strings.
- If this is the first time you've added a language, a new folder named `MultilingualResources` is added to the project. Inside that folder, an .xlf file is added for each language. The .xlf files contain a translation unit for each string in each Resources File (.resw) in your project.
- The Output window confirms the addition of the language(s) that you added.

Whenever you add/remove a default language Resources File (.resw), or you add/remove a string inside a default language Resources File (.resw), rebuild the project to re-synchronize the .xlf files. This ensures that the .xlf files contain the union of the strings in the default language.

Installed Translation Providers—such as the Microsoft Language Portal and Microsoft Translator—can be used to translate your app's resources. When a provider supports a specific language, the provider's icon is displayed next to the language name in the Translation Languages dialog.

In the Translation Languages dialog, any existing .xlf-based languages that are discovered by the toolkit have their selection box pre-checked to indicate that the language is already included the project.

Once a language is added to the project, it cannot be removed by un-checking the box in the Translation Languages dialog. To remove a language, right-click on the language-specific .xlf file and select **Delete**. If you confirm, then this will also delete the corresponding Resources File (.resw).

Step 5. Test your app using pseudo language

Pseudo language is an artificial modification of the software product intended to simulate real language localization, but remaining readable to native speakers. Pseudo translation replaces characters and expands the resource string length to detect potential localizability issues or bugs early in the project cycle and before localization starts in earnest.

Follow these steps to pseudo-localize and test your project.

1. Use the Translation Languages dialog to add Pseudo Language (Pseudo) [qps-ploc] to your project.
2. Right-click the `<project-name>.qps-ploc.xlf` file in Solution Explorer and click **Multilingual App Toolkit** > **Generate machine translations**.
3. In **Settings** > **Time & Language** > **Region & language** > **Languages**, click **Add a language**.
4. In the search box, type `qps-ploc`.
5. Click `English (qps-ploc)` to add it.
6. From the language list, select `English (qps-ploc)` and click **Set as default**.
7. Test your pseudo-localized app. For example, look for UI layout issues where not all of a string is displayed (the string is truncated), or strings that are not translated (but instead hard-coded).

In addition to character replacement and expansion, the pseudo engine provides a unique tracking identifier for each resource. This tracker is prepended to the start of every string and enclosed within brackets `[xxxxx]`. You can use these trackers during visual UI inspection testing. They can help track down specific resources in the product, especially if multiple resources have similar or duplicate text.

In this "Hello, World!" text example, the pseudo translation expands to take about 30 percent more screen space, and then applies the resource tracker.

`"Hello World" -> "Ĥèĺļõ Ŵòŗłđ" -> "[!!_Ĥèĺļõ Ŵòŗłđ_!!]" -> "[hJ8s1][!!_Ĥèĺļõ Ŵòŗłđ_!!]"`

Step 6. Translate your app into selected languages

The Multilingual App Toolkit is integrated into the build process. During a build, updated strings are automatically added to each language .xlf file. After you've tested your app by using Pseudo language, there are three options to translate your app into other languages for release.

Option 1. Translate the strings yourself

You can use the Multilingual Editor to translate strings individually. As already mentioned, this is included in The .msi installer.

- Right-click the .xlf file that you want to translate.
- Click **Open With...** and select Multilingual Editor. You can optionally click **Set as Default**.
- For each string, **Source** shows the original string in the default language. In **Translation**, type the string translated into the appropriate language for the .xlf file that you're editing.
- When you're done, save and close the file.

Rebuild your project to cause the translated strings to be copied into the Resources File (.resw) that corresponds to the .xlf file you were just editing.

You can also launch the Multilingual Editor like this. Go to Start, show all apps, open the Multilingual App Toolkit folder, and click Multilingual Editor to launch it.

Option 2. Send the .xlf files to a third party for translation

To outsource the translation and editing work to localizers, select the desired .xlf files in Solution Explorer, right-click them, and click **Multilingual App Toolkit** > **Export**

translations....

Select **Output: Mail recipient** in the Export string resources dialog, and click OK, and your files will be zipped and attached to a new email. Select **Output: File folder location**, browser for a folder and click OK, optionally choose for the files to be zipped, click OK again, and your files will be (zipped and) saved at the location you chose, inside a new folder named for your project.

After your localizers complete the translation work and send you the translated .xlf files, you can import them into your project. Select the desired .xlf files in Solution Explorer, right-click them, and click **Multilingual App Toolkit** > **Import/recycle translations...**. Click **Add**, navigate to the .xlf or .zip files, and click **Import**.

Note The import process performs basic validation before importing. This ensures that the target culture information in the files being imported matches that in the existing .xlf files.

Rebuild your project to cause the translated strings to be copied into the Resources File(s) (.resw) that corresponds to the .xlf file(s) you just imported.

These third party providers offer localization services, and may be able to assist you.

- Elanex
- Keywords Studios
- Lionbridge
- Moravia
- SDL
- Welocalize

> **Note**
>
> The list above is provided for informational purposes only and is not an endorsement. Microsoft does not make any representation or warranty regarding these vendors or their services, and under no circumstances will Microsoft have any liability for your use of such vendors or services. Any questions, complaints, or claims regarding such vendors or their services must be directed to the appropriate vendor.

Option 3. Use the integrated translation services

Translation services are integrated into the Visual Studio IDE as well as into the Multilingual Editor. This provides easy access to translation services while developing

your product as well as localizing your resources. For this service, you'll need an Azure account subscription, as described in Microsoft Translator Moves to the Azure portal.

To access the translation services inside Visual Studio, select and right-click one or more .xlf files in Solution Explorer and click **Generate machine translations**.

The Multilingual Editor provides the same translation support, as well as adding interactive translation suggestions, which enable you to select the translation that best fits your resource strings. After the translation suggestion is provided, you can fine-tune the string for your translation style.

Two providers are shipped with the Multilingual App Toolkit.

- The Microsoft Language Portal provider enables translation-recycling and terminology-matching support based on translations of the user interface text for Microsoft products and services.
- The Microsoft Translator provider enables on-demand machine translation services.

You and your translator(s) can manage the status of translations in the Multilingual Editor to review uncertain translations later. You can set the status of each string in the **Properties** tab. Status values are: **New**, **Needs review**, **Translated**, **Final**, and **Signed off**. The indicator at the left of the row shows the status. When all rows show green in the Multilingual Editor, then your translation work is done.

Rebuild your project to cause the translated strings to be copied into the Resources File(s) (.resw) that corresponds to the .xlf file(s) you just edited.

Step 7. Upload your app to the Microsoft Store

Before you start the Microsoft Store Certification process, you must exclude the `<project-name>.qps-ploc.xlf` file from your project. Pseudo language is used to detect potential localizability issues or bugs, but it is not a valid Microsoft Store language. If it is not removed then your app will fail during the Microsoft Store Certification process.

Related topics

- Localize strings in your UI and app package manifest
- Globalization and localization
- Guidelines for globalization
- Make your app localizable
- BCP-47 language tag

Telemetry

The Multilingual App Toolkit Visual Studio (MAT VS) extension collects anonymous usage data and sends it to Microsoft servers to help improve our products and services. Telemetry is off by default and can be enabled or disabled at any time. For more details, see Multilingual App Toolkit FAQ & troubleshooting.

Downloads

- Multilingual App Toolkit Visual Studio 2017-2019
- Multilingual App Toolkit Visual Studio 2022+
- Multilingual app toolkit Editor

Translation services

- Microsoft Language Portal
- Microsoft Translator

Multilingual App Toolkit Editor

Article • 11/14/2022

Use the standalone **Multilingual App Toolkit Editor** to help you localize your apps with localization file management, translation support, and editing tools.

Each of the following downloads contain an .msi installer for the Multilingual App Toolkit Editor (also known as the Multilingual Editor). The installer is available in 11 languages, please select the link for your preferred language.

- To start the installation immediately, click **Run**.
- To save the download to your computer for installation at a later time, click **Save**.

English

Japanese

Chinese (Simplified)

Korean

Chinese (Traditional)

Portuguese (Brazil)

French

Russian

German

Spanish

Italian

> **Important**
>
> If using Visual Studio, you should also download and install **Multilingual App Toolkit Visual Studio 2017-2019** or **Multilingual App Toolkit Visual Studio 2022+**.

Overview

The Multilingual Editor helps create simpler translation workflows by focusing on the following areas:

- Use to easily edit translated strings.
- Get translations and suggestions quickly via the integrated Microsoft Language Portal and the Microsoft Translator services (requires an active Internet connection).
- Narrow down on the desired translation units by filtering on text and states.
- You can also quickly edit data stored in XLIFF files by adjusting pseudo and actual translations.

Installation requirements

> **ⓘ Important**
>
> Ensure you have the latest service pack and critical updates for your installed versions of Windows and Visual Studio.

- **Supported operating systems**: Windows 10 or later (x86 and x64)
- **Disk space requirements**: 6 MB (x86 and x64)

Additional info

- You must have an active Internet connection to use the Microsoft Language Portal and Microsoft Translator services.
- Microsoft Translator requires a subscription key for the Microsoft Translator Text API.

Multilingual App Toolkit FAQ & troubleshooting

FAQ

This topic provides answers to frequently-asked questions and issues related to the Multilingual App Toolkit (MAT).

Also see Use the Multilingual App Toolkit.

Note The toolkit supports both .resw (XAML) and .resjson (JavaScript) files. But in this topic we'll refer only to .resw files. A .resw file is known as a Resources File. It contains strings, either in the default language or translated into another language. The folder that contains a .resw file is typically named for the value of a language tag.

Do I need .resw files in multiple languages?

No. One of the key benefits of the Toolkit is that .resw files in multiple languages are not required. The Toolkit manages and synchronizes your app's resources by using .xlf files. This removes the challenges associated with keeping content synchronized across multiple .resw files.

Projects that contain matching .resw and .xlf files cause the translations from the .xlf file to be ignored. When this happens, a warning is displayed during the build to let you know that the .xlf translations are not included in the final app. A .resw file and .xlf file match when they have a target language with the same language code. An example of a matching pair would be `Strings\de-DE\Resources.resw` and an `<project-name>.de-DE.xlf` file (containing `target-language="de-DE"`).

Can I have .resw files in multiple languages?

You can, but we don't recommend it. If you want to include .resw files in multiple languages in your project and use the Toolkit, make sure you don't have matching .resw and .xlf files.

I don't see an option in the Tools menu to enable the Multilingual App Toolkit

Try these steps.

- Make sure you select the project node, not the solution node, before opening the **Tools** menu.
- Confirm that the toolkit extension is installed by using the Visual Studio Extension Manager.
- Confirm that your project is a UWP project.

When I build my project, I don't see a message saying that a Multilingual App Toolkit build has started

Confirm that you've enabled the MAT for your project. On the **Tools** menu, select **Multilingual App Toolkit** > **Enable selection**. If your project was enabled with a previous version, then disable and re-enable the MAT by using the **Tools** menu. This updates the project to work with the new version of the Toolkit.

Ensure that the "Build Task for all Visual Studio editions" component was installed. This build component is installed with the extension, but it can be manually deselected during the installation. This component is required to update the .xlf files and add the translation into the PRI file. When this component is installed and working correctly, you will see these build messages.

dosbatch

```
1> Multilingual App Toolkit build started.
1> Multilingual App Toolkit build completed successfully.
```

The toolkit is reporting that it didn't locate any XLIFF language files during the build

dosbatch

```
No XLIFF language files were found. The app will not contain any localized
resources.
```

This message is displayed when the toolkit doesn't find any files in the project with an extension of .xlf. The toolkit generates and keeps these files in the `MultilingualResources` folder by default. They can be moved; but it's best to leave them in that folder because that allows the Multilingual Editor to locate the related metadata files.

My .xlf file is not included in the list of files processed by the toolkit during build

If you manually exclude an .xlf file from the project, and then re-include it, the file type element may not be set correctly. In Visual Studio, select the file and check the Properties window. The file's Build Action should be set to XliffResource and Copy to Output Directory should be set to Do not copy. This is how the reference should look in your project file.

XML

```
<XliffResource Include="MultilingualResources\<project-name>.fr-FR.xlf" />
```

I've added .xlf-based languages. Where are my strings?

Your default language Resources File (.resw) is your canonical "schema" of the strings that your app uses. Translated Resources Files can contain all, or a subset of, these strings.

When you build your project, your Resources Files and your .xlf files are synchronized.

- The .xlf files are updated to reflect any added or removed string, or added or removed Resources Files.
- The Resources Files are updated to reflected any translated strings in the .xlf files.

This is explained in detail in Use the Multilingual App Toolkit.

When I build my project, the .xlf files remain empty

Before you can use the MAT effectively, your app needs to be localizable. This is explained in detail in Use the Multilingual App Toolkit.

What is Microsoft Translator?

Microsoft Translator is a cloud-based service that provides machine-based translation. Machine translation is ideal for gaining access to translation when human translation is not reasonable to obtain. You can learn more at Microsoft Translator.

The Toolkit uses the Microsoft Translator service to provide translation suggestions to you. You can see which languages are supported by Microsoft Translator when the Microsoft Translator icon is present in the Translation Languages dialog.

You can quickly translate your app using Microsoft Translator from within the Multilingual Editor by selecting a string and clicking **Translate**.

What is pseudo language, and what are pseudo resource trackers?

Pseudo language is an artificial modification of the software product intended to simulate real language localization. You can find more details about pseudo language and pseudo resource trackers in Use the Multilingual App Toolkit.

How do I set my language preference to pseudo language, so that I can test my pseudo-loc'd strings?

This is explained in Use the Multilingual App Toolkit.

What kind of localizability issues can I find using pseudo language?

This is explained in Use the Multilingual App Toolkit.

I'm not seeing any translations when I launch my app, or my app is only partially translated

Open the .xlf file in the Multilingual Editor to see whether the translations are present. When strings in the default language .resw file are changed explicitly, any corresponding translations are removed from .xlf files. This is to ensure that a translation matches its source string. Translate the string(s) in the .xlf file(s), and rebuild to update the non-default language .resw file(s).

If your strings are translated in the .xlf file(s), but they're not appearing in your app, then rebuild your project to update the non-default language .resw file(s). Visual Studio optimizes the Build command to build only files that have changed since the last Build.

Check your language preference order. Ensure that the language you want to test is listed at the top of your language preference list in **Settings**.

The toolkit is reporting error 0x80004004 in the build output

```dosbatch
Merge of Loc PRI file failed calling makepri.exe: "0x80004004"
```

This message can be displayed when the region format conflicts with the toolkit build operation. The workaround is to change your language in **Settings** to en-US while building.

The toolkit is reporting error 0x80004005 in the build output

```dosbatch
Merge of Loc PRI file failed calling makepri.exe: "0x80004005"
```

This message can be displayed when the .xlf file contains an unsupported target language. For example, "zh-cht" is incorrect (change it to "zh-hant"), and "zh-chs" is

incorrect (change it to "zh-hans").

Is there a way to find out more information about the errors I'm seeing?

Yes, you can turn on verbose logging in Visual Studio. Click **Tools** > **Options** > **Projects and Solutions** > **Build and Run**. Change **MSBuild project build output verbosity** from Minimal to Normal or higher.

Running MSBuild from the command line can also produce additional messages.

```dosbatch
msbuild /t:rebuild <project-name>
```

Import translation failed

The import process performs basic validation before importing. This ensures that the target culture information in the files being imported matches that in the existing .xlf files. Open the .xlf files in the Multilingual Editor and make sure that the culture information matches.

What if my translator doesn't have Windows 10, Windows 11 and/or Visual Studio and/or the Multilingual App Toolkit installed?

When you select **Output: Mail recipient** in the Export string resources dialog, the email includes a link to download and install the Multilingual App Toolkit (MAT). Your translator can still install the MAT standalone Multilingual Editor tool even without Windows 10, Windows 11 nor Visual Studio.

For more details, see Use the Multilingual App Toolkit.

What happened to the `MarkupRules.xml` and `ResourcesLocks.xml` files?

The Multilingual App Toolkit doesn't use proprietary resource locking files. Instead, the XLIFF 1.2 tag `<mrk>` is added directly to the .xlf file to identify strings that are not modified during Machine Translation. This enables the XLIFF file to be self-contained, and allows for per-file based resource locking.

These extra support files are no longer needed and you can safely delete them if you have them.

What happened to the .tpx file?

The .tpx file provided an easy way to include the `MarkupRules.xml` and `ResourcesLocks.xml` files when the .xlf file was sent out for translation. This functionality is no longer required.

If you have translations in a .tpx file that you need to retrieve, simply rename the .tpx file extension to .zip. This will allow you to open and extract the contents with File Explorer or any .zip compatible tool.

I think I've done everything right, but it still isn't working

Try these steps.

1. Add translations using one of the methods already described.
2. Dump the .pri file (see MakePri.exe command-line options) to see whether your translations are in the .pri file. Translations will appear with language code and translated value like this.

```xml
<Candidate qualifiers="Language-QPS-PLOC" type="String">
    <Value>[!!_Ŝéãřćĥ_!!]</Value>
</Candidate>
```

3. Build from a Command Prompt; the resulting error may have more details than what is reported in the build output.

My app failed certification to the Microsoft Store

Before you start the Microsoft Store Certification process, you must exclude the `<project-name>.qps-ploc.xlf` file from your project. Pseudo language is used to detect potential localizability issues or bugs, but it is not a valid Microsoft Store language. If it is not removed then your app will fail during the Microsoft Store Certification process.

What data are you collecting?

The Multilingual App Toolkit Visual Studio (MAT VS) extension collects anonymous usage data and sends it to Microsoft servers to help improve our products and services. Telemetry is off by default and can be enabled or disabled at any time.

Usage data: This is anonymous information about what features are being used. For example: adding a new language to a MAT-enabled solution.

Configuration data: This is anonymous information regarding the version of MAT VS installed on and the version of Windows it is being run on.

Below is a concrete example of the data collected in the following format, Key / Value / Notes:

- AddLanguage / MachineID: xxxx / This is based on a hash of the network adapter NIC. It is not guaranteed to be unique.
- TimeStamp / 2022/01/07T22:04:45.0000000Z / Time event took place.
- MAT_Version / 4.1.02 / Version of MAT installed.
- Languages / fr,es,de/DE / List of languages added to project.

All data is collected using Application Insights and sent to Azure Log analytics. Data is retained for 90 days then deleted.

Related topics

- Use the Multilingual App Toolkit
- Microsoft Translator
- MakePri.exe command-line options

Guidelines for app help

Article • 06/24/2021

Applications can be complex, and providing effective help for your users can greatly improve their experience. Not all applications need to provide help for their users, and what sort of help should be provided can vary greatly, depending on the application.

If you decide to provide help, follow these guidelines when creating it. Help that isn't helpful can be worse than no help at all.

Intuitive Design

As useful as help content can be, your app cannot rely on it to provide a good experience for the user. If the user is unable to immediately discover and use the critical functions of your app, the user will not use your app. No amount or quality help will change that first impression.

An intuitive and user-friendly design is the first step to writing useful help. Not only does it keep the user engaged for long enough for them to use more advanced features, but it also provides them with knowledge of an app's core functions, which they can build upon as they continue to use the app and learn.

General instructions

A user will not look for help content unless they already have a problem, so help needs to provide a quick and effective answer to that problem. If help is not immediately useful, or if help is too complicated, then users are more likely to ignore it.

All help, no matter what kind, should follow these principles:

- **Easy to understand:** Help that confuses the user is worse than no help at all.
- **Straightforward:** Users looking for help want clear answers presented directly to them.
- **Relevant:** Users do not want to have to search for their specific issue. They want the most relevant help presented straight to them (this is called "Contextual Help"), or they want an easily navigated interface.
- **Direct:** When a user looks for help, they want to see help. If your app includes pages for reporting bugs, giving feedback, viewing term of service, or similar

functions, it is fine if your help links to those pages. But they should be included as an afterthought on the main help page, and not as items of equal or greater importance.

- **Consistent:** No matter the type, help is still a part of your app, and should be treated as any other part of the UI. The same design principles of usability, accessibility, and style which are used throughout the rest of your app should also be present in the help you offer.

Types of help

There are three primary categories of help content, each with varying strengths and suitable for different purposes. Use any combination of them in your app, depending on your needs.

Instructional UI

Normally, users should be able to use all the core functions of your app without instruction. But sometimes, your app will depend on use of a specific gesture, or there may be secondary features of your app which are not immediately obvious. In this case, instructional UI should be used to educate users with instructions on how to perform specific tasks.

See guidelines for instructional UI

In-app help

The standard method of presenting help is to display it within the application at the user's request. There are several ways in which this can be implemented, such as in help pages or informative descriptions. This method is ideal for general-purpose help, that directly answers a user's questions without complexity.

See guidelines for in-app help

External help

For detailed tutorials, advanced functions, or libraries of help topics too large to fit within your application, links to external web pages are ideal. These links should be used sparingly if possible, as they remove the user from the application experience.

See guidelines for external help

Instructional UI guidelines

Article • 06/24/2021

In some circumstances it can be helpful to teach the user about functions in your app that might not be obvious to them, such as specific touch interactions. In these cases, you need to present instructions to the user through the user interface (UI), so that they can use those features they might have missed.

When to use instructional UI

Instructional UI has to be used carefully. When overused, it can be easily ignored or annoy the user, causing it to be ineffective.

Instructional UI should be used to help the user discover important and non-obvious features of your app, such as touch gestures or settings they may be interested in. It can also be used to inform users about new features or changes in your app that they might have otherwise overlooked.

Unless your app is dependent on touch gestures, instructional UI should not be used to teach users the fundamental features of your app.

Principles of writing instructional UI

Good instructional UI is relevant and educational to the user, and enhances the user experience. It should be:

- **Simple:** Users don't want their experience to be interrupted with complicated information
- **Memorable:** Users don't want to see the same instructions every time they attempt a task, so instructions need to be something they'll remember.
- **Immediately relevant:** If the instructional UI doesn't teach a user about something that they immediately want to do, they won't have a reason to pay attention to it.

Avoid overusing instructional UI, and be sure to choose the right topics. Do not teach:

- **Fundamental features:** If a user needs instructions to use your app, consider making the app design more intuitive.
- **Obvious features:** If a user can figure out a feature on their own without instruction, then the instructional UI will just get in the way.
- **Complex features:** Instructional UI needs to be concise, and users interested in complex features are usually willing to seek out instructions and don't need to be

given them.

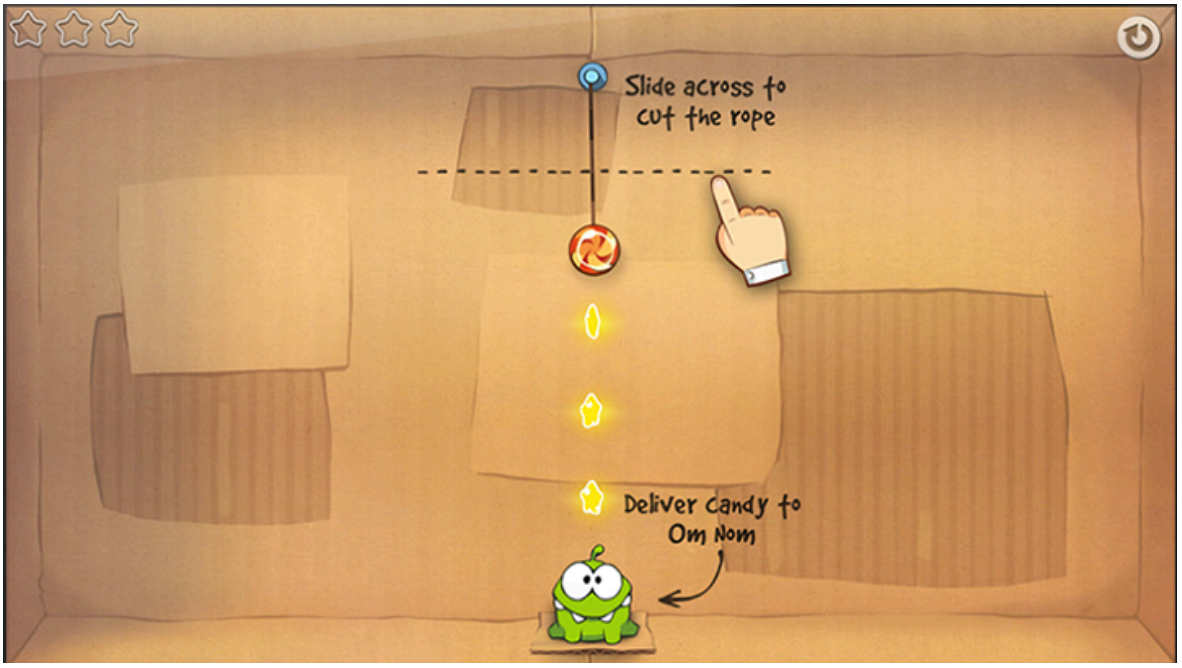
Avoid inconveniencing the user with your instructional UI. Do not:

- **Obscure important information:** Instructional UI should never get in the way of other features of your app.
- **Force users to participate:** Users should be able to ignore instructional UI and still progress through the app.
- **Displaying repeat information:** Don't harass the user with instructional UI, even if they ignore it the first time. Adding a setting to display instructional UI again is a better solution.

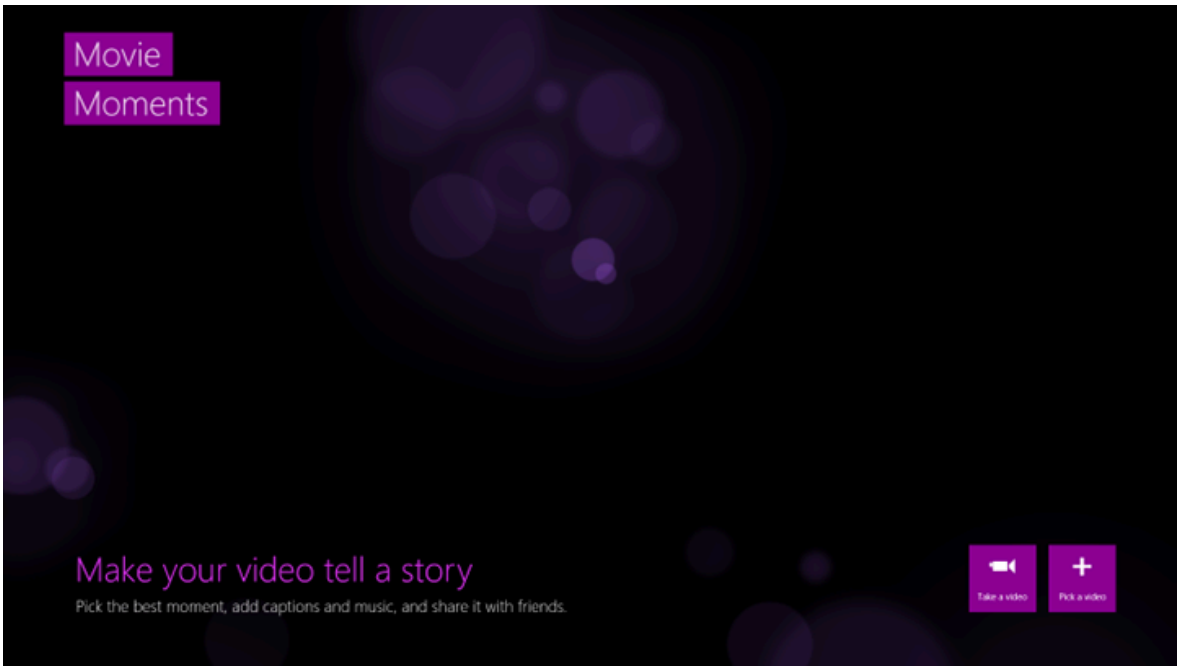
Examples of instructional UI

Here are a few instances in which instructional UI can help your users learn:

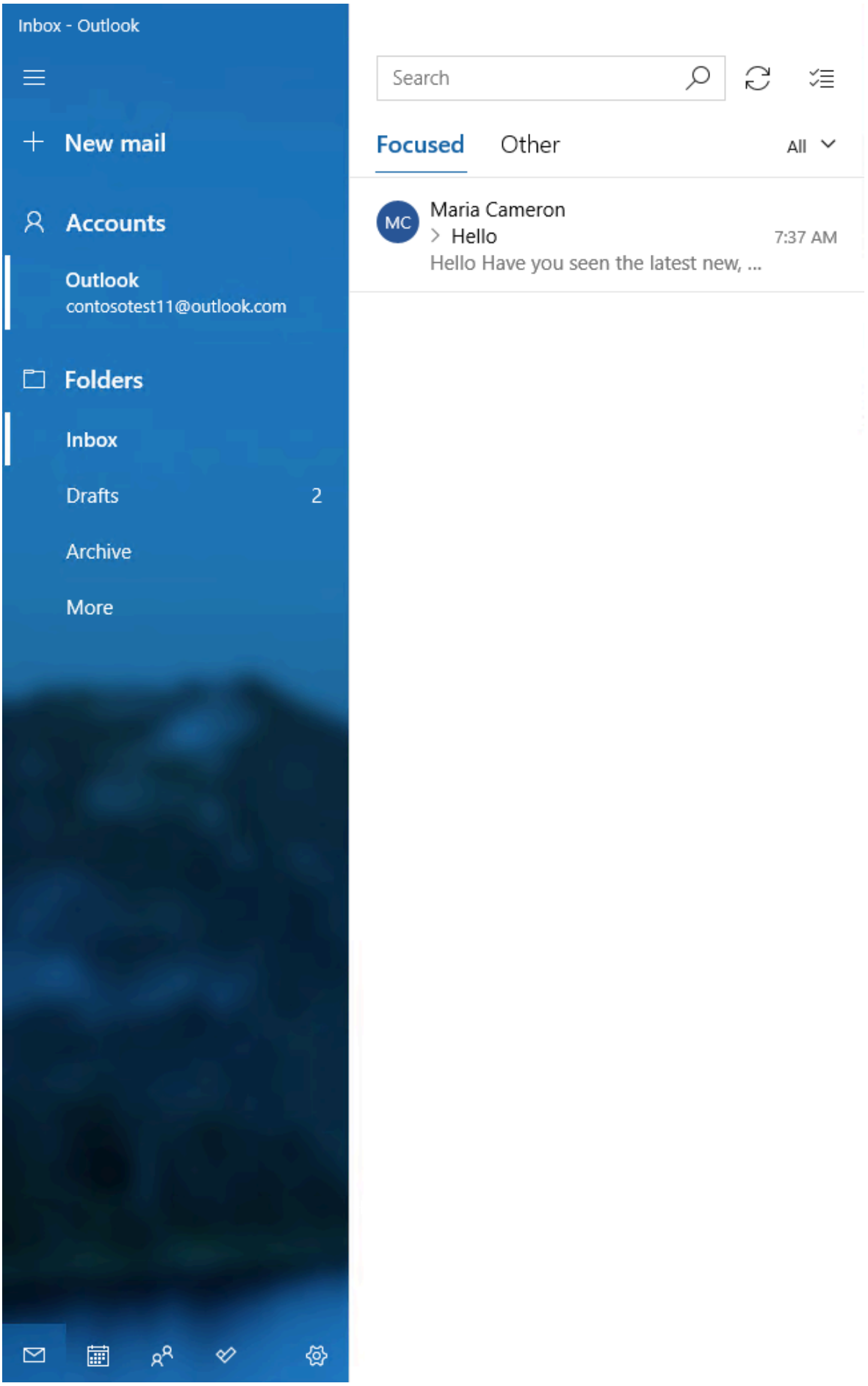
- **Helping users discover touch interactions.** The following screen shot shows instructional UI teaching a player how to use touch gestures in the game, Cut the Rope.



- **Making a great first impression.** When Movie Moments launches for the first time, instructional UI prompts the user to begin creating movies without obstructing their experience.

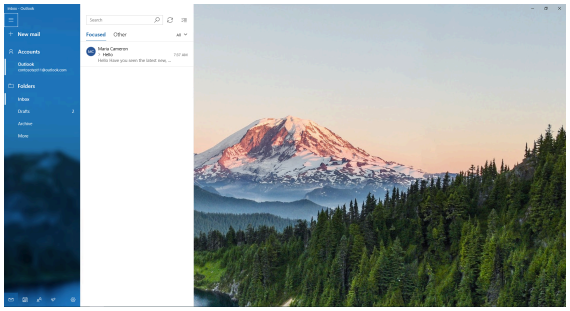
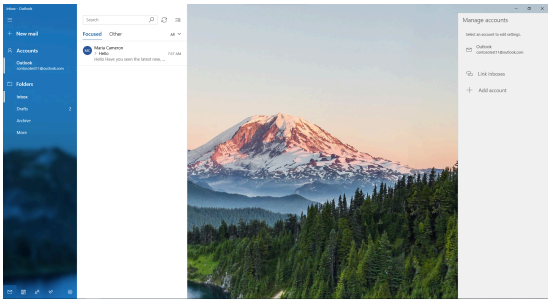


- **Guiding users to take the next step in a complicated task.** In the Windows Mail app, a hint at the bottom of the Inbox directs users to **Settings** to access older messages.



When the user clicks the message, the app's **Settings** flyout appears on the right side of the screen, allowing the user to complete the task. These screen shots show the Mail app before and after a user clicks the instructional UI message.

Expand table

Before	After
	

Related articles

- Guidelines for app help

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

In-app help pages

Article • 06/24/2021

Most of the time, it is best that help be displayed within the application and when the user chooses to view it.

When to use in-app help pages

In-app help should be the default method of displaying help for the user. It should be used for any help which is simple, straightforward, and does not introduce new content to the user. Instructions, advice, and tips & tricks are all suitable for in-app help.

Complex instructions or tutorials are not easy to reference quickly, and they take up large amounts of space. Therefore, they should be hosted externally, and not incorporated into the app itself.

Users should not have to seek out help for basic instructions or to discover new features. If you need to have help that educates users, use instructional UI.

Types of In-app help

In-app help can come in several forms, though they all follow the same general principles of design and usability.

Help Pages

Having a separate page or pages of help within your app is a quick and easy way of displaying useful instructions.

- **Be concise:** A large library of help topics is unwieldy and unsuited for in-app help.
- **Be consistent:** Make sure that users can reach your help pages the same way from any part of your app. They should never have to search for it.
- **Users scan, not read:** Because the help a user is looking for might be on the same page as other help topics, make sure they can easily tell which one they need to focus on.

Popups

Popups allow for highly contextual help, displaying instructions and advice that is relevant to the specific task that the user is attempting.

- **Focus on one issue:** Space is even more restricted in a popup than a help page. Help popups needs to refer specifically a single task to be effective.
- **Visibility is important:** Because help popups can only be viewed from one location, make sure that they're clearly visible to the user without being obstructive. If the user misses it, they might move away from the popup in search of a help page.
- **Don't use too many resources:** Help shouldn't lag or be slow-loading. Using videos or audio files or high resolution images in popups is more likely to frustrate the user than it is to help them.

Descriptions

Sometimes, it can be useful to provide more information about a feature when a user inspects it. Descriptions are similar to instructive UI, but the key difference is that instructional UI attempts to teach and educate the user about features that they don't know about, whereas detailed descriptions enhance a user's understanding of app features that they're already interested in.

- **Don't teach the basics:** Assume that the user already knows the fundamentals of how to use the item being described. Clarifying or offering further information is useful. Telling them what they already know is not.
- **Describe interesting interactions:** One of the best uses for descriptions is to educate the user on how a features that they already know about can interact. This helps users learn more about things they already like to use.
- **Stay out of the way:** Much like instructional UI, descriptions need to avoid interfering with a user's enjoyment of the app.

Related articles

- Guidelines for app help

External help pages

Article • 06/24/2021

If your app requires detailed help for complex content, consider hosting these instructions on a web page.

When to use external help pages

External help pages are less convenient for general use or quick reference. They are suitable for help content that is too extensive to be incorporated into the app itself, as well as for tutorials and instructions for advanced functions of an app that won't be used by its general audience.

If your help content is brief or specific enough to be displayed in-app, you should do so. Do not direct users outside of the app for help unless it is necessary.

Navigating external help pages

When a user is directed to an external help page, follow one of two scenarios:

- They are linked directly to the page that corresponds with their known issue. This is contextual help, and should be used when possible.
- They are linked to a general help page, with a clear display of categories and subcategories to choose from.

Providing users with a way to search your help can be useful, but do not make this search the only way of navigating your help. It can sometimes be difficult for users to describe their problems, which can make searching difficult. Users should be able to quickly find pages relevant to their problems without needing to search.

Tutorials and detailed walkthroughs

External help pages are the ideal place to provide users with tutorials and walkthroughs, whether video or textual.

- Tutorials should focus on more complicated ideas and advanced functions. Users shouldn't need a tutorial to use your app.
- Make sure that these tutorials are displayed differently from standard help instructions. Users who are looking for advanced instructions are more eager to search for them than users who want straightforward solutions to their problems.

- Consider linking to tutorials from both a directory and from individual help pages that correspond to each tutorial.

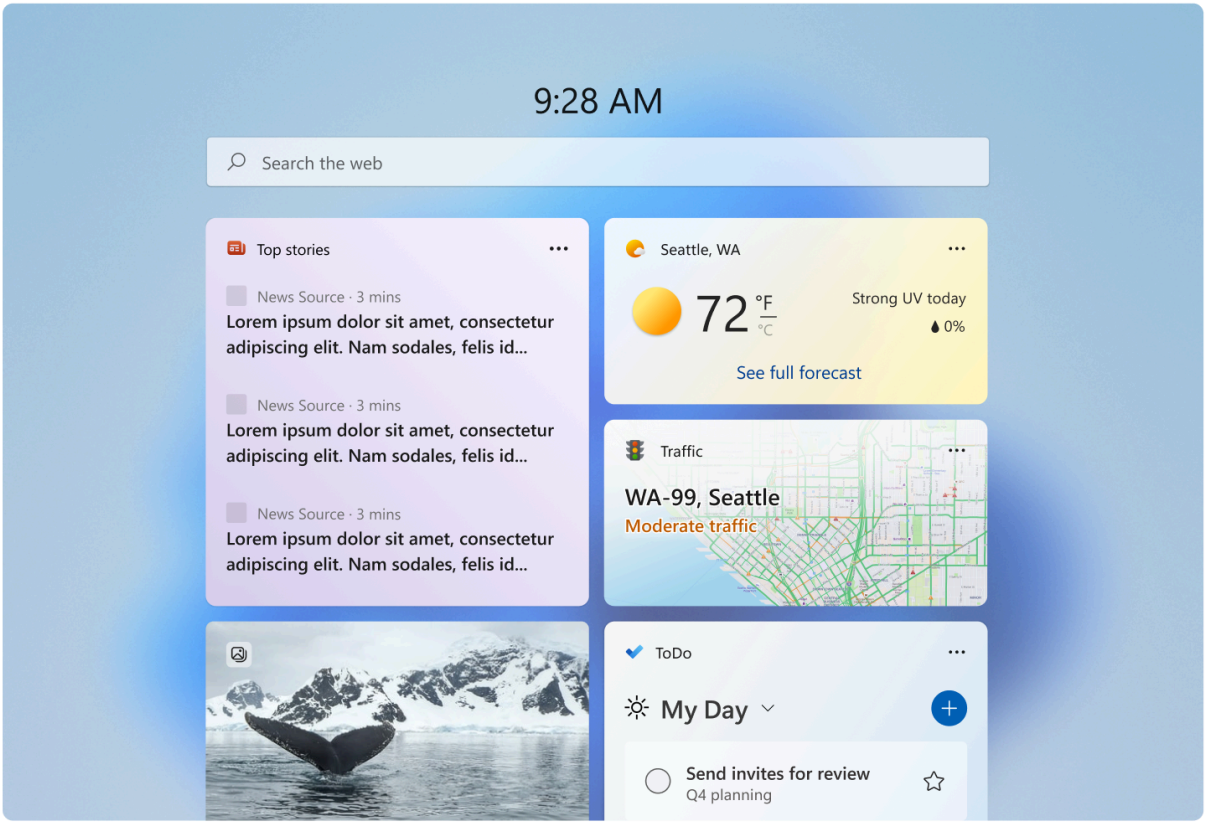
Related articles

- Guidelines for app help

Widgets overview

Article • 03/09/2023

Windows Widgets are small UI containers that display text and graphics, associated with an app installed on the device. Installed widgets are displayed in a grid in the Widgets Board: a flyout plane that overlays the Windows desktop when the user clicks Widgets icon on the taskbar, uses the Windows+W shortcut, or swipes from the left edge of the screen. Widgets help people stay on top of what's important to them by aggregating personalized content and quick actions from the apps they use. They are quickly consumable and actionable. Widgets are not meant to replace apps and websites, but rather provide frictionless access to most-needed information or often-used functionalities that people can read/trigger right away. When designing your widget, consider the kind of value it will bring to your consumers.



Widgets terminology

Expand table

Term	Definition
Widgets host	An application that displays and manages Windows widgets. In the current release, the only Widgets host is the Widgets Board built into Windows 11.

Term	Definition
Widgets Board	The Widgets Board is a Windows 11 system component that is displayed over the desktop when the user clicks the Widgets icon on the taskbar, uses the Windows+W shortcut, or swipes from the left edge of the screen. The Widgets Board displays widgets and manages their layout the on the board.
Widget	A widget is an Adaptive Card that presents important content or actions from an app. It allows users to access desired information instantly without the need to launch the associated app or website. Widget content is refreshed dynamically throughout the day to provide the user with current and interesting content that can be consumed at a glance. Widgets provide basic interactive features that allow the user to launch the associated app for deeper engagement. Widgets are not intended to replace apps and websites.
Widget provider	A widget provider is a Windows app that provides content to be displayed in the widget. The widget provider owns the content, layout, and interactive elements of the widget.

Widget design guidance

The visual experience of a widget includes visual elements and interaction elements that are defined using the Adaptive Cards JSON format. The Adaptive Cards Designer provides a real-time editor for designing adaptive cards as well as templates for the supported widget sizes and themes. It's important that your widget's design adhere the Windows Widget design principles to help ensure that the Widgets Board provides a consistent and familiar experience for all widgets.

For a high-level walkthrough of the visual elements of a widget see Widget states and built-in UI components.

Widget principles

To create great Windows Widgets, consider the following principles as you design and develop your widgets:

Glanceable

Users can take a quick peek to get the most value out of the widget. They only need to click on it if they want richer details or deeper interactions.

Dependable

Surface frequently used information instantly to save users time in repeating those steps. Drive consistent re-engagement to your app.

Useful

Elevate the most useful and relevant information.

Personal

Provide personalized content and build an emotional connection with customers. Widgets should never contain ads. Customers are in control of their widget content and layout.

Focused

Each widget should generally focus on one main task or scenario. Widgets are not intended to replace your apps and websites.

Fresh

Content should dynamically refresh based on available context. It is up to date and provides the right content at the right time.

Planning your app's widget experience

1. Based on your understanding of your customers, identify the most important content or most useful actions that your users would love to have quick access to without opening your app or website. Consider the principles enumerated in the Widget principles section and think about how they can apply to your app.
2. Your app can support multiple individual widgets. Determine the number of separate widgets you want to support so that each widget focuses on a specific purpose.
3. Determine the content you want to include for each widget. A single widget can support three different sizes; small, medium, and large. For each widget, think about what content would bring the most value to users and your business needs. For each size from small to large, the purpose of the widget should remain the same, but the amount of information displayed should expand with larger sizes. We recommend that widget providers implement all widget sizes to give users flexibility when customizing the widget layout.

4. Think about the user interactions your widget will support. Users can click on the widget title or any click targets that you've defined on the widget. These interactions can activate deep-link shortcuts into your app or web site that take users directly to what they're interested in, so that they don't have to navigate from the root of your app. Consider the different navigational models offered.
5. Apps must implement a widget provider that implements the back-end functionality to send your widget's layout and data to the widgets board to be displayed. Currently you can implement a widget provider using a packaged Win32 desktop app or a Progressive Web App (PWA). For more information on creating a Win32 widget provider, see Widget service providers. For information on PWA widget providers, see Build PWA-driven widgets.

In this section

Widget states and UI

Widget design fundamentals

Widget interaction design

Create a widget template with the Adaptive Card Designer

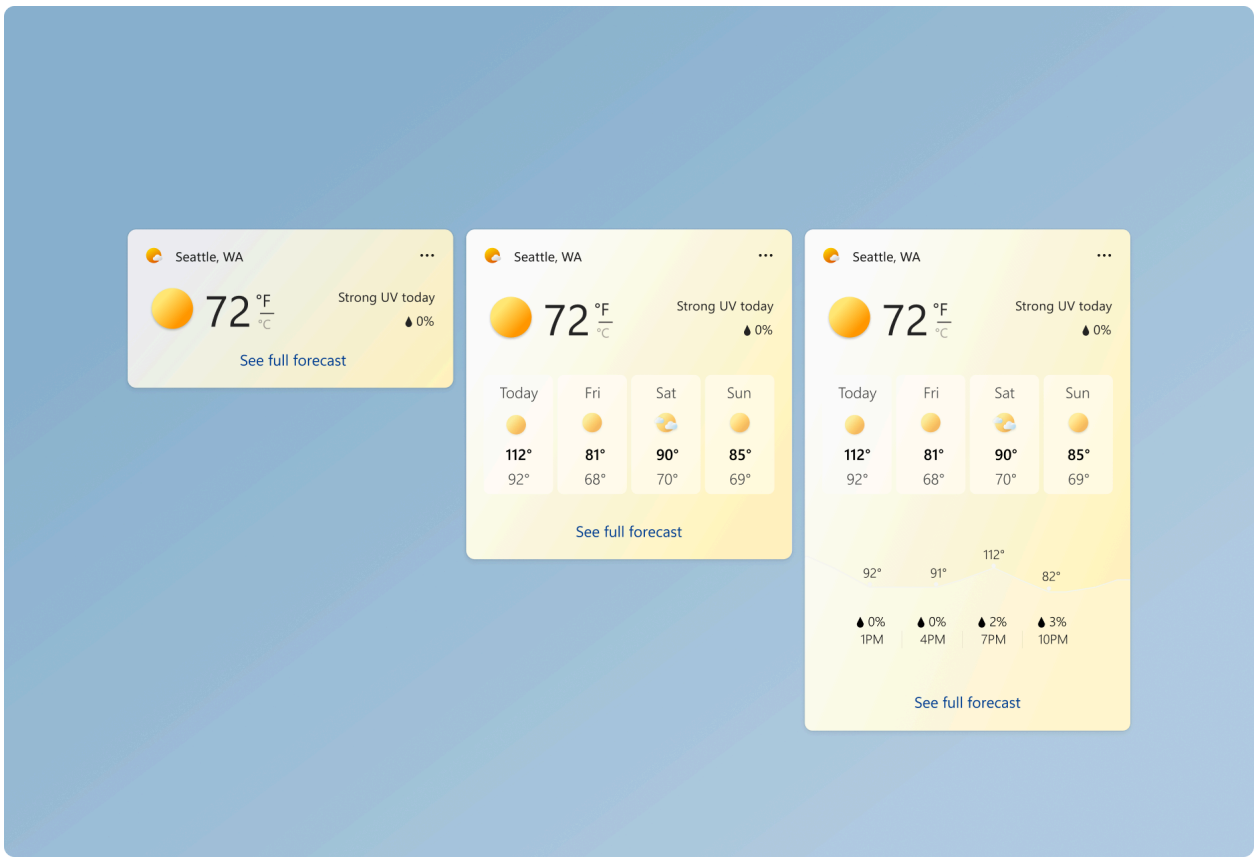
Widget design fundamentals

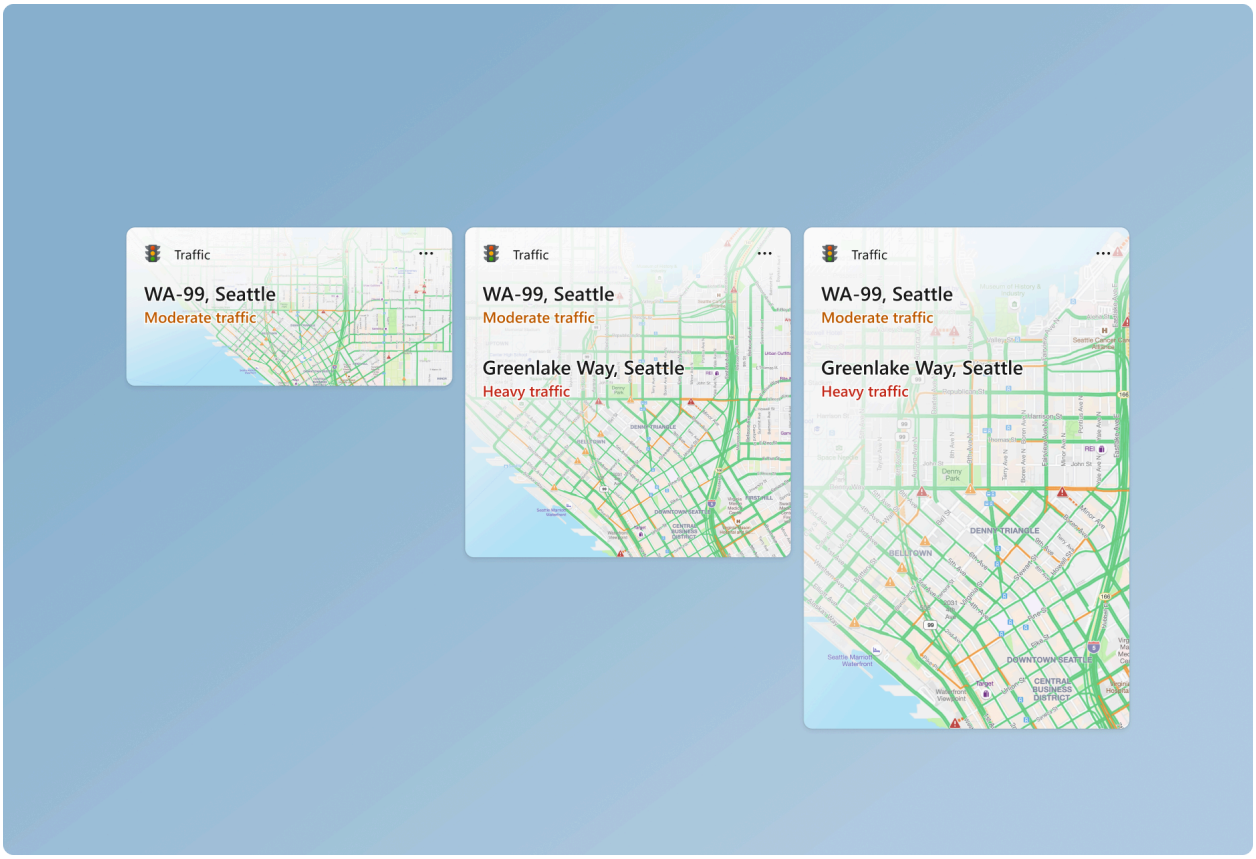
Article • 11/10/2023

This article provides detailed guidance for designing the UI for a Windows widget.

Widget sizes

Text
Text
Text


Seattle, WA
72 °F °C
Strong UV today
0%
See full forecast
Seattle, WA
72 °F °C
Strong UV today
0%
Today Fri Sat Sun
112° 81° 90° 85°
92° 68° 70° 69°
See full forecast
Seattle, WA
72 °F °C
Strong UV today
0%
Today Fri Sat Sun
112° 81° 90° 85°
92° 68° 70° 69°
112°
92° 91° 82°
0% 1PM
0% 4PM
2% 7PM
3% 10PM
See full forecast



Widgets provide three sizes for the user to choose from. It is recommended that you create and consider all 3 sizes and adapt your design specifically for each size. Small and medium sizes provide better discoverability as they get surfaced more often within the dynamic feed. Large sizes are useful for displaying more in-depth information. Supporting multiple sizes allows flexibility in users customizing the widgets they choose to pin to the widgets board.

Small

The widget principles *glanceable* and *focused* become more important in design decisions made for the small size widget. The small size widget should not try to force all of the functionality that could comfortably fit in a large widget. Focus on one user interaction or piece of key information that can be surfaced here with 1 touch target.

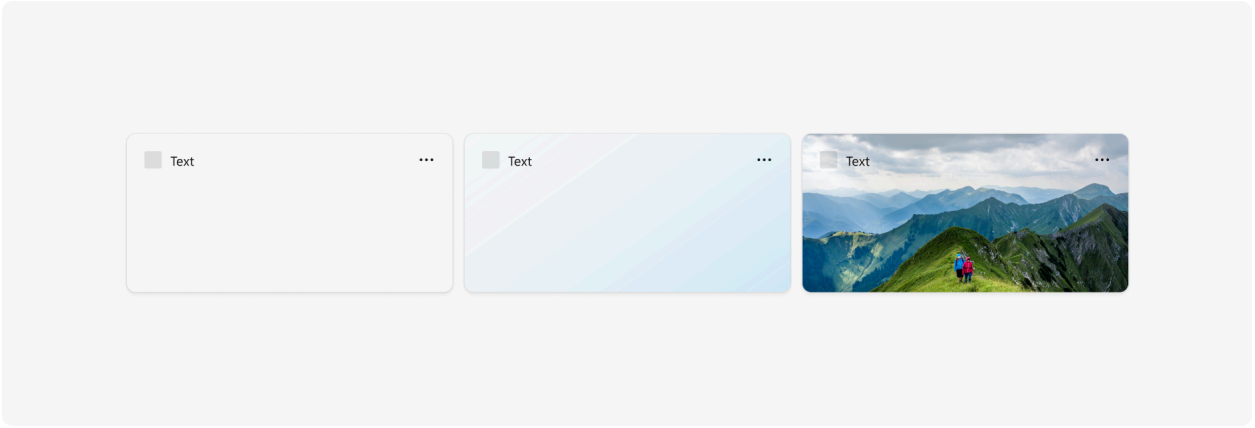
Medium

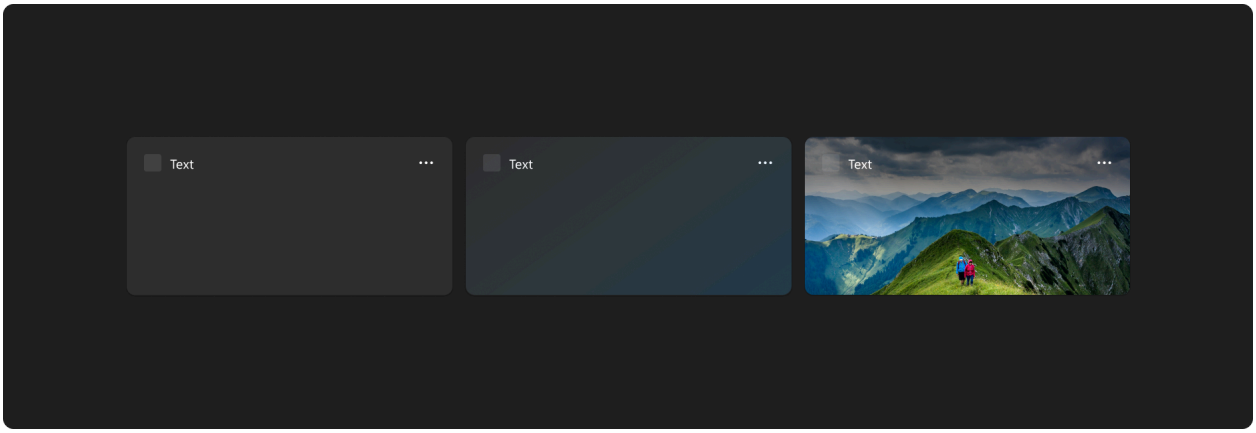
The medium size widget allows more room compared to the small, and so more functionality or additional information can be included. The medium widget could also provide the same focused experience as the small widget, but provide 2-3 touch targets.

Large

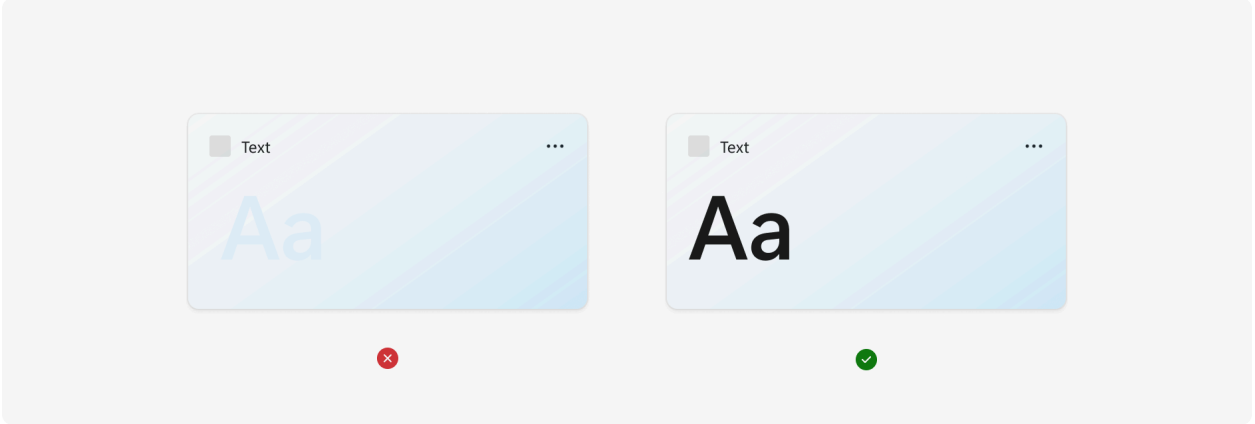
Large sizes allow for more information to be presented, but the content should still be focused and easily consumable. Alternatively, a large size card could highlight one image or topic and have a more immersive experience. The large size should have no more than 3-4 touch targets.

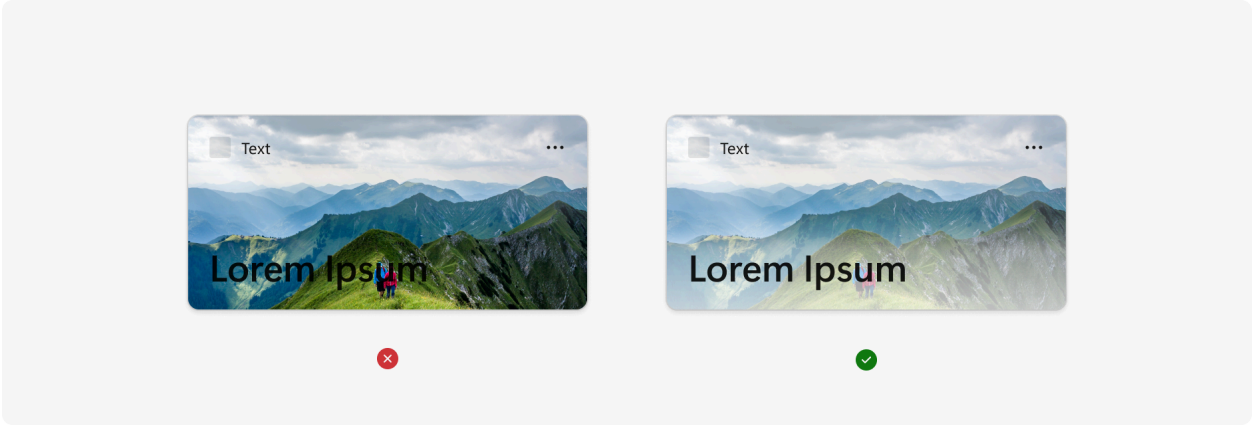
Color and theming





Windows 11 supports two color modes: light and dark. Each mode consists of a set of neutral color values that are automatically adjusted to ensure optimal contrast. For each widget size you support, make sure to create separate designs for light and dark themes so that the widget integrates seamlessly within the wider operating system and user's theme choice. The widget background supports customization with either a solid light/dark background, gradient tint, or image background.

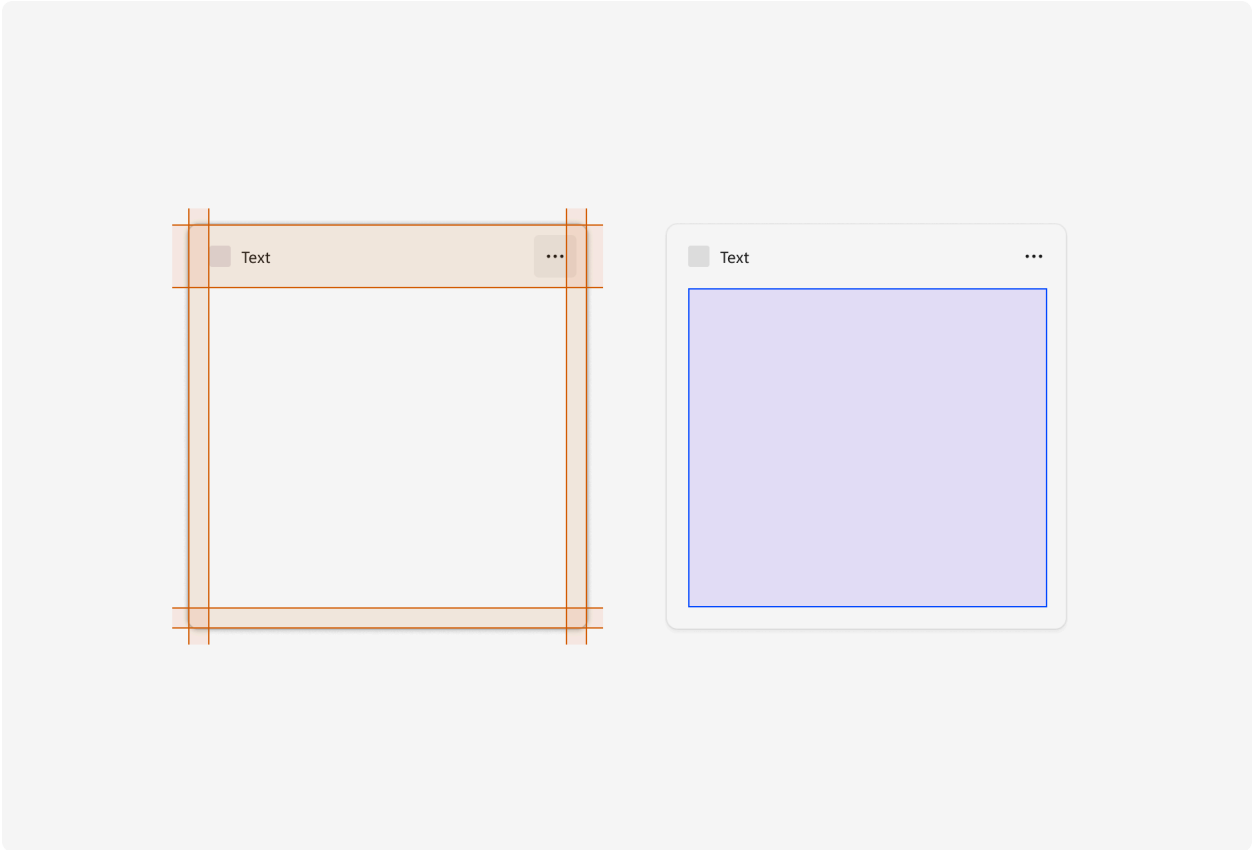




When choosing background colors, images, and content, make sure that there is enough color contrast to ensure legibility and accessibility.

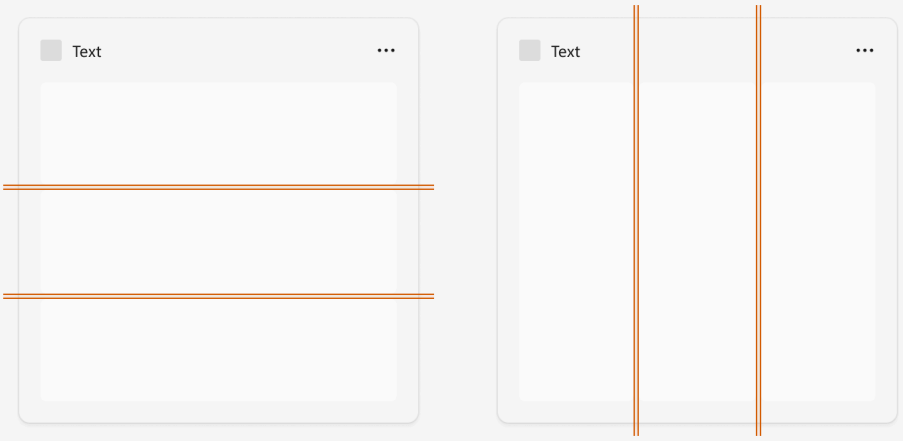
The Web Content Accessibility Guidelines (WCAG) 2.0 level AA requires a contrast ratio of at least 4.5:1 for normal text and 3:1 for large text. WCAG 2.1 requires a contrast ratio of at least 3:1 for graphics and user interface components (such as form input borders). WCAG Level AAA requires a contrast ratio of at least 7:1 for normal text and 4.5:1 for large text. Large text is defined as 14 point (typically 18.66px) and bold or larger, or 18 point (typically 24px) or larger.

Margins



Each widget has a 16px margin around it and a 48px Attribution area in which content cannot be placed. The only component that can live in the right side margin and bottom

margin are the pagination dots. For examples of the positioning of the pagination dots, see the pagination section of Widget interaction design guidance.



For widgets that use containers, the gutter between each element is 4px, and the containers should touch the edges of the margins. Your content should also use spacing and sizing values of Multiples of Four Px to achieve a clean, pixel perfect design across different screen resolutions.

You should also consult the guidance for spacing and gutters in Content design basics for Windows apps when designing your content.

Typography

The quick brown fox jumped over the lazy dog.

The quick brown fox jumped over the lazy dog.

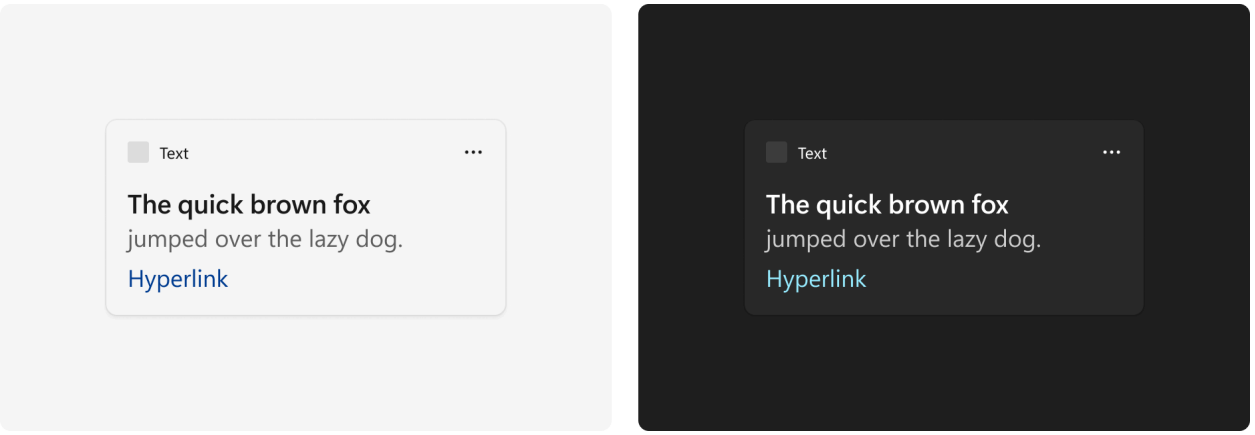
For accessibility, the following table presents the text of the table shown in the image above.

Expand table

Example	Size / Line height	Adaptive cards formula
Caption	12/16 epx	Small, Lighter
Body	14/20 epx	Default, Lighter
Body (for hyperlinks)	14/20 epx	Default, Lighter, Accent
Body Strong	14/20 epx	Default, Bolder
Body Large	18/24 epx	Medium, Lighter
Body Largest	18/24 epx	Medium, Bolder
Subtitle	20/28 epx	Large, Bolder
Title	28/36 epx	Extra Large, Bolder

Segoe UI is the typeface used in Widgets and across Windows. The above type ramp includes the formulations of how to properly set the right styles in the Adaptive Cards Designer. Typeface styling should not deviate from the specified formulas above. For

more information on using the Adaptive Cards Designer to create widget templates, see Create a widget template with the Adaptive Card Designer.



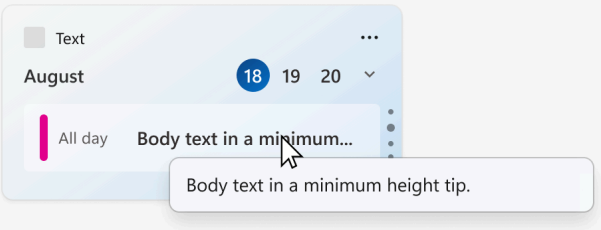
Within the Adaptive Cards Designer, titles and body copy use the default color associated with the widget theme. An additional option to differentiate title from body copy further is to use the subtle version of the default color. The accent color is only used for hyperlinks.

Iconography

Profile pictures

If your widget includes showing user profiles (for example, a social media feed or stream) use one of the following allowed circle profile sizes: 96x96px, 48x48px, 32x32px, or 24x24px.

Tool tips



Tool tips can be used when title text gets truncated in the widget. For best practices, text should fit neatly within the widget space and not need truncation however, that may not always happen depending on scenarios like language localization, system text scaling, or when quoting something (i.e. article title, name of a song). This does not apply to body text on a widget.

Widget states and built-in UI components

Article • 11/10/2022

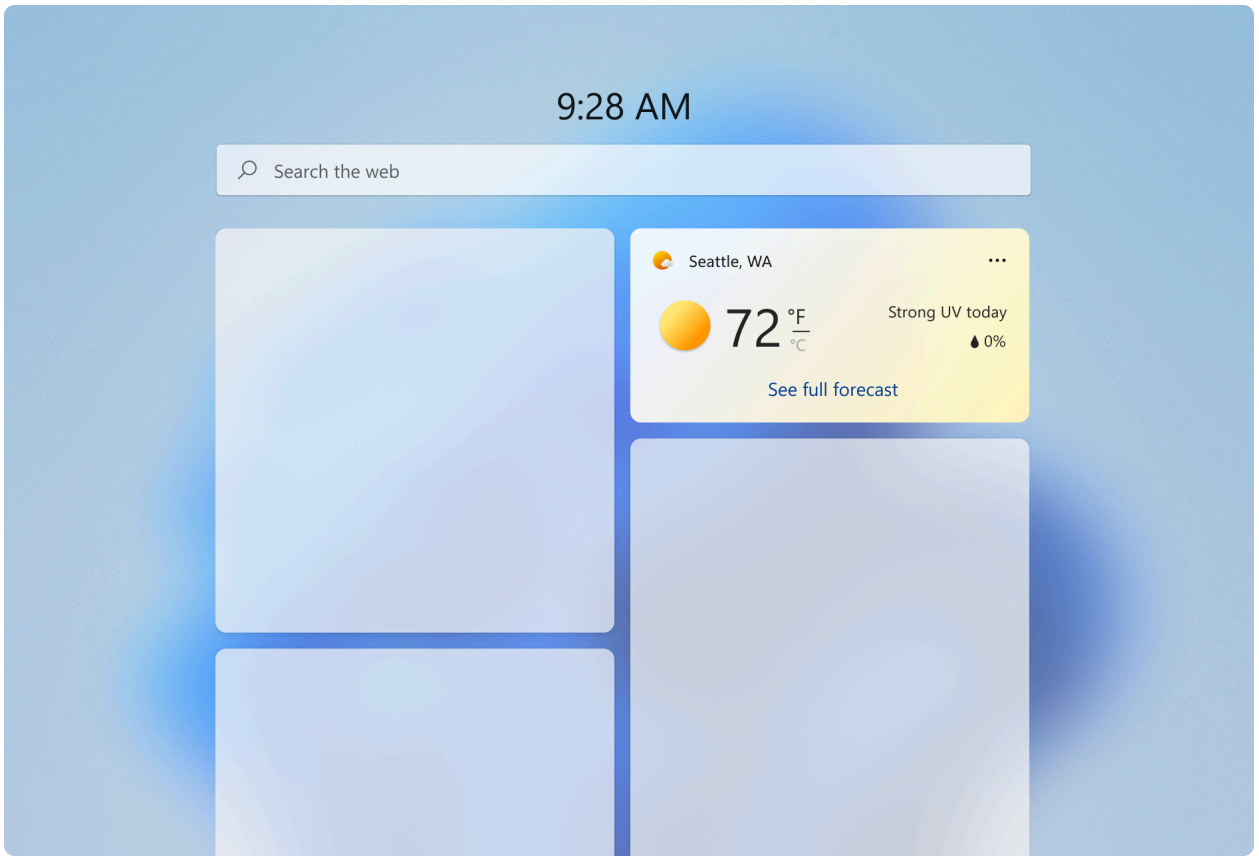
> **Note**
>
> **Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.**

> **Important**
>
> The feature described in this topic is available in Dev Channel preview builds of Windows starting with build 25217. For information on preview builds of Windows, see Windows 10 Insider Preview.

Widget states

When a widget is displayed on the Widgets Board, there are several different states that it can be in, depending on the current state of the widgets board and your app, such as when the widget is loading, when the widget is in an error state, or when the user has customized the widgets layout. Some states are designed and implemented by your app while others are built into the Widgets host. This section shows and describes each widget state. Keep in mind that widgets support both light and dark themes, so the built-in states and the states you customize may both look different depending on the current theme.

Default state



The default state is what the widget looks like when it is running normally. This is the primary user experience for your widget. You design the layout for your widget's default state. Although the UI of your widget's default state may change in response to user configuration, your widget's default state should be fully implemented and should not be empty before user configuration. If your widget requires the user to sign-in, you may want to implement a signed-out state, described below. For design guidance for creating the default state for your widget, see Widget design fundamentals.

DO

- When in the default, active state a widget should feel personal and connected to user.
- Widgets should show engaging content that brings the user value in the current moment.
- Give the user the ability to start interacting with widget right away.
- Provide a UI that reflects the UI of your app, while staying within the design constraints for widgets, in order to maximize consistency and to lessen the learning curve.
- Consider using the user's location to pre-populate the data for content like sports and suggested calendars to add instead of generic data.
- Allow ample breathing space between elements.

DON'T

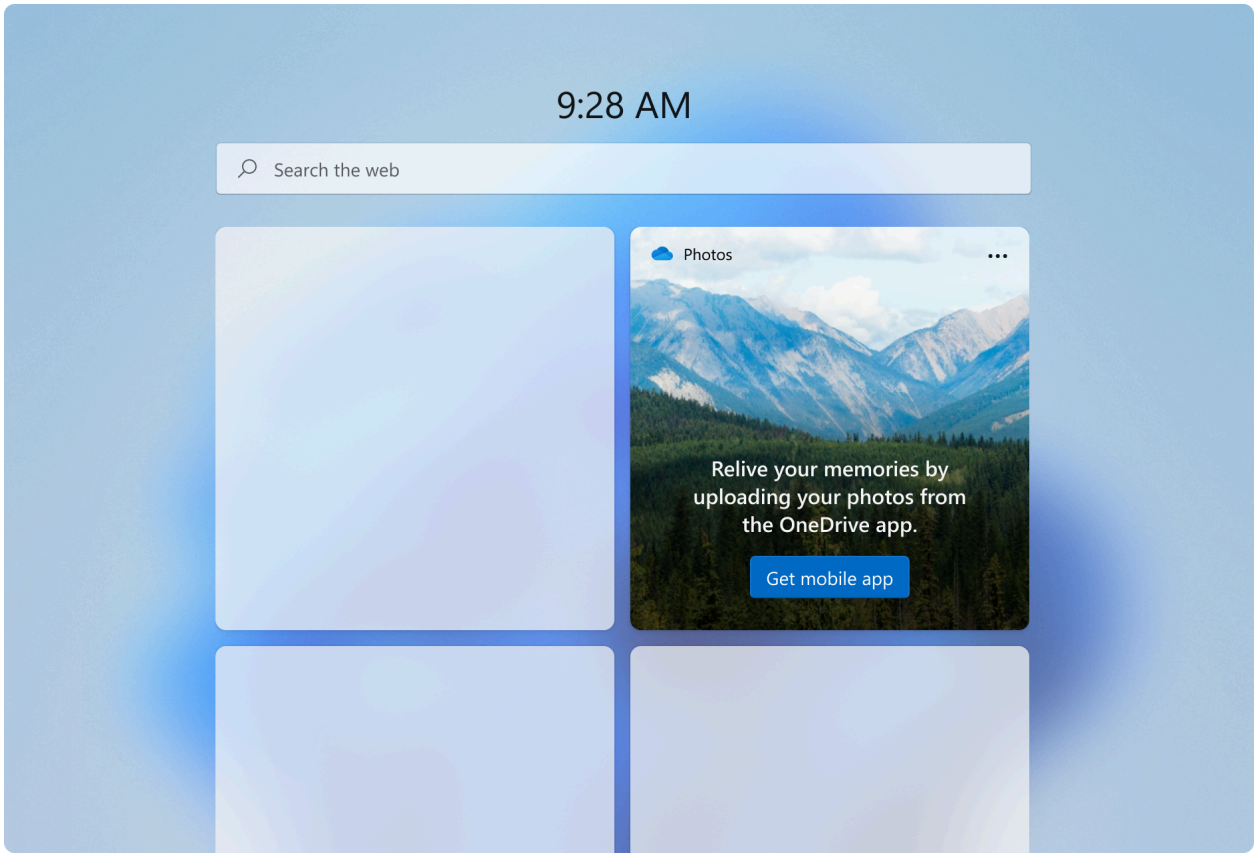
- Use your widget for generic commercial offers. The content should reflect the user's desires and intent.
- Avoid busy, complex layouts.

Aim for comfortable information density and healthy negative space within each widget size to help with the glance and go model. If you have a lot of information to include, consider the next size up to show more content. Also consider how difficult/easy the content will be for a user to glance and consume.

Consider adding moments of surprise + delight to your widgets to elevate the experience. For example, for the Family or Calendar widgets you could highlight a child's birthday via different visual treatments.

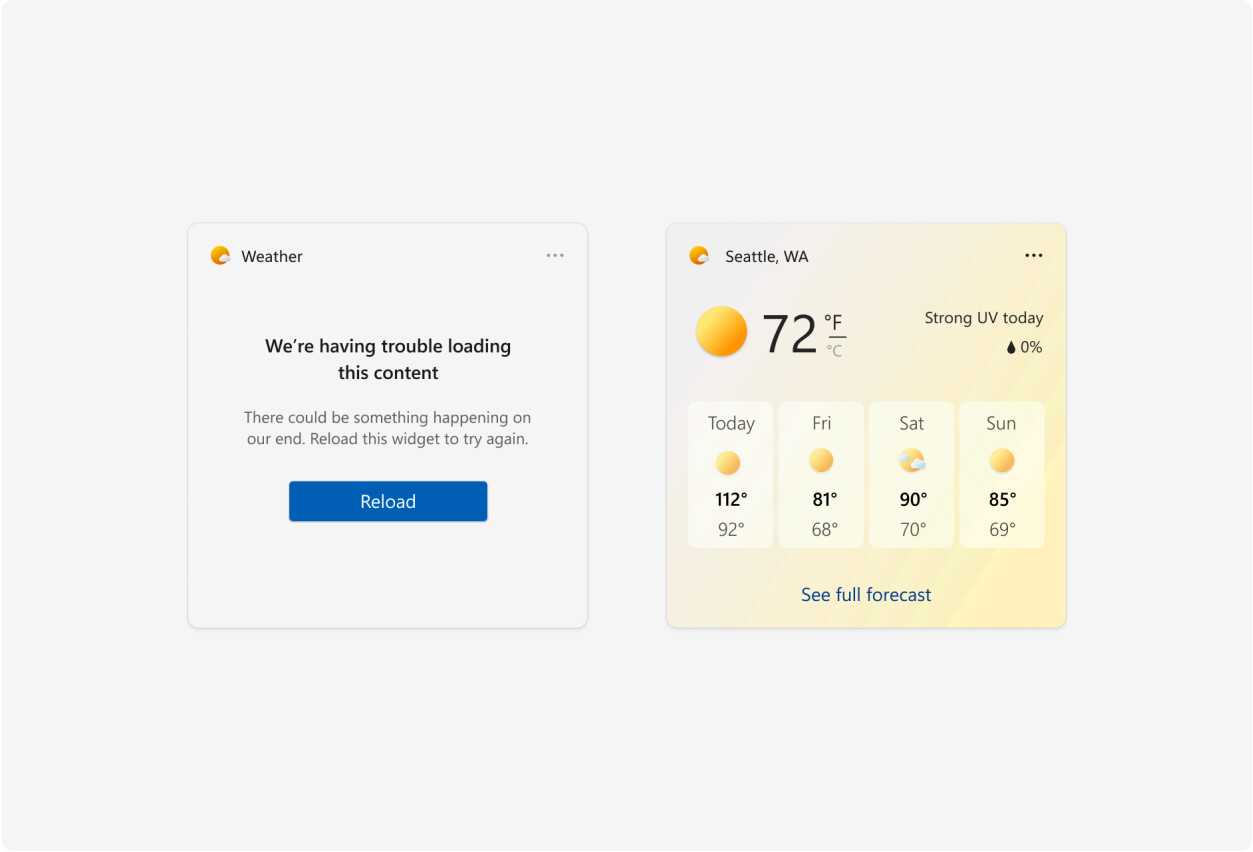
This is the state with the most design flexibility. Use the guidelines in this articles as well as Widget principles, Widget design fundamentals, and Widget interaction design guidance to design how you want to layout your content.

Signed-out state (for widgets that require authentication)



Some widget scenarios may require that the user must sign in or perform other actions to be able to see personalized widget content. When the user is not signed in, you should consider presenting non-personalized content.

Error state - system provided



If for some reason, the Widgets Board can't retrieve the layout or data for a widget, it will display an error state. Windows will show the widget header with an error message and a reload button. This message will look the same for every widget.

If there is cached content available to be displayed, the widget header will show when data was last refreshed in the following format:

1. The number of minutes if less than an hour
2. Rounding to the nearest hour if over an hour

Long widget partner names will get truncated while showing a cached message at 15 characters max.

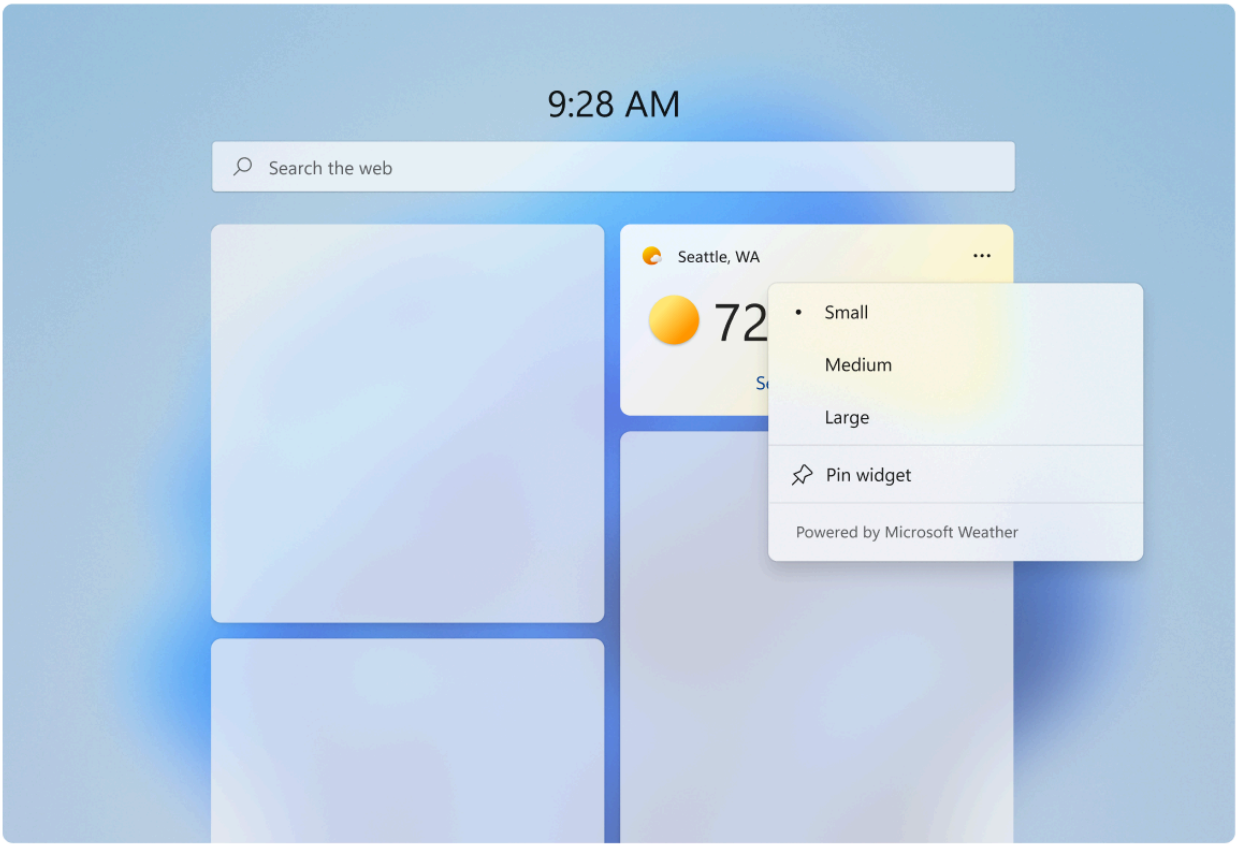
Customization state

Starting with Windows App SDK 1.4, widgets can provide a customization template that provides controls for the user to customize the widget's appearance or the data displayed by the widget. The customization UI is defined in a JSON template.

Built-in widget UI components

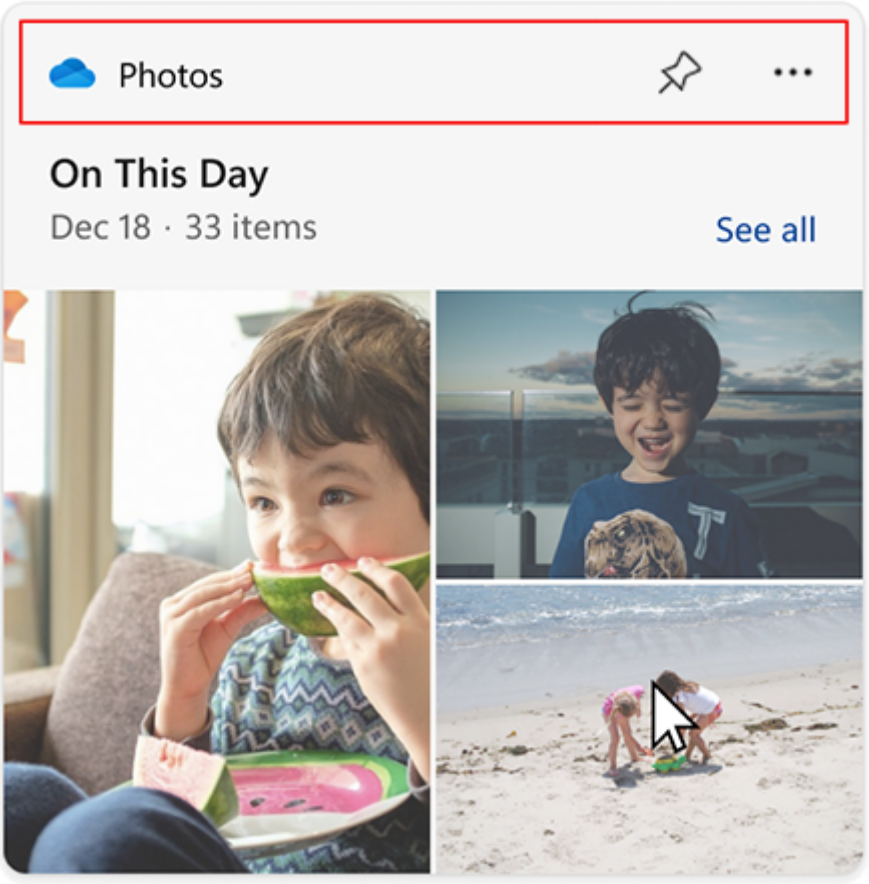
Some UI elements of a widget are built into the widgets experience and, while these elements are not customizable by widget providers, it's important to be aware of what these elements are and how they behave.

Context menu (system provided)



The context menu is displayed when the user clicks on the three-dot icon on the top right. This menu allows users to select their preferred widget size and access the widget's configuration state. Partners will use the same template widget register “powered by ___”.

Attribution area



The attribution area is rendered by the widgets board based on the widget name and icon provided during widget registration. For more information on registering widgets, see Widget provider package manifest XML format.

Widget interaction design guidance

Article • 11/10/2022

> **Note**
>
> **Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.**

> **Important**
>
> The feature described in this topic is available in Dev Channel preview builds of Windows starting with build 25217. For information on preview builds of Windows, see Windows 10 Insider Preview.

This article provides detailed guidance for designing interaction for Windows widgets.

Navigation

A widget should be glanceable and focused, and should represent a single aspect of the app's primary purpose. Widgets may provide one or more calls to action. When the user clicks on a call to action, the widget should launch the associated app or website instead of implementing the action in the widget itself. A widget has only one primary page that can house multiple interactions. Clicking on an item in the widget should never take you to a completely different view of the widget. For example, in a weather widget you might show the weather for multiple days but clicking on one of the days will not expand details inline, but will instead launch the app or web.

The following are the maximum number of touch points recommended for each supported widget size.

Expand table

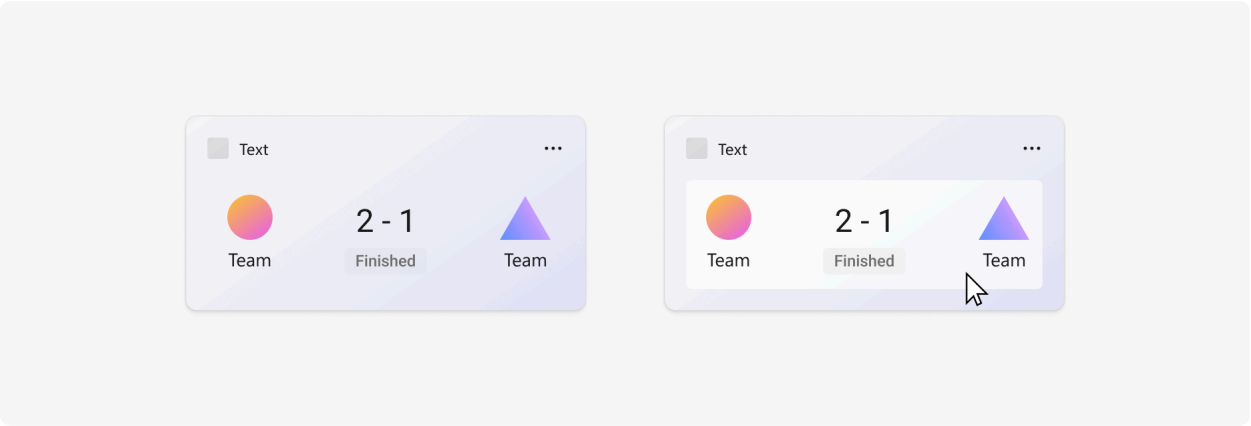
Widget size	Maximum touchpoints
small	1
medium	3
large	4

The following navigation elements are not supported in Windows Widgets:

- Pivots will not be supported within widgets
- L2 Pages will not be supported within widgets
- Vertical or horizontal scrolling will not be supported within Widgets

Containers

The following images show example uses of container elements in a widget template. The containers group visual elements into columns and rows to create a hierarchical grid structure.



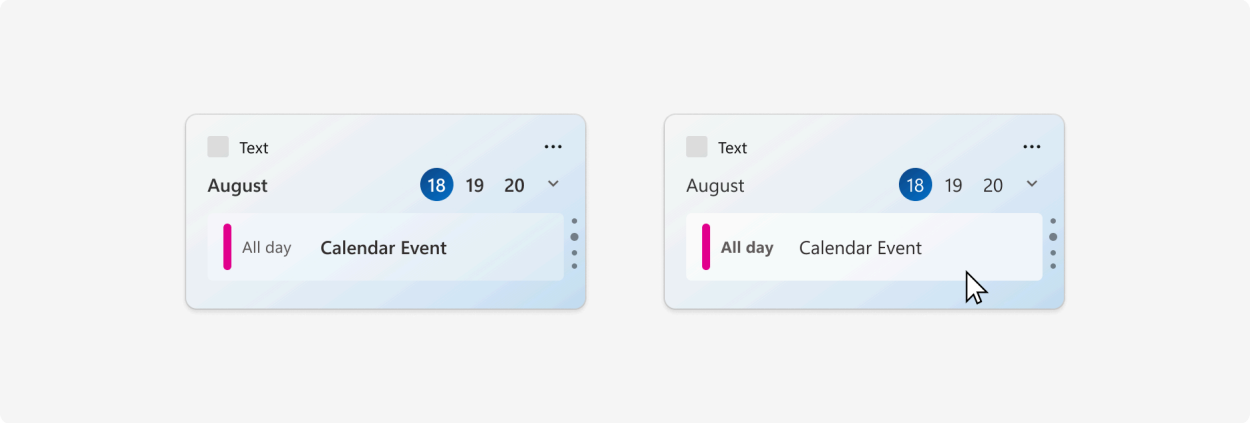
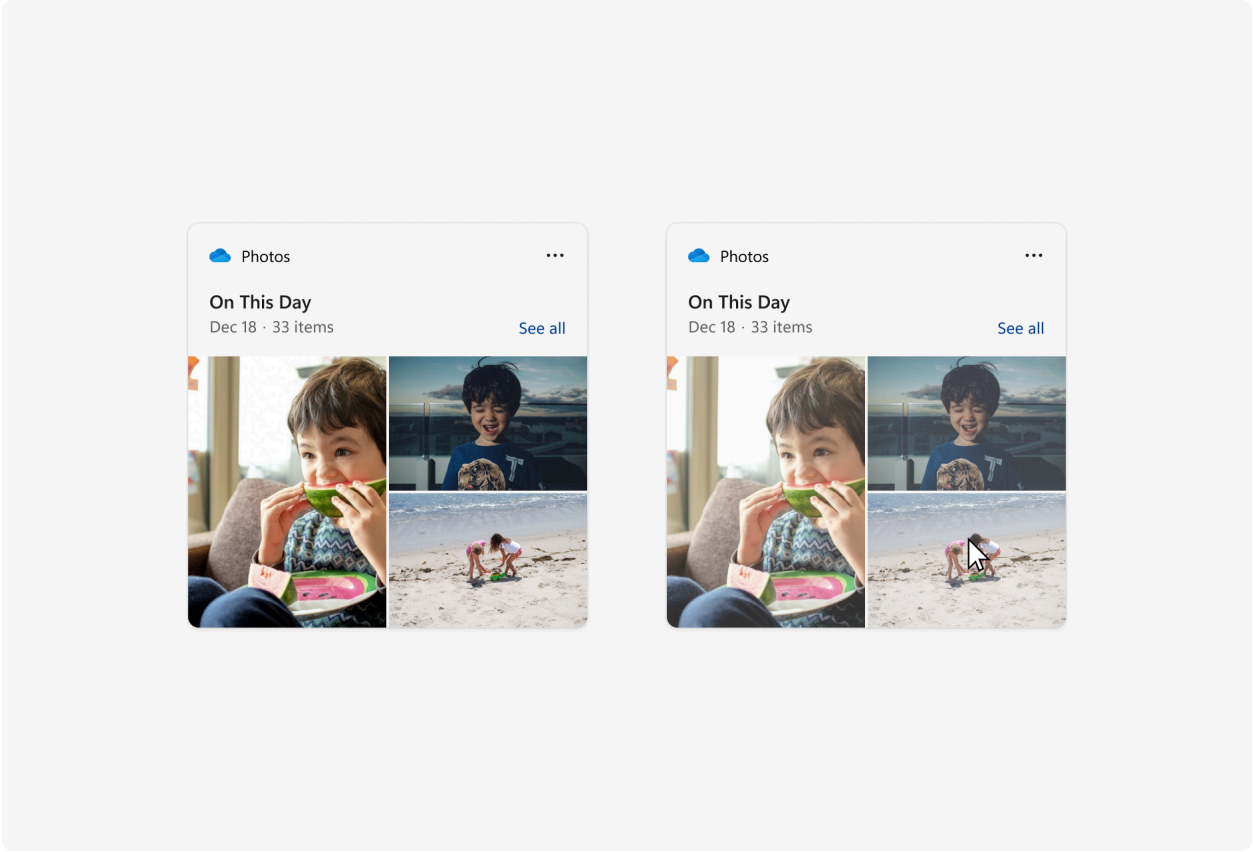


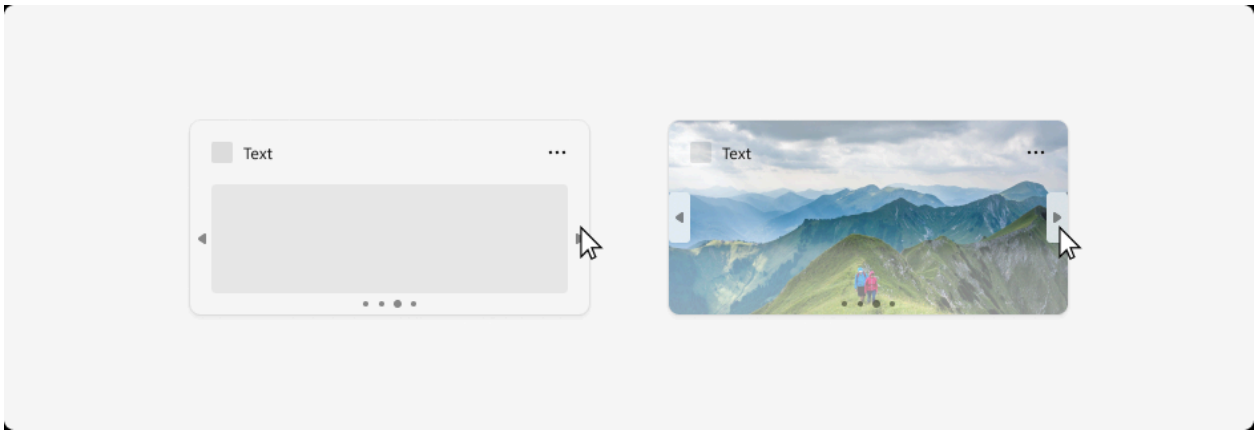
Image links

The following images show example uses of image link elements in a widget template.



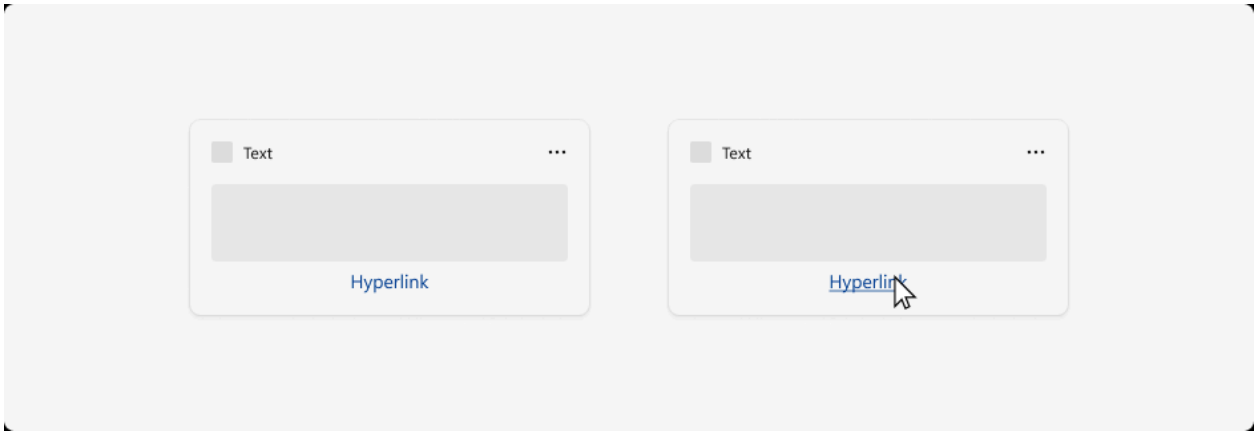
Pagination

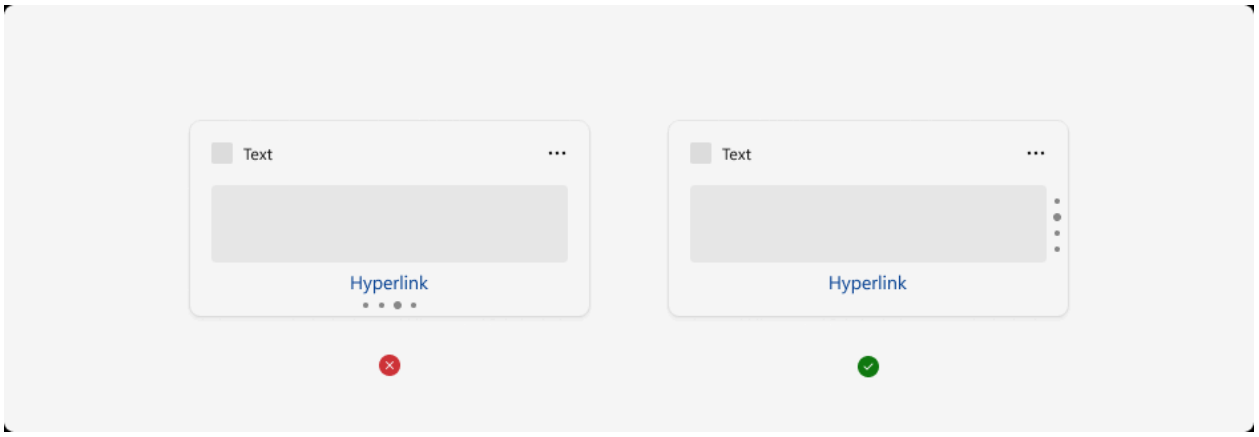
The following images show examples of pagination in a widget template. The pagination controls can be aligned horizontally or vertically. Navigation arrows appear in response to a cursor hover.



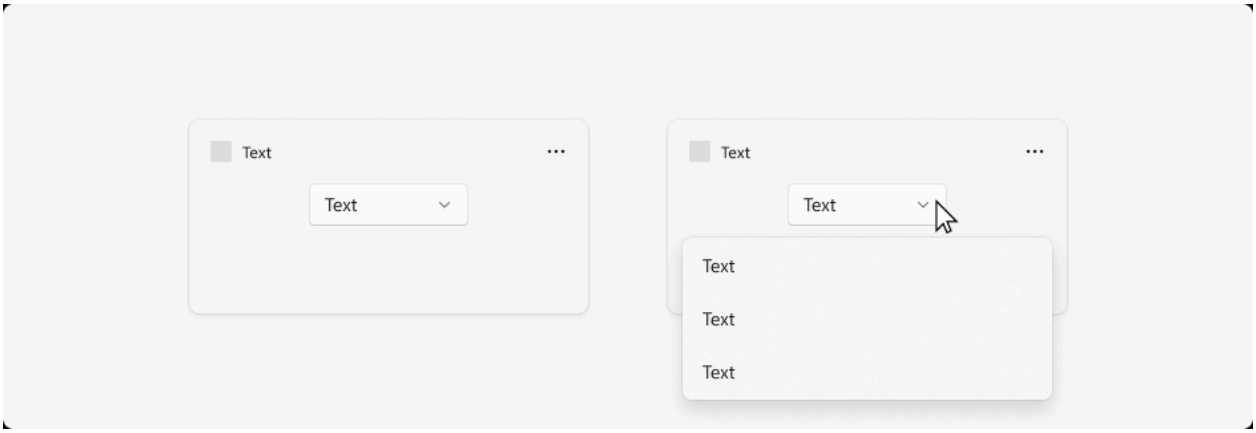
Hyperlinks

The following images show example of hyperlinks in a widget template.





Dropdown menus



Widgets are able to extend slightly beyond their widget size temporarily if the user is interacting with a menu or dropdown. The menu behavior should be light dismiss and close the menu if a user click outside of the menu / dropdown area.

Create a widget template with the Adaptive Cards Designer

Article • 01/24/2025

> **Note**
>
> **Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.**

> **Important**
>
> The feature described in this topic is available in Dev Channel preview builds of Windows starting with build 25217. For information on preview builds of Windows, see Windows 10 Insider Preview.

The UI and interaction for Windows Widgets are implemented using Adaptive Cards. Each widget provides a visual template and, optionally, a data template which are defined using JSON documents that conform to the Adaptive Cards schema. This article walks you through the steps to create a simple widget template.

A counting widget

The example in this article is a simple counting widget that displays an integer value and allows the user to increment the value by clicking on a button in the widget's UI. This example template uses data binding to automatically update the UI based on the data context.

Apps need to implement a widget provider to generate and update the widget template and/or data and pass them to the widget host. The article Implement a widget provider in a win32 app provides step-by-step guidance for implementing the widget provider for the counting widget that we will generate in the steps below.

The Adaptive Cards Designer

The Adaptive Cards Designer is an online interactive tool that makes it easy to generate JSON templates for Adaptive Cards. Using the designer, you can see the

rendered visuals and the data binding behavior in real-time as you build your widget template. Follow the link to open the designer, which will be used for all of the steps in this walkthrough.

Create an empty template from a preset

At the top of the page, from the **Select host app** dropdown, choose Widgets Board. This will set the container size for the Adaptive Card to have a size that is supported for widgets. Note that widgets support small, medium, and large sizes. The size of the default template preset is the correct size for a small widget. Don't worry if the content overflows the borders because we will be replacing it with content designed to fit inside the widget.

There are three text editors at the bottom of the page. The one labeled **Card Payload Editor** contains the JSON definition of your widget's UI. The editor labeled **Sample Data Editor** contains JSON that defines an optional data context for your widget. The data context is bound dynamically to the Adaptive Card when the widget is rendered. For more information about data binding in Adaptive Cards, see Adaptive Cards Template Language.

The third text editor is labeled **Sample Host Data Editor**. Note that this editor may collapse below the page's other two editors. If so, click the + to expand the editor. Widget host apps can specify host properties that you can use in your widget template to dynamically display different content based on the current property values. The Widgets Board supports the following host properties.

Expand table

Property	Value	Description
host.widgetSize	"small", "medium", or "large"	The size of the pinned widget.
host.hostTheme	"light" or "dark"	The current theme of the device on which the Widgets Board is displayed.
host.isSettingsPayload	true or false	When this value is true, the user has clicked on the **Customize widget** button in the widget context menu. You can use this property value to display customization settings UI elements. This is an alternative method to using IWidgetProvider2.OnCustomizationRequested to alter the JSON payload in the widget provider

Property	Value	Description
		app. For more information, see Implementing widget customization.
host.isHeaderSupported	true or false	When this value is true, header customization is supported. For more information, see isHeaderSupported.
host.isHeader	true or false	When this value is true, the host is requesting a payload specifically for rendering of the widget header.
host.isWebSupported	true or false	When this value is false, the host does not currently support loading a widget's web content. When this occurs, web widgets will display the fallback JSON payload supplied by the widget provider, but this value can be use to further customize the content. For more information, see Web widget providers
host.isUserContextAuthenticated	true or false	When this value is false, the only action that is supported is Action.OpenUrl. The value of *isUserContextAuthenticated* can be used to adjust widget content appropriately, given the interactivity limitations.

The **Container size** and **Theme** dropdowns next to the **Select host app** dropdown at the top of the page allow you to set these properties without manually editing the sample host JSON in the editor.

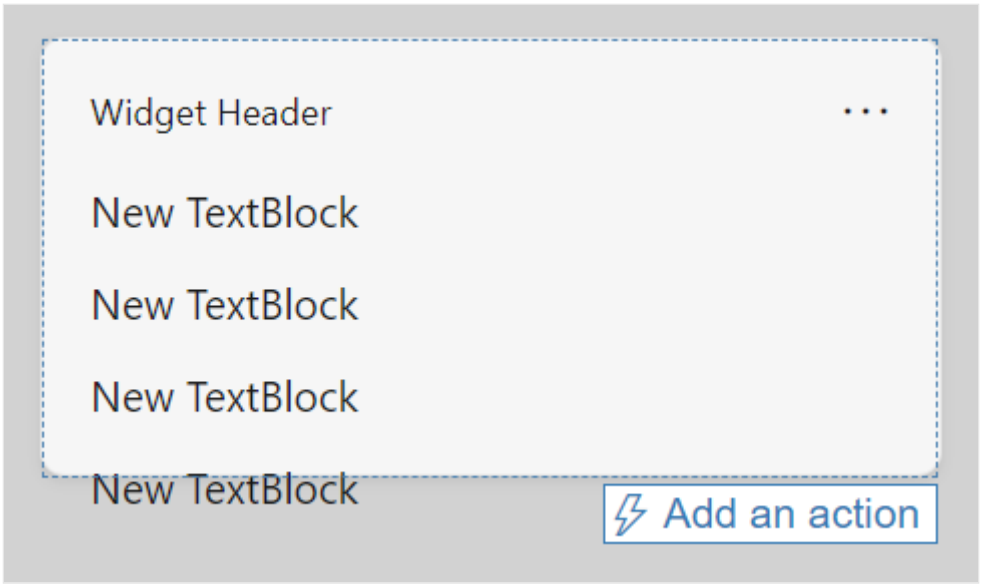
Create a new card

In the upper left corner of the page, click **New card**. In the *Create* dialog, select **Blank Card**. You should now see an empty Adaptive Card. You will also notice that the JSON document in the sample data editor is empty.

The counting widget that we will create is very simple, only consisting of 4 TextBlock elements and one action of type Action.Execute, that defines the widget's button.

Add TextBlock elements

Add four **TextBlock** elements by dragging them from the **Card elements** pane on the left of the page onto the blank adaptive card in the preview pane. At this point, the widget preview should look like the following image. The content again overflows outside of the widget borders, but this will be fixed in the following steps.



Implementing conditional layout

The **Card Payload Editor** has been updated to reflect the **TextBlock** elements that we added. Replace the JSON string for the **body** object with the following:

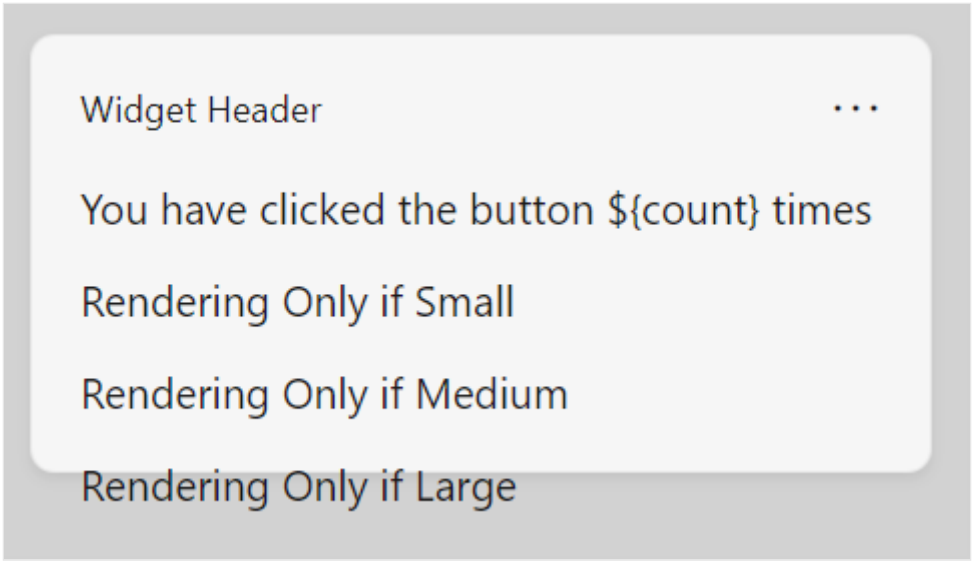
JSON

```
"body": [
    {
        "type": "TextBlock",
        "text": "You have clicked the button ${count} times"
    },
    {
        "type": "TextBlock",
        "text": "Rendering only if medium",
        "$when": "${$host.widgetSize==\"medium\"}"
    },
    {
        "type": "TextBlock",
        "text": "Rendering only if small",
        "$when": "${$host.widgetSize==\"small\"}"
    },
    {
        "type": "TextBlock",
        "text": "Rendering only if large",
        "$when": "${$host.widgetSize==\"large\"}"
    }
]
```

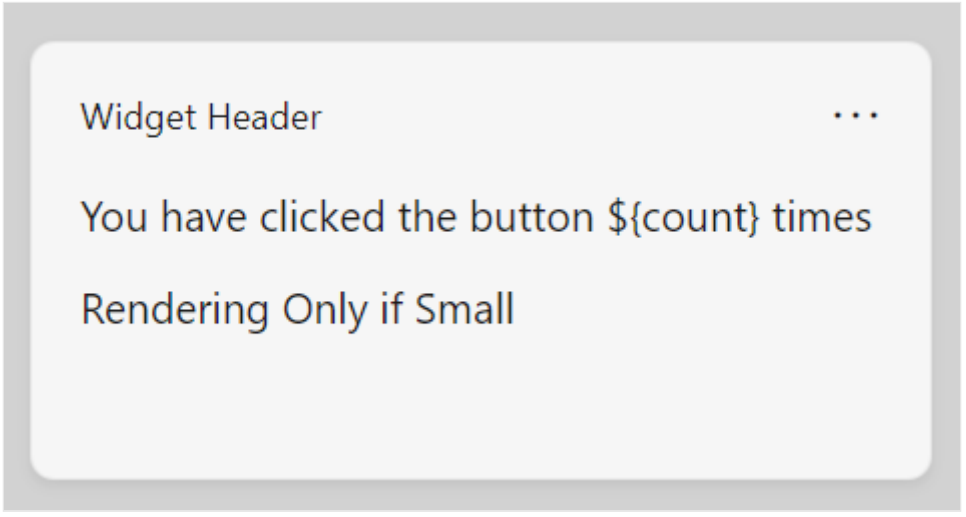
In the Adaptive Cards Template Language, the `$when` property specifies that the containing element is displayed when the associated value evaluates to true. If the value evaluates to false, the containing element is not displayed. In the **body** element in our example, one of the three **TextBlock** elements will be shown, and the other two hidden,

depending on the value of the `$host.widgetSize` property. For more information about the conditionals supported in Adaptive Cards, see Conditional layout with $when.

Now the preview should look like the following image:



Note that the conditional statements aren't being reflected in the preview. This is because the designer isn't simulating the behavior of the widget host. Click the **Preview mode** button at the top of the page to start the simulation. The widget preview now looks like the following image:



From the **Container size** dropdown, select "Medium" and note that the preview switches to only show the **TextBlock** for the medium size. The container in the preview also changes size, demonstrating how you can use the preview to make sure that your UI fits within the widget container for each supported size.

Bind to the data context

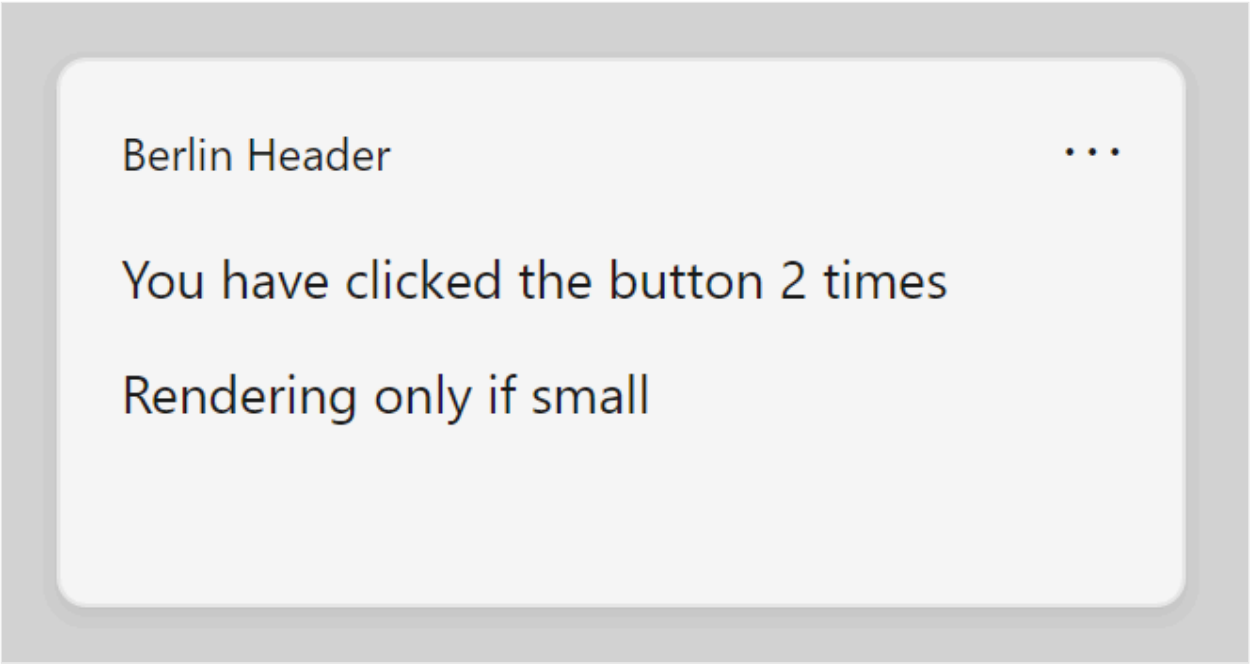
Our example widget will use a custom state property named "count". You can see in the current template that the value for the first **TextBlock** includes the variable reference

`$count`. When the widget is running in the Widgets Board, the widget provider is responsible for assembling the data payload and passing it to the widget host. At design time, you can use the **Sample Data Editor** to prototype your data payload and see how different values impact the display of your widget. Replace the empty data payload with the following JSON.

JSON

```
{"count": "2"}
```

Note that the preview now inserts the value specified for the *count* property into the text for the first **TextBlock**.



Add a button

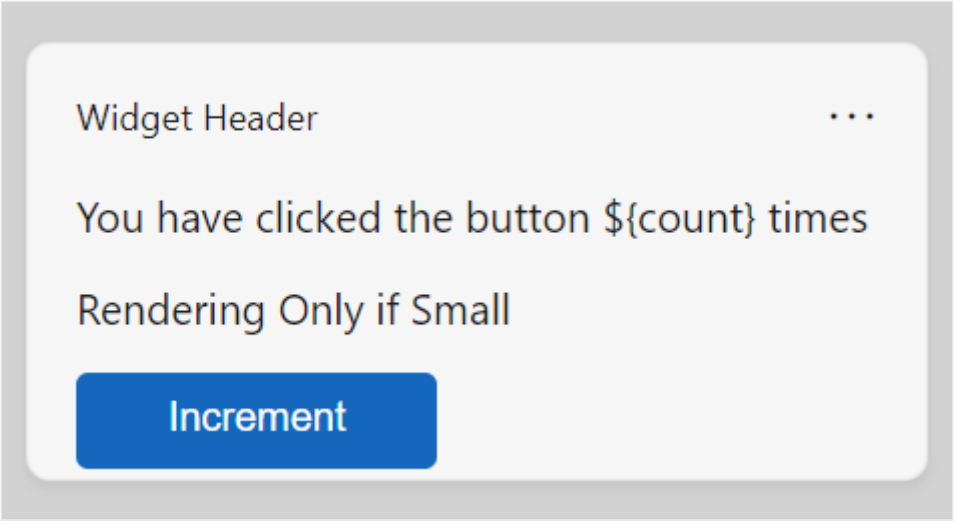
The next step is to add a button to our widget. In the widget host, when the user clicks the button, the host will make a request to the widget provider. For this example, the widget provider will increment the count value and return an updated data payload. Because this operation requires a widget provider, you won't be able to view this behavior in the Adaptive Cards Designer, but you can still use the designer to adjust the layout of your button within your UI.

With Adaptive Cards, interactive elements are defined with **action** elements. Add the following block of JSON directly after the **body** element in the card payload editor. Be sure to add a comma after the closing bracket (]) of the body element or the designer will report a formatting error.

JSON

```
,
"actions": [
    {
        "type": "Action.Execute",
        "title": "Increment",
        "verb": "inc"
    }
]
```

In this JSON string, *type* property specifies the type of action that is being represented. Widgets only support the "Action.Execute" action type. The *title* contains the text that is displayed on the button for the action. The *verb* property is an app-defined string that the widget host will send to the widget provider to communicate the intent associated with the action. A widget can have multiple actions, and the widget provider code will check the value of the verb in the request to determine what action to take.



The complete widget template

The following code listing shows the final version of the JSON payload.

JSON

```
{
    "type": "AdaptiveCard",
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "version": "1.6",
    "body": [
    {
      "type": "TextBlock",
      "text": "You have clicked the button ${count} times"
    },
    {
```

```
      "type": "TextBlock",
       "text": "Rendering Only if Small",
      "$when": "${$host.widgetSize==\"small\"}"
    },
    {
      "type": "TextBlock",
      "text": "Rendering Only if Medium",
      "$when": "${$host.widgetSize==\"medium\"}"
    },
    {
      "type": "TextBlock",
      "text": "Rendering Only if Large",
      "$when": "${$host.widgetSize==\"large\"}"
    }
    ],
   "actions": [
    {
      "type": "Action.Execute",
      "title": "Increment",
      "verb": "inc"
    }
  ]
}
```

Settings payload example

The following code listing shows a simple example of a JSON payload that uses the **host.isSettingsPayload** property to display different content when the user has clicked the **Customize widget** button.

JSON

```
{
    "type": "AdaptiveCard",
    "body": [
    {
        "type": "Container",
        "items":[
            {
                "type": "TextBlock",
                "text": "Content payload",
                "$when": "${!$host.isSettingsPayload}"
            }
        ]
    },
    {
        "type": "Container",
        "items":[
            {
                "type": "TextBlock",
                "text": "Settings payload",
```

```
                "$when": "${$host.isSettingsPayload}"
            }
        ]
    }
],
"actions": [
    {
    "type": "Action.Submit",
    "title": "Increment",
    "verb": "inc",
    "$when": "${!$host.isSettingsPayload}"
    },
    {
    "type": "Action.Submit",
    "title": "Update Setting",
    "verb": "setting",
    "$when": "${$host.isSettingsPayload}"
    }
],
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "version": "1.6"
}
```

Integrate with the widget picker

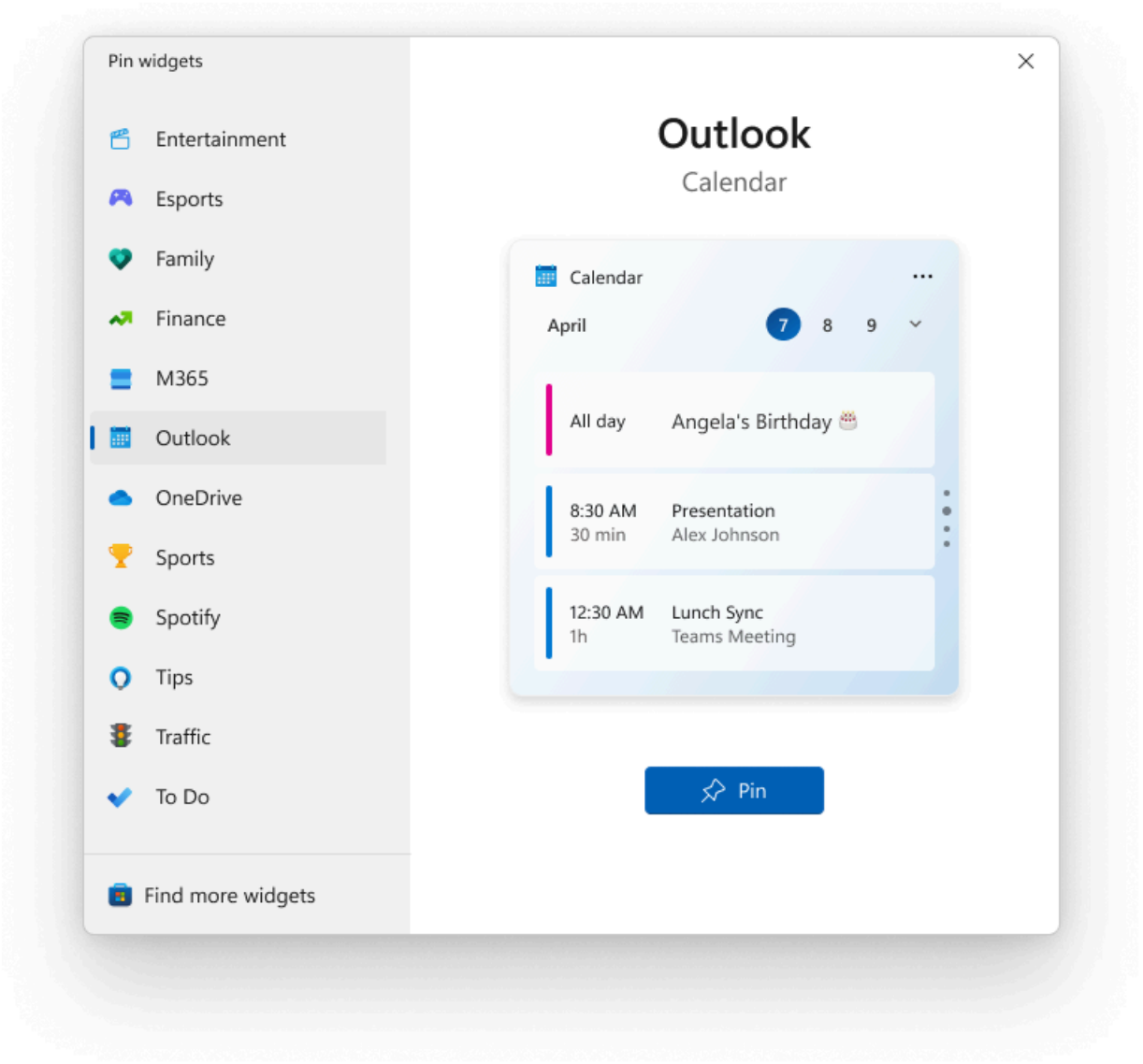
Article • 02/02/2023

In the current release, the only Widgets host is the Widgets Board built into Windows 11. The Widgets Board displays widgets and manages their layout on the board. It also provides a widget picker that allows the user to select which available widgets are visible on the board. This article describes the assets required for a widget to successfully integrate into the widget picker.

Widget screenshot image

Each widget must provide a screenshot image that is displayed as a preview in the widget picker when the widget has focus. The screenshot is specified by the widget provider in the package manifest for the app. For technical information on how to specify an image file to use for the widget screenshot, see Implement a widget provider in a win32 app and Widget provider package manifest XML format.

The following screenshot illustrates the placement of the screenshot image within the widget picker.



Screenshot image requirements

In order to provide a consistent user experience, widget screenshots for the widget picker must follow the following guidelines.

- The screenshot should display the medium size version of your widget.
- The image should be 300 pixels wide and 304 pixels tall.
- The image should have transparent, rounded corners.

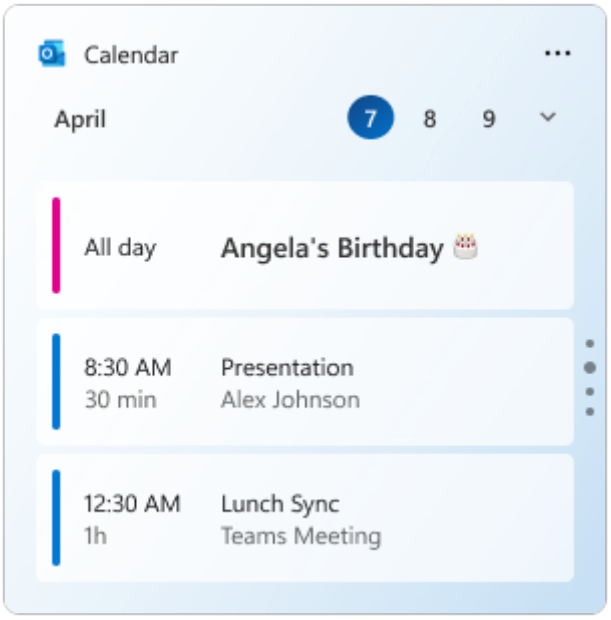
The widget provider manifest includes three different **Screenshot** elements that you can set. The top-level screenshot is required and will be used as the default by the widget picker. The **LightMode** and **DarkMode** sections of the manifest are optional and allow you to specify separate screenshots for your widget in light and dark modes. If you supply one or both of these optional screenshots, the widget picker will use the one that matches the device's current theme. If you do not supply an image specifically for the current theme, the widget picker will use the default, top-level screenshot instead.

You may provide widget screenshots for multiple locales. The locale for each screenshot is specified with the path to the asset relative to your app package.

- Create a directory in your packaging project for the image assets. For example: "Assets".
- The language-neutral or fallback assets can be placed directly in this folder. For example: "Assets/Screenshot.png"
- Language-specific assets can be placed in subdirectories named after the locale to which they apply. For example:
  - "English (United States)" assets should be placed in a subdirectory named "en-us": "Assets/en-us/Screenshot.png"
  - "French (France)" assets should be placed in a subdirectory named "fr-fr": "Assets/fr-fr/Screenshot.png"
- Reference these images in the appxmanifest using the ms-appx: URI scheme. For example: Screenshot="ms-appx:Assets\Screenshot.png"
- If you don't include an image for a specific locale, then the fallback image in the root directory will be used.

The resource loader will automatically choose the language-appropriate asset to display. For more information, see: Load images and assets tailored for scale, theme, high contrast, and others.

The following is an example of a widget screenshot image.



Design resources for Windows apps

Article • 01/07/2025

This topic provides a variety of design and UI-related resources for creating visually appealing and user-friendly Windows applications.

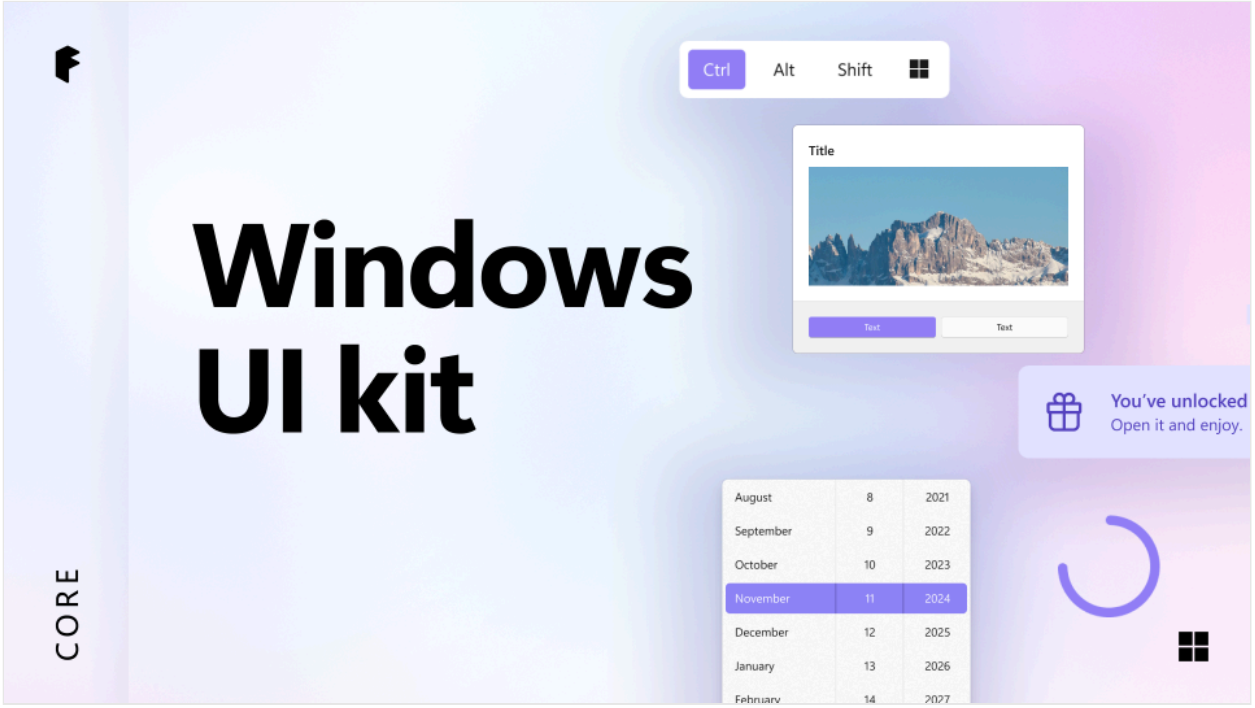
Fonts

The Windows type system helps you create structure and hierarchy in your content in order to maximize legibility and readability in your UI (for more details, see Segoe UI font family).

The following fonts are recommended:

- Segoe UI Variable (see Typography in Windows)
- Segoe Fluent Icons (see Segoe Fluent Icons font)
- Segoe MDL2 (see Segoe MDL2 Assets icons)

Windows UI Kit



The Windows Design Kit for Figma provides a comprehensive toolkit to help developers and designers create visually consistent and engaging Windows apps

The Windows Design Kit is a Figma library that includes:

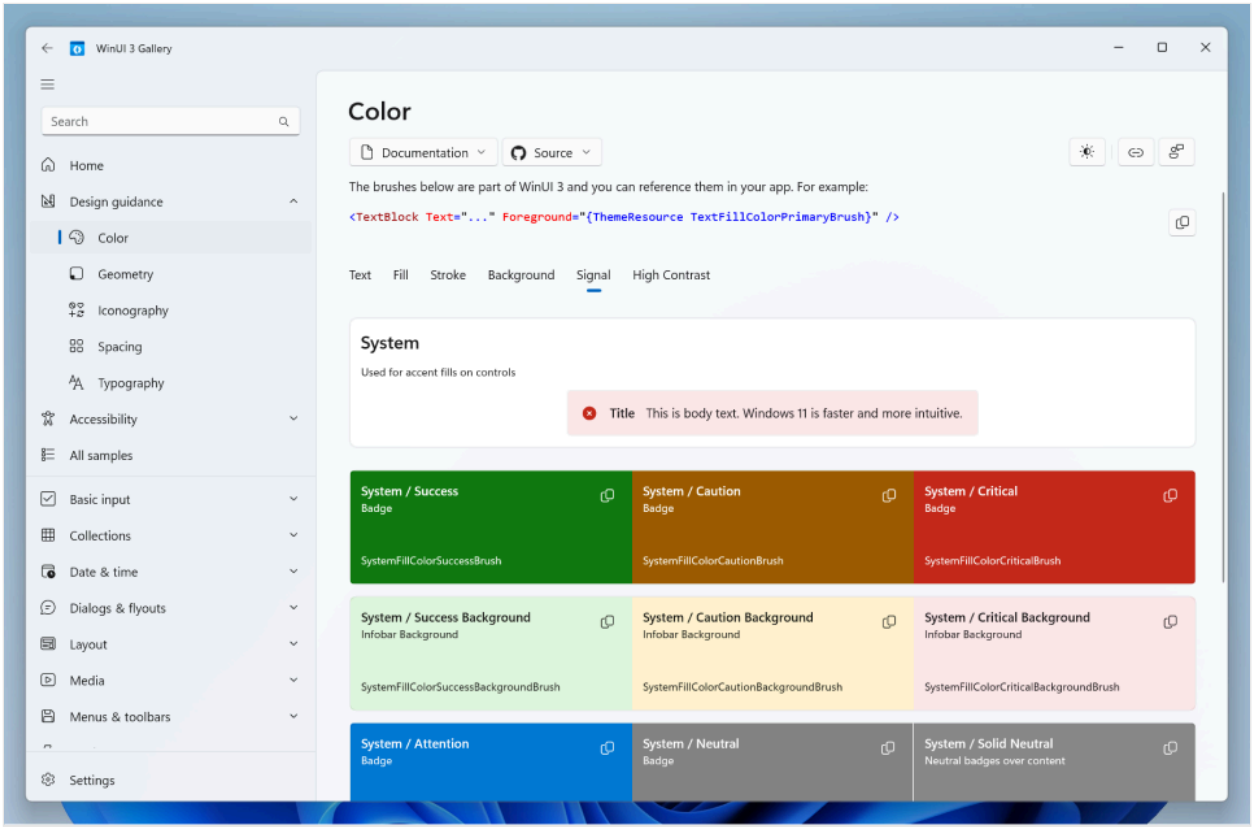
- **UI components**: Prebuilt elements like buttons, toggles, sliders, and more.

- **Design patterns**: Templates for navigation, dialogs, and layouts.
- **Styles and tokens**: Predefined colors, typography, and spacing.

Windows UI Kit for Figma

WinUI Gallery

The Design Guidance section of the WinUI Gallery offers best practices and tools to help you choose and apply the right colors, typography, and icons for your app. The Accessibility section includes code snippets and tips to make your app more inclusive and accessible to all users.



Design Guidance in WinUI Gallery

Samples

Check out the Samples and resources page for complete list of all our Windows app samples.

Develop Windows desktop apps

Article • 02/13/2025

This section of the documentation provides information about APIs and features you can use while developing Windows desktop apps. Some of these features are available by using APIs in the Windows App SDK. Other features are available by using APIs in the Windows OS (via the Windows SDK) and .NET, and don't require use of the Windows App SDK.

> **Important**
>
> If you're working on a UWP app, then refer to Migrate from UWP to the Windows App SDK.

Windows App SDK features

The following table highlights the development features that are provided by the current releases of the Windows App SDK. For more details about the release channels of the Windows App SDK that include each of these features, see Features available by release channel.

Learn about the Windows App SDK

Expand table

Feature	Description
WinUI 3	The premiere native user interface (UI) framework for Windows desktop apps, including managed apps that use C# and .NET and native apps that use C++ with the Win32 API. WinUI 3 provides consistent, intuitive, and accessible experiences using the latest user interface (UI) patterns.
Render text with DWriteCore	Render text using a device-independent text layout system, high quality sub-pixel Microsoft ClearType text rendering, hardware-accelerated text, multi-format text, wide language support, and much more.
Manage resources with MRT Core	Manage app resources such as strings and images in multiple languages, scales, and contrast variants independently of your app's logic.
App lifecycle: App instancing	Control whether multiple instances of your app's process can run at the same time.

Feature	Description
App lifecycle: Rich activation	Process information about different kinds activations for your app.
App lifecycle: Power management	Gain visibility into how your app affects the device's power state, and enable the app to make intelligent decisions about resource usage.
Manage app windows	Create and manage the windows associated with your app.
Push notifications	Send raw notifications and app notifications to your app from the cloud using Azure App Registration identities.
App notifications	Deliver messages to your user with app notifications.
Deployment	Deploy the Windows App SDK runtime with your unpackaged and packaged app

Windows app development features organized by scenario

The following articles provide information to help you get started using features of the full Windows app development platform for common app scenarios, including features provided by the Windows App SDK, the Windows SDK, and the .NET SDK.

- Modernize your existing desktop apps
- User interface and input
- App lifecycle and system services
- Launching Windows apps and managing background tasks
- Communication
- Accessibility
- Audio, video, and camera
- Graphics
- Data and files
- Windows AI and machine learning
- Integrate with Windows
- Devices and sensors
- Security and identity
- Deployment overview

For information about setting up your development environment and getting started creating a new app, see:

- Get started with WinUI

Related topics

- Windows App SDK
- WinUI
- Deployment overview

Accessibility in Windows 11 and Windows 10

Article • 11/28/2023

Build accessibility into your applications to empower people of all abilities

Products and services—including electronic media—are accessible when they are designed to provide full and successful experiences for as many people as possible.

Build accessible and inclusive Windows applications, with improved functionality and usability, for people with disabilities (both temporary and permanent), personal preferences, specific work styles, or situational constraints (such as shared work spaces, driving, cooking, glare, and so on). Some common solutions include providing information in alternative formats (such as captions on a video) or enabling the use of assistive technologies (such as screen readers).

Everyone should have access to the same rooms in a building, whether they need to use the stairs or the elevator.

This page provides information on how the various Windows development frameworks provide accessibility support for developers building Windows applications, assistive technology developers building tools such as screen readers and magnifiers, and software test engineers creating automated scripts for testing applications.

Platform-specific documentation

Universal Windows Platform (UWP)

Develop accessible apps and tools on the modern platform for Windows applications and games on any Windows device (including PCs, phones, Xbox, HoloLens, and more), and publish them to the Microsoft Store.

Designing inclusive software

Developing inclusive Windows apps

Accessibility testing

Accessibility in the Store

Win32 platform

Develop accessible apps and tools on the original platform for C/C++ Windows applications.

What's new in Windows accessibility and automation

Developing accessible applications for Windows

Developing accessible UI frameworks for Windows

Developing assistive technology for Windows

Testing for accessibility

Legacy accessibility and automation technology - MSAA to UI Automation

Windows Accessibility features

Guidelines for designing accessible apps

Windows Presentation Foundation (WPF)

Develop accessible apps and tools on the established platform for managed Windows applications with a XAML UI model and the .NET Framework.

Accessibility Best Practices

UI Automation Fundamentals

UI Automation Providers for Managed Code

UI Automation Clients for Managed Code

UI Automation Control Patterns

UI Automation Text Pattern

UI Automation Control Types

UI Automation Specification and Community Promise

Windows Forms (WinForms)

Develop accessible apps and tools for managed Windows applications with a XAML UI model and the .NET Framework.

Windows Forms Accessibility

Creating an Accessible Windows Application

Properties on Windows Forms Controls That Support Accessibility Guidelines

Providing Accessibility Information for Controls on a Windows Form

Web accessibility

Design, build, and test accessible web sites in Microsoft Edge.

Introduction to web accessibility

Designing accessible websites

Building accessible websites

Testing accessible websites

Samples

Download and run full Windows samples that demonstrate various accessibility features and functionality.

Code sample browser

The new samples browser replacing the MSDN Code Gallery.

MSDN Code Gallery (GitHub archive)

Download samples for Windows, Windows Phone, Microsoft Azure, Office, SharePoint, Silverlight and other products.

Windows classic samples on GitHub

These samples demonstrate the functionality and programming model for Windows and Windows Server.

Universal Windows Platform (UWP) samples on GitHub

These samples demonstrate the API usage patterns for the Universal Windows Platform (UWP) in the Windows Software Development Kit (SDK) for Windows 10 and later.

WinUI 2 Gallery

This app demonstrates the various Xaml controls supported in the Fluent Design System.

Videos

Various videos covering how to build accessible Windows applications to general accessibility concerns and how Microsoft addresses them.

Microsoft's Accessibility API

Introduction to disability and accessibility

https://www.youtube-nocookie.com/embed/6b0K2883rXA

https://www.youtube-nocookie.com/embed/Kl4CT4DaypM

One Dev Minute: Developing apps for accessibility

Windows 11 accessibility features empower everyone

Making the mouse pointers easier to see

https://learn-video.azurefd.net/vod/player?id=001a0bd5-c137-4a17-b45f-9f4368bf379d&locale=en-us&embedUrl=%2Fwindows%2Fapps%2Fdevelop%2Faccessibility
https://www.youtube-nocookie.com/embed/uDzVBNry31s
https://www.youtube-nocookie.com/embed/BoME-Wr_jNw

Other resources

Blogs and news

The latest from the world of Microsoft accessibility.

In the news

Accessibility blogs

Windows UI Automation blogs

Community and support

A place where Windows developers and users meet and learn together.

Windows community - Accessibility

Windows Accessibility and Automation Development Forum

Stack Overflow

Disability Answer Desk

AI for Windows apps

Learn how to take advantage of state-of-the-art local models, NPU hardware, and remote APIs to make your apps smarter than ever before.

Introducing Windows AI Foundry

GET STARTED

Get started with AI on Windows

AI on Windows Sample Gallery

FAQs about using AI on Windows

Responsible Generative AI Development on Windows

Our principles for using AI-generated content on Microsoft Learn

Use Windows AI APIs

CONCEPT

Learn about on-device APIs

Get started with Windows AI APIs

Phi Silica API - Small Language Model with reasoning and language understanding

Text Recognition API (Optical Character Recognition / OCR)

Imaging API - Image Super Resolution to scale and sharpen, Image Segmentation to identify objects.

Recall API - Retrieval of past activities for user to continue where they left off

Click to Do - Uses the local Phi Silica model to surface actions on content (text or images)

Studio Effects - NPU-accelerated camera and microphone AI effects

Use customized local AI models

HOW-TO GUIDE

Learn about Windows ML

Run ONNX models with Windows ML

Copilot+ PCs Developer Guide

Web Neural Network API (WebNN) Overview

Get started with WebNN with Windows

AI Toolkit for Visual Studio Code overview

Use app actions in Windows apps

HOW-TO GUIDE

Learn about app actions on Windows

Get started building apps with app actions

AI Tools on Windows

OVERVIEW

Try and test samples with the AI Dev Gallery App

Supercharge development with AI Toolkit for Visual Studio Code

Introducing Microsoft.Extensions.AI Preview

Introduction to Semantic Kernel

OpenAI basic samples

Additional AI resources

CONCEPT

Microsoft AI learning hub

Microsoft Copilot learning hub

Microsoft Responsible AI

Microsoft AI Show Live

App lifecycle, background tasks, and system services

Article • 03/19/2025

This article provides an index of development features that are related to scenarios involving managing the lifecycle of Windows apps and using system-level services provided by the Windows OS.

Windows App SDK features

The Windows App SDK provides the following features related to app lifecycle and system services for Windows 10 and later OS releases.

> **Important**
>
> If you're working on a UWP app, then refer to Migrate from UWP to the Windows App SDK.

Expand table

Feature	Description
App lifecycle	Get an overview of managing the lifecycle of your app.
App instancing	Control whether multiple instances of your app's process can run at the same time.
Background tasks	Background tasks are app components that run in the background without a user interface. They can perform actions such as downloading files, syncing data, sending notifications, or updating tiles.
Rich activation	Receive information about different kinds activations for your app.
Power management	Get visibility into how your app affects the device's power state, and enable your app to make intelligent decisions about resource usage.
Restart	Programmatically restart your application and set restart options after app termination.

Windows OS features

Windows 10 and later OS releases provide a wide variety of APIs related to app lifecycle and system services for apps. These features are available via a combination of WinRT and Win32 (C++ and COM) APIs provided by the Windows SDK.

WinRT APIs

The following articles provide information about features available via WinRT APIs provided by the Windows SDK.

Expand table

Article	Description
Use app services and extensions	Learn how to integrate app services into your Windows app to allow the sharing of data and functionality across apps.
Support your app with background tasks	Learn how make lightweight code run in the background in response to triggers.
Launch an app through file activation	Learn how to set up your app to launch when a file of a certain type is opened.
Launch an app with a URI	Learn how to use a Uniform Resource Identifier (URI) to launch one app from another app, enabling helpful app-to-app scenarios.
Threading and async programming	Learn how to use the thread pool to accomplish work asynchronously in parallel threads.

Win32 (C++ and COM) APIs

The following articles provide information about features available via Win32 (C++ and COM) APIs provided by the Windows SDK.

Expand table

Article	Description
System services	Learn about system services available to apps, including COM, Windows messaging, service applications, and much more.
Memory management	Learn how to use virtual memory, memory mapped files, copy-on-write memory, large memory support, and other memory related features in your app.
Processes and threads	Learn how to control processes, threads, jobs, and other units of code execution in your app.

Article	Description
Windows system information	Learn how to access system information including the registry, handles and objects, and more.

.NET features

The .NET SDK also provides APIs related to system services for WPF and Windows Forms apps.

Expand table

Article	Description
Threading model (WPF)	Learn about the threading model of WPF apps.
System information	Learn how to access system information in Windows Forms apps.

Windows App SDK app lifecycle

Article • 02/01/2023

This article provides an overview of managing the lifecycle of **Windows App SDK** desktop apps.

App lifecycle overview

The application lifecycle of a Windows App SDK app is not that same as a UWP app. The lifecycle of Windows App SDK apps is similar to other .NET and Win32 desktop apps. Windows App SDK apps, like UWP apps, are started and stopped. They are either running or not running. However, unlike UWP apps, they cannot be suspended and resumed. At the window level, your app can subscribe to events to react when windows are activated and deactivated.

Microsoft.UI.Xaml.Application lifecycle

The Application object is the main entry point for a Windows App SDK app. It's similar to the UWP Application class, but with some important differences. The `Application` object is created by the Windows App SDK framework and is accessible from the `Microsoft.UI.Xaml.Application.Current` property.

The `Application` class in Windows App SDK has only one lifecycle method, OnLaunched, which is called when the app is launched. The `OnLaunched` method is responsible for creating the app's main window and displaying it. The `OnLaunched` method is also responsible for initializing the Windows App SDK framework and starting the app. When you create a new Windows App SDK app, the `OnLaunched` method is automatically generated for you.

In contrast, the UWP `Application` class has several activation-related lifecycle methods, including OnLaunched, OnActivated, and OnBackgroundActivated. The `OnActivated` and `OnBackgroundActivated` methods are called when the app is activated. The `OnActivated` method is called when the app is activated by the user, and the `OnBackgroundActivated` method is called when the app is activated by the system.

UWP's `Application` class also has several lifecycle events: Suspending, Resuming, EnteredBackground, and LeavingBackground. The `Suspending` event is raised when the app is suspended, and the `Resuming` event is raised when the app is resumed. The `EnteredBackground` event is raised when the app enters the background, and the

`LeavingBackground` event is raised when the app leaves the background. For a full explanation of UWP lifecycle events, see Windows 10 UWP app lifecycle.

If you are migrating a UWP app to Windows App SDK, you can use the Application lifecycle functionality migration guide to understand the differences between the UWP and Windows App SDK app lifecycles.

Microsoft.UI.Xaml.Window lifecycle

The Window object in Windows App SDK has some lifecycle events as well, `Window.Activated` and `Window.Closed`.

Window.Activated

The Activated event is raised when the window has been activated or deactivated by the system. Apps can determine what the status of the Window activation is by checking the WindowActivationState property of the WindowActivatedEventArgs parameter. This event will fire any time the window is activated or deactivated, including when the window is minimized or maximized.

Window.Closed

The Closed event is raised when the window closes. If this is the last window to be closed, usually the app's MainWindow, the application will be terminated. Because there is no `Suspending` event raised by the `Application` object in Windows App SDK, you should use your main window's `Closed` event to save application state and clean up any managed resources.

See also

App lifecycle and system services

UWP app lifecycle

App instancing with the app lifecycle API

Article • 09/20/2024

An app's instancing model determines whether multiple instances of your app's process can run at the same time. The app lifecycle API in the Windows App SDK provides a way to control how many instances of your app can run at the same time, and to redirect activations to other instances when necessary.

This article describes how to use the app lifecycle API to control app instancing in your WinUI apps.

Prerequisites

To use the app lifecycle API in WinUI 3 apps:

- Download and install the latest release of the Windows App SDK. For more information, see Get started with WinUI.
- Follow the instructions to Create your first WinUI 3 project or to use the Windows App SDK in an existing project.

Single-instance apps

Apps are single-instanced if there can be only one main process running at a time. Attempting to launch a second instance of a single-instanced app typically results in the first instance's main window being activated instead. Note that this only applies to the main process. Single-instanced apps can create multiple background processes and still be considered single instanced.

WinUI apps are multi-instanced by default but have the ability to become single-instanced by deciding at launch-time whether to create a new instance or activate an existing instance instead.

The Microsoft Photos app is a good example of a single instanced WinUI app. When you launch Photos for the first time, a new window will be created. If you attempt to launch Photos again, the existing window will be activated instead.

For an example of how to implement single instancing in a WinUI 3 app with C#, see Create a single-instanced WinUI app.

Multi-instanced apps

Apps are multi-instanced if the main process can be run multiple times simultaneously. Attempting to launch a second instance of a multi-instanced app creates a new process and main window.

Traditionally, unpackaged apps are multi-instanced by default, but can implement single-instancing when necessarily. Typically this is done using a single named mutex to indicate if an app is already running.

Notepad is a good example of a multi instanced app. Each time you attempt to launch Notepad, a new instance of Notepad will be created regardless of how many instances are already running.

How the Windows App SDK instancing differs from UWP instancing

Instancing behavior in the Windows App SDK is based on UWP's model, class, but with some key differences:

AppInstance class

- **UWP**: The Windows.ApplicationModel.AppInstance class is focused purely on instance redirection scenarios.
- **Windows App SDK**: The Microsoft.Windows.AppLifeycle.AppInstance class supports instance redirection scenarios, and contains additional functionality to support new features in later releases.

List of instances

- **UWP**: GetInstances returns only the instances that the app explicitly registered for potential redirection.
- **Windows App SDK**: GetInstances returns all running instances of the app that are using the AppInstance API, whether or not they have registered a key. This can include the current instance. If you want the current instance to be included in the list, call `AppInstance.GetCurrent`. Separate lists are maintained for different versions of the same app, as well as instances of apps launched by different users.

Registering keys

Each instance of a multi-instanced app can register an arbitrary key via the `FindOrRegisterForKey` method. Keys have no inherent meaning; apps can use keys in whatever form or way they wish.

An instance of an app can set its key at any time, but only one key is allowed for each instance; setting a new value overwrites the previous value.

An instance of an app cannot set its key to the same value that another instance has already registered. Attempting to register an existing key will result in `FindOrRegisterForKey` returning the app instance that has already registered that key.

- **UWP**: An instance must register a key in order to be included in the list returned from GetInstances.
- **Windows App SDK**: Registering a key is decoupled from the list of instances. An instance does not need to register a key in order to be included in the list.

Unregistering keys

An instance of an app can unregister its key.

- **UWP**: When an instance unregisters its key, it is no longer available for activation redirection and is not included in the list of instances returned from GetInstances.
- **Windows App SDK**: An instance that has unregistered its key is still available for activation redirection and is still included in the list of instances returned from GetInstances.

Instance redirection targets

Multiple instances of an app can activate each other, a process called "activation redirection". For example, an app might implement single instancing by only initializing itself if no other instances of the app are found at startup, and instead redirect and exit if another instance exists. Multi-instanced apps can redirect activations when appropriate according to that app's business logic. When an activation is redirected to another instance, it uses that instance's `Activated` callback, the same callback that's used in all other activation scenarios.

- **UWP**: Only instances that have registered a key can be a target for redirection.
- **Windows App SDK**: Any instance can be a redirection target, whether or not it has a registered key.

Post-redirection behavior

- **UWP**: Redirection is a terminal operation; the app is terminated after redirecting the activation, even if the redirect failed.
- **Windows App SDK**: In the Windows App SDK, redirection is not a terminal operation. This in part reflects the potential problems in arbitrarily terminating a Win32 app that may have already allocated some memory, but also allows support of more sophisticated redirection scenarios. Consider a multi-instanced app where an instance receives an activation request while performing a large amount of CPU-intensive work. That app can redirect the activation request to another instance and continue its processing. That scenario would not be possible if the app was terminated after redirection.

An activation request can be redirected multiple times. Instance A could redirect to instance B, which could in turn redirect to instance C. Windows App SDK apps taking advantage of this functionality must guard against circular redirection - if C redirects to A in the example above, there is a potential infinite activation loop. It is up to the app to determine how to handle circular redirection depending on what makes sense for the workflows that app supports.

Activation events

In order to handle reactivation, the app can register for an Activated event.

- **UWP**: The event passes an IActivatedEventArgs to the app.
- **Windows App SDK**: The event passes a Microsoft.Windows.AppLifecycle.AppActivationArguments instance to the app, which contains one of the `-ActivatedEventArgs` instances.

Examples

Handling activations

This example demonstrates how an app registers for and handles an `Activated` event. When it receives an `Activated` event, this app uses the event arguments to determine what sort of action caused the activation, and responds appropriately.

C++

```
int APIENTRY wWinMain(
    _In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPWSTR lpCmdLine, _In_ int nCmdShow)
{
```

```
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // Initialize the Windows App SDK framework package for unpackaged apps.
    HRESULT hr{ MddBootstrapInitialize(majorMinorVersion, versionTag,
minVersion) };
    if (FAILED(hr))
    {
        OutputFormattedDebugString(
            L"Error 0x%X in MddBootstrapInitialize(0x%08X, %s,
%hu.%hu.%hu.%hu)\n",
            hr, majorMinorVersion, versionTag,
            minVersion.Major, minVersion.Minor, minVersion.Build,
minVersion.Revision);
        return hr;
    }

    if (DecideRedirection())
    {
        return 1;
    }

    // Connect the Activated event, to allow for this instance of the app
    // getting reactivated as a result of multi-instance redirection.
    AppInstance thisInstance = AppInstance::GetCurrent();
    auto activationToken = thisInstance.Activated(
        auto_revoke, [&thisInstance](
            const auto& sender, const AppActivationArguments& args)
        { OnActivated(sender, args); }
    );

    // Carry on with regular Windows initialization.
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_CLASSNAME, szWindowClass, MAX_LOADSTRING);
    RegisterWindowClass(hInstance);
    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

    MSG msg;
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    MddBootstrapShutdown();
    return (int)msg.wParam;
}

void OnActivated(const IInspectable&, const AppActivationArguments& args)
{
    int const arraysize = 4096;
    WCHAR szTmp[arraysize];
```

```
    size_t cbTmp = arraysize * sizeof(WCHAR);
    StringCbPrintf(szTmp, cbTmp, L"OnActivated (%d)", activationCount++);

    ExtendedActivationKind kind = args.Kind();
    if (kind == ExtendedActivationKind::Launch)
    {
        ReportLaunchArgs(szTmp, args);
    }
    else if (kind == ExtendedActivationKind::File)
    {
        ReportFileArgs(szTmp, args);
    }
}
```

Redirection logic based on activation kind

In this example, the app registers a handler for the Activated event, and also checks for the activation event args to decide whether to redirect activation to another instance.

For most types of activations, the app continues with its regular initialization process. However, if the activation was caused by an associated file type being opened, and if another instance of this app already has the file opened, the current instance will redirect the activation to the existing instance and exit.

This app uses key registration to determine which files are open in which instances. When an instance opens a file, it registers a key that includes that filename. Other instances can then examine the registered keys and look for particular filenames, and register themselves as that file's instance if no other instance already has.

Note that, though key registration itself is part of the app lifecycle API in the Windows App SDK's, the contents of the key are specified only within the app itself. An app does not need to register a file name, or any other meaningful data. This app, however, has decided to track open files via keys based on its particular needs and supported workflows.

C++

```
bool DecideRedirection()
{
    // Get the current executable filesystem path, so we can
    // use it later in registering for activation kinds.
    GetModuleFileName(NULL, szExePath, MAX_PATH);
    wcscpy_s(szExePathAndIconIndex, szExePath);
    wcscat_s(szExePathAndIconIndex, L",1");

    // Find out what kind of activation this is.
    AppActivationArguments args =
AppInstance::GetCurrent().GetActivatedEventArgs();
```

```
    ExtendedActivationKind kind = args.Kind();
    if (kind == ExtendedActivationKind::Launch)
    {
        ReportLaunchArgs(L"WinMain", args);
    }
    else if (kind == ExtendedActivationKind::File)
    {
        ReportFileArgs(L"WinMain", args);

        try
        {
            // This is a file activation: here we'll get the file
information,
            // and register the file name as our instance key.
            IFileActivatedEventArgs fileArgs =
args.Data().as<IFileActivatedEventArgs>();
            if (fileArgs != NULL)
            {
                IStorageItem file = fileArgs.Files().GetAt(0);
                AppInstance keyInstance =
AppInstance::FindOrRegisterForKey(file.Name());
                OutputFormattedMessage(
                    L"Registered key = %ls", keyInstance.Key().c_str());

                // If we successfully registered the file name, we must be
the
                // only instance running that was activated for this file.
                if (keyInstance.IsCurrent())
                {
                    // Report successful file name key registration.
                    OutputFormattedMessage(
                        L"IsCurrent=true; registered this instance for %ls",
                        file.Name().c_str());
                }
                else
                {
                    keyInstance.RedirectActivationToAsync(args).get();
                    return true;
                }
            }
        }
        catch (...)
        {
            OutputErrorString(L"Error getting instance information");
        }
    }
    return false;
}
```

Arbitrary redirection

This example expands on the previous example by adding more sophisticated redirection rules. The app still performs the open file check from the previous example. However, where the previous example would always create a new instance if it did not redirect based on the open file check, this example adds the concept of a "reusable" instance. If a reusable instance is found, the current instance redirects to the reusable instance and exits. Otherwise, it registers itself as reusable and continues with its normal initialization.

Again, note that the concept of a "reusable" instance does not exist in the app lifecycle API; it is created and used only within the app itself.

C++

```
int APIENTRY wWinMain(
    _In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPWSTR lpCmdLine, _In_ int nCmdShow)
{
    // Initialize COM.
    winrt::init_apartment();

    AppActivationArguments activationArgs =
        AppInstance::GetCurrent().GetActivatedEventArgs();

    // Check for any specific activation kind we care about.
    ExtendedActivationKind kind = activationArgs.Kind;
    if (kind == ExtendedActivationKind::File)
    {
        // etc... as in previous scenario.
    }
    else
    {
        // For other activation kinds, we'll trawl all instances to see if
        // any are suitable for redirecting this request. First, get a list
        // of all running instances of this app.
        auto instances = AppInstance::GetInstances();

        // In the simple case, we'll redirect to any other instance.
        AppInstance instance = instances.GetAt(0);

        // If the app re-registers re-usable instances, we can filter for
these instead.
        // In this example, the app uses the string "REUSABLE" to indicate
to itself
        // that it can redirect to a particular instance.
        bool isFound = false;
        for (AppInstance instance : instances)
        {
            if (instance.Key == L"REUSABLE")
            {
                isFound = true;
                instance.RedirectActivationToAsync(activationArgs).get();
```

```
                break;
            }
        }
        if (!isFound)
        {
            // We'll register this as a reusable instance, and then
            // go ahead and do normal initialization.
            winrt::hstring szKey = L"REUSABLE";
            AppInstance::FindOrRegisterForKey(szKey);
            RegisterClassAndStartMessagePump(hInstance, nCmdShow);
        }
    }
    return 1;
}
```

Redirection orchestration

This example again adds more sophisticated redirection behavior. Here, an app instance can register itself as the instance that handles all activations of a specific kind. When an instance of an app receives a `Protocol` activation, it first checks for an instance that has already registered to handle `Protocol` activations. If it finds one, it redirects the activation to that instance. If not, the current instance registers itself for `Protocol` activations, and then applies additional logic (not shown) which may redirect the activation for some other reason.

C++

```
void OnActivated(const IInspectable&, const AppActivationArguments& args)
{
    const ExtendedActivationKind kind = args.Kind;

    // For example, we might want to redirect protocol activations.
    if (kind == ExtendedActivationKind::Protocol)
    {
        auto protocolArgs = args.Data().as<ProtocolActivatedEventArgs>();
        Uri uri = protocolArgs.Uri();

        // We'll try to find the instance that handles protocol activations.
        // If there isn't one, then this instance will take over that duty.
        auto instance =
AppInstance::FindOrRegisterForKey(uri.AbsoluteUri());
        if (!instance.IsCurrent)
        {
            instance.RedirectActivationToAsync(args).get();
        }
        else
        {
            DoSomethingWithProtocolArgs(uri);
        }
    }
```

```
    else
    {
        // In this example, this instance of the app handles all other
        // activation kinds.
        DoSomethingWithNewActivationArgs(args);
    }
}
```

Unlike the UWP version of `RedirectActivationTo`, the Windows App SDK's implementation of RedirectActivationToAsync requires explicitly passing event arguments when redirecting activations. This is necessary because whereas UWP tightly controls activations and can ensure the correct activation arguments are passed to the correct instances, the Windows App SDK's version supports many platforms, and cannot rely on UWP-specific features. One benefit of this model is that apps that use the Windows App SDK have the chance to modify or replace the arguments that will be passed to the target instance.

Redirection without blocking

Most apps will want to redirect as early as possible, before doing unnecessary initialization work. For some app types, initialization logic runs on an STA thread, which must not be blocked. AppInstance.RedirectActivationToAsync method is asynchronous, and the calling app must wait for the method to complete, otherwise the redirection will fail. However, waiting on an async call will block the STA. In these situations, call RedirectActivationToAsync in another thread, and set an event when the call completes. Then wait on that event using non-blocking APIs such as CoWaitForMultipleObjects. Here's a C# sample for a WPF app.

C#

```
private static bool DecideRedirection()
{
    bool isRedirect = false;

    // Find out what kind of activation this is.
    AppActivationArguments args =
AppInstance.GetCurrent().GetActivatedEventArgs();
    ExtendedActivationKind kind = args.Kind;
    if (kind == ExtendedActivationKind.File)
    {
        try
        {
            // This is a file activation: here we'll get the file
information,
            // and register the file name as our instance key.
            if (args.Data is IFileActivatedEventArgs fileArgs)
            {
```

```
                IStorageItem file = fileArgs.Files[0];
                AppInstance keyInstance =
AppInstance.FindOrRegisterForKey(file.Name);

                // If we successfully registered the file name, we must be
the
                // only instance running that was activated for this file.
                if (keyInstance.IsCurrent)
                {
                    // Hook up the Activated event, to allow for this
instance of the app
                    // getting reactivated as a result of multi-instance
redirection.
                    keyInstance.Activated += OnActivated;
                }
                else
                {
                    isRedirect = true;

                    // Ensure we don't block the STA, by doing the redirect
operation
                    // in another thread, and using an event to signal when
it has completed.
                    redirectEventHandle = CreateEvent(IntPtr.Zero, true,
false, null);
                    if (redirectEventHandle != IntPtr.Zero)
                    {
                        Task.Run(() =>
                        {

keyInstance.RedirectActivationToAsync(args).AsTask().Wait();
                            SetEvent(redirectEventHandle);
                        });
                        uint CWMO_DEFAULT = 0;
                        uint INFINITE = 0xFFFFFFFF;
                        _ = CoWaitForMultipleObjects(
                            CWMO_DEFAULT, INFINITE, 1,
                            new IntPtr[] { redirectEventHandle }, out uint
handleIndex);
                    }
                }
            }
        }
        catch (Exception ex)
        {
            Debug.WriteLine($"Error getting instance information:
{ex.Message}");
        }
    }

    return isRedirect;
}
```

Unregister for redirection

Apps that have registered a key can unregister that key at any time. This example assumes the current instance had previously registered a key indicating that it had a specific file opened, meaning subsequent attempts to open that file would be redirected to it. When that file is closed, the key that contains the filename must be deleted.

C++

```
void CALLBACK OnFileClosed(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam)
{
    AppInstance::GetCurrent().UnregisterKey();
}
```

> ⚠ **Warning**
>
> Although keys are automatically unregistered when their process terminates, race conditions are possible where another instance may have initiated a redirection to the terminated instance before the terminated instance was unregistered. To mitigate this possibility, an app can use UnregisterKey to manually unregister its key before it is terminated, giving the app a chance to redirect activations to another app that is not in the process of exiting.

Instance information

The Microsoft.Windows.AppLifeycle.AppInstance class represents a single instance of an app. In the current preview, `AppInstance` only includes the methods and properties necessary to support activation redirection. In later releases, `AppInstance` will expand to include other methods and properties relevant to an app instance.

C++

```
void DumpExistingInstances()
{
    for (AppInstance const& instance : AppInstance::GetInstances())
    {
        std::wostringstream sStream;
        sStream << L"Instance: ProcessId = " << instance.ProcessId
            << L", Key = " << instance.Key().c_str() << std::endl;
        ::OutputDebugString(sStream.str().c_str());
    }
}
```

Related content

Create a single-instanced WinUI app

Microsoft.Windows.AppLifeycle.AppInstance

Create a single-instanced WinUI app with C#

Article • 09/20/2024

This how-to demonstrates how to create a single-instanced WinUI 3 app with C# and the Windows App SDK. Single-instanced apps only allow one instance of the app running at a time. WinUI apps are multi-instanced by default. They allow you to launch multiple instances of the same app simultaneously. That's referred to a multiple instances. However, you may want to implement single-instancing based on the use case of your app. Attempting to launch a second instance of a single-instanced app will only result in the first instance's main window being activated instead. This tutorial demonstrates how to implement single-instancing in a WinUI app.

In this article, you will learn how to:

- ✓ Turn off XAML's generated `Program` code
- ✓ Define customized `Main` method for redirection
- ✓ Test single-instancing after app deployment

Pre-requisites

This tutorial uses Visual Studio and builds on the WinUI blank app template. If you're new to WinUI development, you can get set up by following the instructions in Get started with WinUI. There you'll install Visual Studio, configure it for developing apps with WinUI while ensuring you have the latest version of WinUI and the Windows App SDK, and create a Hello World project.

When you've done that, come back here to learn how to turn your "Hello World" project into a single-instanced app.

> ⓘ **Note**
>
> This how-to is based on the Making the app single-instanced (Part 3) blog post from a Windows blog series on WinUI 3. The code for those articles is available on GitHub.

Disable auto-generated Program code

We need to check for redirection as early as possible, before creating any windows. To do this, we must define the symbol “DISABLE_XAML_GENERATED_MAIN” in the project file. Follow these steps to disable the auto-generated Program code:

1. Right-click on the project name in Solution Explorer and select **Edit Project File**.
2. Define the **DISABLE_XAML_GENERATED_MAIN** symbol. Add the following XML to the project file:

```
XML

<PropertyGroup>

<DefineConstants>$(DefineConstants);DISABLE_XAML_GENERATED_MAIN</Define
Constants>
</PropertyGroup>
```

Adding the **DISABLE_XAML_GENERATED_MAIN** symbol will disable the auto-generated Program code for your project.

Define a Program class with a Main method

A customized Program.cs file must be created instead of running the default Main method. The code added to the **Program** class enables the app to check for redirection, which isn't the default behavior of WinUI apps.

1. Navigate to Solution Explorer, right-click on the project name, and select **Add | Class**.
2. Name the new class `Program.cs` and select **Add**.
3. Add the following namespaces to the Program class, replacing any existing namespaces:

```
C#

using System;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Threading;
using System.Threading.Tasks;
using Microsoft.UI.Dispatching;
using Microsoft.UI.Xaml;
using Microsoft.Windows.AppLifecycle;
```

4. Replace the empty **Program** class with the following:

C#

```
public class Program
{
    [STAThread]
    static int Main(string[] args)
    {
        WinRT.ComWrappersSupport.InitializeComWrappers();
        bool isRedirect = DecideRedirection();

        if (!isRedirect)
        {
            Application.Start((p) =>
            {
                var context = new
DispatcherQueueSynchronizationContext(
                    DispatcherQueue.GetForCurrentThread());

SynchronizationContext.SetSynchronizationContext(context);
                _ = new App();
            });
        }

        return 0;
    }
}
```

The **Main** method determines whether the app should redirect to the first instance or start a new instance after calling **DecideRedirection**, which we will define next.

5. Define the **DecideRedirection** method below the **Main** method:

C#

```
private static bool DecideRedirection()
{
    bool isRedirect = false;
    AppActivationArguments args =
AppInstance.GetCurrent().GetActivatedEventArgs();
    ExtendedActivationKind kind = args.Kind;
    AppInstance keyInstance =
AppInstance.FindOrRegisterForKey("MySingleInstanceApp");

    if (keyInstance.IsCurrent)
    {
        keyInstance.Activated += OnActivated;
    }
    else
    {
        isRedirect = true;
        RedirectActivationTo(args, keyInstance);
    }
```

```
    return isRedirect;
}
```

DecideRedirection determines if the app has been registered by registering a unique key that represents your app instance. Based on the result of key registration, it can determine if there's a current instance of the app running. After making the determination, the method knows whether to redirect or allow the app to continue launching the new instance. The **RedirectActivationTo** method is called if redirection is necessary.

6. Next, let's create the RedirectActivationTo method below the DecideRedirection method, along with the required DllImport statements. Add the following code to the Program class:

C#

```
[DllImport("kernel32.dll", CharSet = CharSet.Unicode)]
private static extern IntPtr CreateEvent(
    IntPtr lpEventAttributes, bool bManualReset,
    bool bInitialState, string lpName);

[DllImport("kernel32.dll")]
private static extern bool SetEvent(IntPtr hEvent);

[DllImport("ole32.dll")]
private static extern uint CoWaitForMultipleObjects(
    uint dwFlags, uint dwMilliseconds, ulong nHandles,
    IntPtr[] pHandles, out uint dwIndex);

[DllImport("user32.dll")]
static extern bool SetForegroundWindow(IntPtr hWnd);

private static IntPtr redirectEventHandle = IntPtr.Zero;

// Do the redirection on another thread, and use a non-blocking
// wait method to wait for the redirection to complete.
public static void RedirectActivationTo(AppActivationArguments args,
                                        AppInstance keyInstance)
{
    redirectEventHandle = CreateEvent(IntPtr.Zero, true, false, null);
    Task.Run(() =>
    {
        keyInstance.RedirectActivationToAsync(args).AsTask().Wait();
        SetEvent(redirectEventHandle);
    });

    uint CWMO_DEFAULT = 0;
    uint INFINITE = 0xFFFFFFFF;
    _ = CoWaitForMultipleObjects(
       CWMO_DEFAULT, INFINITE, 1,
```

```
        [redirectEventHandle], out uint handleIndex);

    // Bring the window to the foreground
    Process process =
Process.GetProcessById((int)keyInstance.ProcessId);
    SetForegroundWindow(process.MainWindowHandle);
}
```

The **RedirectActivationTo** method is responsible for redirecting the activation to the first instance of the app. It creates an event handle, starts a new thread to redirect the activation, and waits for the redirection to complete. After the redirection is complete, the method brings the window to the foreground.

7. Finally, define the helper method **OnActivated** below the **DecideRedirection** method:

```
C#

private static void OnActivated(object sender, AppActivationArguments
args)
{
    ExtendedActivationKind kind = args.Kind;
}
```

Test single-instancing via app deployment

Until this point, we've been testing the app by debugging within Visual Studio. However, we can only have one debugger running at once. This limitation prevents us from knowing whether the app is single-instanced because we can’t debug the same project twice at the same time. For an accurate test, we'll deploy the application to our local Windows client. After deploying, we can launch the app from the desktop like you would with any app installed on Windows.

1. Navigate to Solution Explorer, right-click on the project name, and select **Deploy**.
2. Open the Start menu and click into the search field.
3. Type your app's name in the search field.
4. Click the app icon from the search result to launch your app.

> ⓘ **Note**
>
> If you experience app crashes in release mode, there are some known issues with trimmed apps in the Windows App SDK. You can disable trimming in the

project by setting the **PublishTrimmed** property to **false** for all build configurations in your project's `.pubxml` files. For more information, see this issue on GitHub.

5. Repeat steps 2 to 4 to launch the same app again and see if another instance opens. If the app is single-instanced, the first instance will be activated instead of a new instance opening.

> **Tip**
>
> You can optionally add some logging code to the **OnActivated** method to verify that the existing instance has been activated. Try asking **Copilot** for help with adding an **ILogger implementation** to your WinUI app.

Summary

All the code covered here is on GitHub, with branches for the different steps in the original Windows blog series. See the single-instancing branch for code specific to this how-to. The `main` branch is the most comprehensive. The other branches are intended to show you how the app architecture evolved.

Related content

App instancing with the app lifecycle API

Making the app single-instanced (Part 3)

WinAppSDK-DrumPad sample on GitHub

Rich activation with the app lifecycle API

Article • 03/07/2024

In the Windows App SDK, the app lifecycle API brings support for UWP-style rich activation behavior to all apps, packaged and unpackaged alike. This first release focuses on bringing the most commonly-used activation kinds to unpackaged apps, and future releases aim to support more of UWP's 44 activation kinds.

Supporting rich activations requires two steps:

- Tell the system that your app supports one or more rich activation kinds.
- Receive and process the rich activation payloads your app receives when it is activated.

Prerequisites

To use the app lifecycle API in the Windows App SDK:

1. Download and install the latest release of the Windows App SDK. For more information, see Get started with WinUI.
2. Follow the instructions to Create your first WinUI 3 project or to use the Windows App SDK in an existing project.

Activation details for unpackaged apps

The current version of the Windows App SDK supports the four most common activation kinds to unpackaged apps. These activation kinds are defined by the ExtendedActivationKind enum.

Expand table

Activation kind	Description
`Launch`	Activate the app from the command line, when the user double-clicks the app's icon, or programmatically via ShellExecute or CreateProcess.
`File`	Activate an app that has registered for a file type when a file of the type is opened via ShellExecute, Launcher.LaunchFileAsync, or the command line.
`Protocol`	Activate an app that has registered for a protocol when a string of that protocol is executed via ShellExecute, Launcher.LaunchUriAsync, or the command-line.

Activation kind	Description
`StartupTask`	Activate the app when the user logs into Windows, either because of a registry key, or because of a shortcut in a well-known startup folder.

Each type of unpackaged app retrieves its command line arguments in different ways. For example, C++ Win32 apps expect to receive activation arguments to be passed into `WinMain` in the form of a string (though they also have the option to call GetCommandLineW). Windows Forms apps, however, *must* call Environment.GetCommandLineArgs, because arguments will not be automatically passed to them.

Activation details for packaged apps

Packaged apps that use the Windows App SDK support all 44 of UWP's activation kinds. Each activation kind has its own corresponding implementation of IActivatedEventArgs which contain properties relevant to that specific kind of activation.

Packaged apps will always receive activation event arguments in their AppInstance.Activated event handler, and also have the option of calling AppInstance.GetActivatedEventArgs.

Activation registration

All apps support the `Launch` activation kind by default. Unlike UWP, the Windows App SDK `Launch` activation kind includes command line launches. Apps can register for additional activation kinds in several ways.

- Unpackaged apps that use the Windows App SDK can register (and unregister) for additional activation kinds via the app lifecycle API in the Windows App SDK.
- Unpackaged apps can continue to register for additional activation kinds using the traditional method of writing registry keys.
- Packaged apps can register for additional activation kinds via entries in their application manifest.

Activation registrations are per-user. If your app is installed for multiple users, you will need to re-register activations for each user.

Examples

Register for rich activation

Although apps can call the registration APIs at any time, the most common scenario is checking registrations on app startup.

This example shows how an unpackaged app can use the following static methods of the ActivationRegistrationManager class to register for several activation kinds when the app is launched:

- RegisterForFileTypeActivation
- RegisterForProtocolActivation
- RegisterForStartupActivation

This example also demonstrates how to use the MddBootstrapInitialize and MddBootstrapShutdown functions to initialize and clean up references to the Windows App SDK framework package. All unpackaged app must do this to use APIs provided by the Windows App SDK. For more information, see Use the Windows App SDK runtime for apps packaged with external location or unpackaged.

> **Note**
>
> This example registers associations with three image file types at once. This is convenient, but the outcome is the same as registering each file type individually; registering new image types does not overwrite previous registrations. However, if an app re-registers an already registered file type with a different set of verbs, the previous set of verbs will be overwritten for that file type.

c++

```
const UINT32 majorMinorVersion{ WINDOWSAPPSDK_RELEASE_MAJORMINOR };
PCWSTR versionTag{ WINDOWSAPPSDK_RELEASE_VERSION_TAG_W };
const PACKAGE_VERSION minVersion{ WINDOWSAPPSDK_RUNTIME_VERSION_UINT64 };
WCHAR szExePath[MAX_PATH]{};
WCHAR szExePathAndIconIndex[MAX_PATH + 8]{};

int APIENTRY wWinMain(
    _In_ HINSTANCE hInstance, _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPWSTR lpCmdLine, _In_ int nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // Initialize Windows App SDK framework package for unpackaged apps.
    HRESULT hr{ MddBootstrapInitialize(majorMinorVersion, versionTag,
minVersion) };
    if (FAILED(hr))
```

```
    {
        wprintf(L"Error 0x%X in MddBootstrapInitialize(0x%08X, %s,
%hu.%hu.%hu.%hu)\n",
            hr, majorMinorVersion, versionTag, minVersion.Major,
            minVersion.Minor, minVersion.Build, minVersion.Revision);
        return hr;
    }

    // Get the current executable filesystem path, so we can
    // use it later in registering for activation kinds.
    GetModuleFileName(NULL, szExePath, MAX_PATH);
    wcscpy_s(szExePathAndIconIndex, szExePath);
    wcscat_s(szExePathAndIconIndex, L",1");

    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_CLASSNAME, szWindowClass, MAX_LOADSTRING);
    RegisterWindowClass(hInstance);
    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

    MSG msg;
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    // Uninitialize Windows App SDK.
    MddBootstrapShutdown();
    return (int)msg.wParam;
}

void RegisterForActivation()
{
    OutputMessage(L"Registering for rich activation");

    // Register one or more supported filetypes, specifying
    // an icon (specified by binary file path plus resource index),
    // a display name to use in Shell and Settings,
    // zero or more verbs for the File Explorer context menu,
    // and the path to the EXE to register for activation.
    hstring myFileTypes[3] = { L".foo", L".foo2", L".foo3" };
    hstring verbs[2] = { L"view", L"edit" };
    ActivationRegistrationManager::RegisterForFileTypeActivation(
        myFileTypes,
        szExePathAndIconIndex,
        L"Contoso File Types",
        verbs,
        szExePath
    );

    // Register a URI scheme for protocol activation,
    // specifying the scheme name, icon, display name and EXE path.
```

```
    ActivationRegistrationManager::RegisterForProtocolActivation(
        L"foo",
        szExePathAndIconIndex,
        L"Contoso Foo Protocol",
        szExePath
    );

    // Register for startup activation.
    // As we're registering for startup activation multiple times,
    // and this is a multi-instance app, we'll get multiple instances
    // activated at startup.
    ActivationRegistrationManager::RegisterForStartupActivation(
        L"ContosoStartupId",
        szExePath
    );

    // If we don't specify the EXE, it will default to this EXE.
    ActivationRegistrationManager::RegisterForStartupActivation(
        L"ContosoStartupId2",
        L""
    );
}
```

Get rich activation event arguments

Once activated, an app must retrieve its activation event arguments. In this example, an unpackaged app calls the AppInstance.GetActivatedEventArgs method to get the event args for the activation event and then uses the AppActivationArguments.Kind property to retrieve the event args for different types of activations.

> ⓘ **Note**
>
> Win32 apps typically get command-line arguments very early their `WinMain` method. Similarly, these apps should call AppInstance.GetActivatedEventArgs in the same place where they previously would have used the supplied the `lpCmdLine` parameter or called `GetCommandLineW`.

```
c++

void GetActivationInfo()
{
    AppActivationArguments args =
AppInstance::GetCurrent().GetActivatedEventArgs();
    ExtendedActivationKind kind = args.Kind();
    if (kind == ExtendedActivationKind::Launch)
    {
        ILaunchActivatedEventArgs launchArgs =
```

```
            args.Data().as<ILaunchActivatedEventArgs>();
        if (launchArgs != NULL)
        {
            winrt::hstring argString = launchArgs.Arguments().c_str();
            std::vector<std::wstring> argStrings = split_strings(argString);
            OutputMessage(L"Launch activation");
            for (std::wstring s : argStrings)
            {
                OutputMessage(s.c_str());
            }
        }
    }
    else if (kind == ExtendedActivationKind::File)
    {
        IFileActivatedEventArgs fileArgs =
            args.Data().as<IFileActivatedEventArgs>();
        if (fileArgs != NULL)
        {
            IStorageItem file = fileArgs.Files().GetAt(0);
            OutputFormattedMessage(
                L"File activation: %s", file.Name().c_str());
        }
    }
    else if (kind == ExtendedActivationKind::Protocol)
    {
        IProtocolActivatedEventArgs protocolArgs =
            args.Data().as<IProtocolActivatedEventArgs>();
        if (protocolArgs != NULL)
        {
            Uri uri = protocolArgs.Uri();
            OutputFormattedMessage(
                L"Protocol activation: %s", uri.RawUri().c_str());
        }
    }
    else if (kind == ExtendedActivationKind::StartupTask)
    {
        IStartupTaskActivatedEventArgs startupArgs =
            args.Data().as<IStartupTaskActivatedEventArgs>();
        if (startupArgs != NULL)
        {
            OutputFormattedMessage(
                L"Startup activation: %s", startupArgs.TaskId().c_str());
        }
    }
}
```

Unregister

This example demonstrates how an unpackaged app can unregister for specific activation kinds dynamically, using the following static methods of the ActivationRegistrationManager class:

- UnregisterForFileTypeActivation
- UnregisterForProtocolActivation
- UnregisterForStartupActivation

> **Note**
>
> When unregistering for startup activation, the app must use the same taskId that it used when it originally registered.

c++

```
void UnregisterForActivation()
{
    OutputMessage(L"Unregistering for rich activation");

    // Unregister one or more registered filetypes.
    try
    {
        hstring myFileTypes[3] = { L".foo", L".foo2", L".foo3" };
        ActivationRegistrationManager::UnregisterForFileTypeActivation(
            myFileTypes,
            szExePath
        );
    }
    catch (...)
    {
        OutputMessage(L"Error unregistering file types");
    }

    // Unregister a protocol scheme.
    ActivationRegistrationManager::UnregisterForProtocolActivation(
        L"foo",
        L"");

    // Unregister for startup activation.
    ActivationRegistrationManager::UnregisterForStartupActivation(
        L"ContosoStartupId");
    ActivationRegistrationManager::UnregisterForStartupActivation(
        L"ContosoStartupId2");
}
```


Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

issues and pull requests. For more information, see our contributor guide.

Open a documentation issue

Provide product feedback

Power management with the app lifecycle API

Article • 03/07/2024

The app lifecycle API in the Windows App SDK provides a set of power management APIs in the Microsoft.Windows.System.Power namespace. These APIs provide visibility into how an app affects the device's power state, and they enable the app to make intelligent decisions about resource usage. For example, an app might use this API to postpone resource-intensive background tasks while the device is running on battery power.

The power management APIs use a callback-based model similar to the existing PowerSettingRegisterNotification function. Using a callback model extends the reach of the API to all apps, including background apps, headless apps, and others.

Prerequisites

To use the app lifecycle API in the Windows App SDK:

1. Download and install the latest release of the Windows App SDK. For more information, see Get started with WinUI.
2. Follow the instructions to Create your first WinUI 3 project or to use the Windows App SDK in an existing project.

Subscribe and respond to events

The following example demonstrates how to subscribe and respond to PowerManager events. This code subscribes to the BatteryStatusChanged event during startup. The app then responds to changes by checking the current power level and adjusting its resource usage appropriately. For example, if the battery discharges at a low power state, the app might defer any non-critical background work.

C++/WinRT

> **Note**

Apps can register and unregister for these events at any time, but most apps will want to set callbacks in `WinMain` that persist as long as the app continues to run.

C++

```
BOOL bWorkInProgress;
winrt::event_token batteryToken;
winrt::event_token powerToken;
winrt::event_token powerSourceToken;
winrt::event_token chargeToken;
winrt::event_token dischargeToken;

void RegisterPowerManagerCallbacks()
{
    batteryToken = PowerManager::BatteryStatusChanged([&](
        const auto&, winrt::Windows::Foundation::IInspectable obj) {
OnBatteryStatusChanged(); });
    powerToken = PowerManager::PowerSupplyStatusChanged([&](
        const auto&, winrt::Windows::Foundation::IInspectable obj) {
OnPowerSupplyStatusChanged(); });
    powerSourceToken = PowerManager::PowerSourceKindChanged([&](
        const auto&, winrt::Windows::Foundation::IInspectable obj) {
OnPowerSourceKindChanged(); });
    chargeToken = PowerManager::RemainingChargePercentChanged([&](
        const auto&, winrt::Windows::Foundation::IInspectable obj) {
OnRemainingChargePercentChanged(); });
    dischargeToken = PowerManager::RemainingDischargeTimeChanged([&](
        const auto&, winrt::Windows::Foundation::IInspectable obj) {
OnRemainingDischargeTimeChanged(); });

    if (batteryToken && powerToken && powerSourceToken && chargeToken &&
dischargeToken)
    {
        OutputMessage(L"Successfully registered for state
notifications");
    }
    else
    {
        OutputMessage(L"Failed to register for state notifications");
    }
}

void OnBatteryStatusChanged()
{
    const size_t statusSize = 16;
    WCHAR szStatus[statusSize];
    wmemset(&(szStatus[0]), 0, statusSize);

    BatteryStatus batteryStatus = PowerManager::BatteryStatus();
    int remainingCharge = PowerManager::RemainingChargePercent();
    switch (batteryStatus)
```

```
    {
    case BatteryStatus::Charging:
        wcscpy_s(szStatus, L"Charging");
        break;
    case BatteryStatus::Discharging:
        wcscpy_s(szStatus, L"Discharging");
        break;
    case BatteryStatus::Idle:
        wcscpy_s(szStatus, L"Idle");
        break;
    case BatteryStatus::NotPresent:
        wcscpy_s(szStatus, L"NotPresent");
        break;
    }

    OutputFormattedMessage(
        L"Battery status changed: %s, %d%% remaining",
        szStatus, remainingCharge);
    DetermineWorkloads();
}

void OnPowerSupplyStatusChanged()
{
//...etc
}
```

Configure app logic based on multiple status values

PowerManager events are relatively low-level, and in some scenarios, a single event handler being called might not provide enough information for the app to decide how to behave. In this example, the PowerSupplyStatusChanged event could be called when the device is disconnected from power. In that case, the app must check the current battery status before deciding how to proceed.

C++/WinRT

C++

```
void DetermineWorkloads()
{
    BatteryStatus batteryStatus = PowerManager::BatteryStatus();
    int remainingCharge = PowerManager::RemainingChargePercent();
    PowerSupplyStatus powerStatus = PowerManager::PowerSupplyStatus();
    PowerSourceKind powerSource = PowerManager::PowerSourceKind();

    if ((powerSource == PowerSourceKind::DC
```

```
        && batteryStatus == BatteryStatus::Discharging
        && remainingCharge < 25)
        || (powerSource == PowerSourceKind::AC
        && powerStatus == PowerSupplyStatus::Inadequate))
    {
        // The device is not in a good battery/power state,
        // so we should pause any non-critical work.
        PauseNonCriticalWork();
    }
    else if ((batteryStatus != BatteryStatus::Discharging &&
remainingCharge > 75)
        && powerStatus != PowerSupplyStatus::Inadequate)
    {
        // The device is in good battery/power state,
        // so let's kick of some high-power work.
        StartPowerIntensiveWork();
    }
}
```

Check screen status

The PowerManager class offers information about other device states relevant to an app's power usage. For example, apps can disable graphics processing when the device's display is turned off.

C++/WinRT

C++

```
void OnDisplayStatusChanged()
{
    const size_t statusSize = 16;
    WCHAR szStatus[statusSize];
    wmemset(&(szStatus[0]), 0, statusSize);

    DisplayStatus displayStatus = PowerManager::DisplayStatus();
    switch (displayStatus)
    {
    case DisplayStatus::Dimmed:
        wcscpy_s(szStatus, L"Dimmed");
        break;
    case DisplayStatus::Off:
        wcscpy_s(szStatus, L"Off");
        break;
    case DisplayStatus::On:
        wcscpy_s(szStatus, L"On");
        break;
    }
```

```
    OutputFormattedMessage(
        L"Display status changed: %s", szStatus);
    if (displayStatus == DisplayStatus::Off)
    {
        // The screen is off, let's stop rendering foreground graphics,
        // and instead kick off some background work now.
        StopUpdatingGraphics();
        StartDoingBackgroundWork();
    }
}
```

Unsubscribe from events

Apps can register and deregister for notifications during their lifecycle. Use your language's preferred event registration management system if your app doesn't need to receive power status notifications during its entire lifecycle.

C++/WinRT

C++

```
void UnregisterPowerManagerCallbacks()
{
    OutputMessage(L"Unregistering state notifications");
    PowerManager::BatteryStatusChanged(batteryToken);
    PowerManager::PowerSupplyStatusChanged(powerToken);
    PowerManager::PowerSourceKindChanged(powerSourceToken);
    PowerManager::RemainingChargePercentChanged(chargeToken);
    PowerManager::RemainingDischargeTimeChanged(dischargeToken);
}
```

Related topics

- Windows.System.Power.PowerManager
- PowerSettingRegisterNotification
- PowerSettingUnregisterNotification

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you

Windows developer feedback

Windows developer is an open source project. Select a link to

can also create and review issues and pull requests. For more information, see our contributor guide.

provide feedback:

- Open a documentation issue
- Provide product feedback

Detect and react to focus session state

Article • 07/27/2022

Windows 11 introduced the Focus feature which helps users minimize distractions by turning on Do Not Disturb and silencing icon flashing and badge notifications for apps in the taskbar. This article shows you how to use the FocusSessionManager API to detect whether a Focus session is currently active or receive updates when the Focus session state changes, allowing you to customize your app's behavior to minimize distractions when a Focus session is active. For more information on the Focus feature, see How to use focus in Windows 11.

Get the current Focus session state

Before accessing the properties of the **FocusSessionManager**, make sure it's supported on the current device by checking the IsSupported property. After verifying that the feature is supported you can determine if a Focus session is currently active by checking the IsFocusActive property.

C#

```
private void UpdateStatusBar(string message)
{
    var focusActive = false;
    if (Windows.UI.Shell.FocusSessionManager.IsSupported)
    {
        var manager = Windows.UI.Shell.FocusSessionManager.GetDefault();
        focusActive = manager.IsFocusActive;
    }

    if (!focusActive)
    {
        statusTextBlock.Text = message;
    }
}
```

Continuously monitor focus state

You can subscribe to be notified when the Focus session state on the device changes by registering for the IsFocusActiveChanged event. In the following example **SetAnimatedGifAutoPlay** is an app-implemented helper method that changes whether the app auto-plays animated gifs based on the current Focus session state.

C#

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    if (Windows.UI.Shell.FocusSessionManager.IsSupported)
    {
        var manager = Windows.UI.Shell.FocusSessionManager.GetDefault();
        manager.IsFocusActiveChanged += Manager_IsFocusActiveChanged;
        SetAnimatedGifAutoPlay(true);
    }
}

private void
Manager_IsFocusActiveChanged(Windows.UI.Shell.FocusSessionManager
sender, object args)
{
    if(sender.IsFocusActive)
    {
        SetAnimatedGifAutoPlay(true);
    }
    else
    {
        SetAnimatedGifAutoPlay(false);
    }
}
```

Launching Windows apps and managing background tasks

Article • 03/19/2025

Learn how to use a Uniform Resource Identifier (URI) or a file type to launch a Windows app or manage background tasks in your apps.

Launch a Windows app with a URI

This section describes how to use a URI to launch your Windows app.

Expand table

Topic	Description
Launch the default app for a URI	Learn how to launch the default app for a Uniform Resource Identifier (URI). URIs allow you to launch another app to perform a specific task. This topic also provides an overview of the many URI schemes built into Windows.
Handle URI activation	Learn how to register an app to become the default handler for a Uniform Resource Identifier (URI) scheme name.
Launch an app for results	Learn how to launch an app from another app and exchange data between the two. This is called launching an app for results.
Launch the Windows Settings app	Learn how to launch the Windows Settings app from your app. This topic describes the ms-settings URI scheme. Use this URI scheme to launch the Windows Settings app to specific settings pages.
Launch the Microsoft Store app	This topic describes the ms-windows-store URI scheme. Your app can use this URI scheme to launch the Microsoft Store app to specific pages in the Store.
Launch the Windows Maps app	Learn how to launch the Windows Maps app from your app.
Launch the People app	This topic describes the ms-people URI scheme. Your app can use this URI scheme to launch the People app for specific actions.
Launch screen snipping	Learn how to use URI schemes to open a new snip, or to open the Snip & Sketch app.
Enable apps for websites using app URI handlers	Drive user engagement with your app by supporting the Apps for Websites feature.

Launch a Windows app through file activation

This section shows how to set up your app to launch when a file of a certain type is opened.

Expand table

Topic	Description
Launch the default app for a file	Learn how to launch the default app for a file.
Handle file activation	Learn how to register your app to become the default handler for a certain file type.

Manage background tasks in your app

This section describes how to manage background tasks in your app.

Expand table

Topic	Description
Working with background tasks in Windows apps	Learn how to create and register a background task in your app with the Windows Runtime (WinRT) BackgroundTaskBuilder class.
BackgroundTaskBuilder in the Windows App SDK	Learn how to create and register a background task in a WinUI app with the Windows App SDK BackgroundTaskBuilder APIs.

Related content

- Launch Click to Do (Recall)
- App lifecycle and system services

Handle file activation in a Windows app

Article • 02/13/2025

Your app can register to become the default handler for a certain file type. Both Windows desktop applications and Universal Windows Platform (UWP) apps can register to be a default file handler. If the user chooses your app as the default handler for a certain file type, your app will be activated when that type of file is launched.

We recommend that you only register for a file type if you expect to handle all file launches for that type of file. If your app only needs to use the file type internally, then you don't need to register to be the default handler. If you do choose to register for a file type, you must provide the end user with the functionality that is expected when your app is activated for that file type. For example, a picture viewer app may register to display a .jpg file. For more info on file associations, see Guidelines for file types and URIs.

These steps show how to register for a custom file type, .alsdk, and how to activate your app when the user launches an .alsdk file.

> **Note**
>
> In Windows, certain URIs and file extensions are reserved for use by built-in apps and the operating system. Attempts to register your app with a reserved URI or file extension will be ignored. See Reserved URI scheme names and file types for an alphabetic list of Uri schemes that you can't register for your apps because they are either reserved or forbidden.

Important APIs

The following APIs are used in this topic:

- Windows.ApplicationModel.Activation.FileActivatedEventArgs
- Windows.UI.Xaml.Application.OnFileActivated
- AppInstance.GetActivatedEventArgs

Step 1: Specify the extension point in the package manifest

The app receives activation events only for the file extensions listed in the package manifest. Here's how you indicate that your app handles the files with the `.alsdk` extension.

1. In the **Solution Explorer**, double-click package.appxmanifest to open the manifest designer. Select the **Declarations** tab and in the **Available Declarations** drop-down, select **File Type Associations** and then click **Add**. See Programmatic Identifiers for more details of identifiers used by file associations.

   Here is a brief description of each of the fields that you may fill in the manifest designer:

Expand table

Field	Description
Display Name	Specify the display name for a group of file types. The display name is used to identify the file type in the Set Default Programs on the **Control Panel**.
Logo	Specify the logo that is used to identify the file type on the desktop and in the Set Default Programs on the **Control Panel**. If no Logo is specified, the application's small logo is used.
Info Tip	Specify the info tip for a group of file types. This tool tip text appears when the user hovers on the icon for a file of this type.
Name	Choose a name for a group of file types that share the same display name, logo, info tip, and edit flags. Choose a group name that can stay the same across app updates. **Note** The Name must be in all lower case letters.
Content Type	Specify the MIME content type, such as **image/jpeg**, for a particular file type. **Important Note about allowed content types:** Here is an alphabetic list of MIME content types that you cannot enter into the package manifest because they are either reserved or forbidden: **application/force-download**, **application/octet-stream**, **application/unknown**, **application/x-msdownload**.
File type	Specify the file type to register for, preceded by a period, for example, ".jpeg". **Reserved and forbidden file types:** See Reserved URI scheme names and file types for an alphabetic list of file types for built-in apps that you can't register for your UWP apps because they are either reserved or forbidden.

2. Enter `alsdk` as the **Name**.
3. Enter `.alsdk` as the **File Type**.
4. Enter "images\Icon.png" as the Logo.
5. Press Ctrl+S to save the change to package.appxmanifest.

The steps above add an Extension element like this one to the package manifest. The **windows.fileTypeAssociation** category indicates that the app handles files with the `.alsdk` extension.

XML

```
      <Extensions>
        <uap:Extension Category="windows.fileTypeAssociation">
          <uap:FileTypeAssociation Name="alsdk">
            <uap:Logo>images\icon.png</uap:Logo>
            <uap:SupportedFileTypes>
              <uap:FileType>.alsdk</uap:FileType>
            </uap:SupportedFileTypes>
          </uap:FileTypeAssociation>
        </uap:Extension>
      </Extensions>
```

Step 2: Add the proper icons

Apps that become the default for a file type have their icons displayed in various places throughout the system. For example, these icons are shown in:

- Windows Explorer Items View, context menus, and the Ribbon
- Default programs Control Panel
- File picker
- Search results on the Start screen

Include a 44x44 icon with your project so that your logo can appear in those locations. Match the look of the app tile logo and use your app's background color rather than making the icon transparent. Have the logo extend to the edge without padding it. Test your icons on white backgrounds. See Guidelines for tile and icon assets for more details about icons.

Step 3: Handle the activated event

The OnFileActivated event handler receives all file activation events.

C#

```
protected override void OnFileActivated(FileActivatedEventArgs args)
{
       // TODO: Handle file activation
       // The number of files received is args.Files.Count
```

```
        // The name of the first file is args.Files[0].Name
}
```

> **Note**
>
> When launched via File Contract, make sure that Back button takes the user back to the screen that launched the app and not to the app's previous content.

> **Note**
>
> In a WinUI app, in App.OnLaunched (or in fact at any time) you can call (AppInstance.GetActivatedEventArgs) to retrieve the activated event args, and check them to determine how the app was activated. See Application lifecycle functionality migration for more information about lifecycle differences between UWP and WinUI apps.

We recommend that you create a new XAML **Frame** for each activation event that opens a new page. That way, the navigation backstack for the new XAML Frame doesn't contain any previous content that the app might have on the current window when suspended. If you decide to use a single XAML **Frame** for Launch and for File Contracts, then you should clear the pages in the **Frame**'s navigation journal before navigating to a new page.

When your app is launched via File activation, you should consider including UI that allows the user to go back to the top page of the app.

Remarks

The files that you receive could come from an untrusted source. We recommend that you validate the content of a file before taking action on it.

Related content

- Association launching sample
- Launch the default app for a file
- Handle URI activation

Launch the default app for a file

Article • 02/13/2025

Learn how to launch the default app for a file from your WinUI, Universal Windows Platform (UWP), or other desktop app. Many apps need to work with files that they can't handle themselves. For example, e-mail apps receive a variety of file types and need a way to launch these files in their default handlers. These steps show how to use the Windows.System.Launcher Windows Runtime (WinRT) API to launch the default handler for a file that your app can't handle itself.

Important APIs

The following APIs are featured in this topic:

- Windows.System.Launcher.LaunchFileAsync

> **Note**
>
> Unless noted otherwise, all the WinRT APIs used in this topic can be used in both UWP apps, WinUI apps, and other desktop apps. To read more about enabling your desktop app to work with WinRT APIs, see **Call Windows Runtime APIs in desktop apps**.

Get the file object

First, get a Windows.Storage.StorageFile object for the file.

If the file is included in the package for your app, you can use the Package.InstalledLocation property to get a Windows.Storage.StorageFolder object and the Windows.Storage.StorageFolder.GetFileAsync method to get the StorageFile object.

If the file is in a known folder, you can use the properties of the Windows.Storage.KnownFolders class to get a StorageFolder and the GetFileAsync method to get the StorageFile object.

Launch the file

Windows provides several different options for launching the default handler for a file. These options are described in this chart and in the sections that follow.

Expand table

Option	Method	Description
Default launch	LaunchFileAsync(IStorageFile)	Launch the specified file with the default handler.
Open With launch	LaunchFileAsync(IStorageFile, LauncherOptions)	Launch the specified file letting the user pick the handler through the Open With dialog.
Launch with a recommended app fallback	LaunchFileAsync(IStorageFile, LauncherOptions)	Launch the specified file with the default handler. If no handler is installed on the system, recommend an app in the store to the user.
Launch with a desired remaining view	LaunchFileAsync(IStorageFile, LauncherOptions) (Windows-only)	Launch the specified file with the default handler. Specify a preference to stay on screen after the launch and request a specific window size. LauncherOptions.DesiredRemainingView isn't supported on the mobile device family.

Default launch

Call the Windows.System.Launcher.LaunchFileAsync(IStorageFile) method to launch the default app. This example uses the Windows.Storage.StorageFolder.GetFileAsync method to launch an image file, test.png, that is included in the app package.

C#

```
async void DefaultLaunch()
{
   // Path to the file in the app package to launch
   string imageFile = @"images\test.png";

   var file = await
Windows.ApplicationModel.Package.Current.InstalledLocation.GetFileAsync(imag
eFile);

   if (file != null)
   {
      // Launch the retrieved file
      var success = await Windows.System.Launcher.LaunchFileAsync(file);

      if (success)
      {
         // File launched
      }
      else
      {
         // File launch failed
```

```
      }
   }
   else
   {
      // Could not find file
   }
}
```

C++/WinRT

```
Windows::Foundation::IAsyncAction MainPage::DefaultLaunch()
{
    auto installFolder{
Windows::ApplicationModel::Package::Current().InstalledLocation() };

    Windows::Storage::StorageFile file{ co_await
installFolder.GetFileAsync(L"images\\test.png") };

    if (file)
    {
        // Launch the retrieved file
        bool success = co_await
Windows::System::Launcher::LaunchFileAsync(file);
        if (success)
        {
            // File launched
        }
        else
        {
            // File launch failed
        }
    }
    else
    {
        // Could not find file
    }
}
```

C++

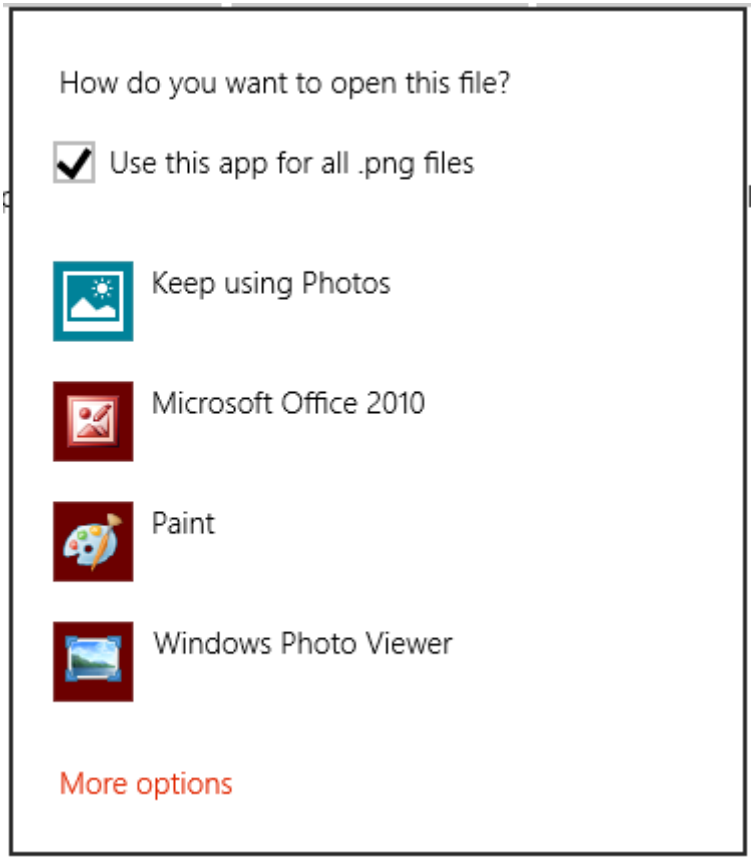
```
void MainPage::DefaultLaunch()
{
   auto installFolder = Windows::ApplicationModel::Package::Current-
>InstalledLocation;


concurrency::task<Windows::Storage::StorageFile^getFileOperation(installFold
er->GetFileAsync("images\\test.png"));
   getFileOperation.then([](Windows::Storage::StorageFile^ file)
   {
      if (file != nullptr)
      {
```

```
            // Launch the retrieved file
            concurrency::task<bool>
    launchFileOperation(Windows::System::Launcher::LaunchFileAsync(file));
            launchFileOperation.then([](bool success)
            {
               if (success)
               {
                  // File launched
               }
               else
               {
                  // File launch failed
               }
            });
         }
         else
         {
            // Could not find file
         }
      });
   }
```

Open With launch

Call the Windows.System.Launcher.LaunchFileAsync(IStorageFile, LauncherOptions) method with LauncherOptions.DisplayApplicationPicker set to **true** to launch the app that the user selects from the **Open With** dialog box.

We recommend that you use the **Open With** dialog box when the user may want to select an app other than the default for a particular file. For example, if your app allows the user to launch an image file, the default handler will likely be a viewer app. In some cases, the user may want to edit the image instead of viewing it. Use the **Open With** option along with an alternative command in the **AppBar** or in a context menu to let the user bring up the **Open With** dialog and select the editor app in these types of scenarios.



C#

```
async void DefaultLaunch()
{
   // Path to the file in the app package to launch
      string imageFile = @"images\test.png";

   var file = await
Windows.ApplicationModel.Package.Current.InstalledLocation.GetFileAsync(imag
eFile);

   if (file != null)
   {
      // Set the option to show the picker
      var options = new Windows.System.LauncherOptions();
      options.DisplayApplicationPicker = true;

      // Launch the retrieved file
      bool success = await Windows.System.Launcher.LaunchFileAsync(file,
options);
      if (success)
      {
         // File launched
      }
      else
      {
         // File launch failed
      }
   }
   else
```

```
    {
        // Could not find file
    }
}
```

C++/WinRT

```
Windows::Foundation::IAsyncAction MainPage::DefaultLaunch()
{
    auto installFolder{
Windows::ApplicationModel::Package::Current().InstalledLocation() };

    Windows::Storage::StorageFile file{ co_await
installFolder.GetFileAsync(L"images\\test.png") };

    if (file)
    {
        // Set the option to show the picker
        Windows::System::LauncherOptions launchOptions;
        launchOptions.DisplayApplicationPicker(true);

        // Launch the retrieved file
        bool success = co_await
Windows::System::Launcher::LaunchFileAsync(file, launchOptions);
        if (success)
        {
            // File launched
        }
        else
        {
            // File launch failed
        }
    }
    else
    {
        // Could not find file
    }
}
```

C++

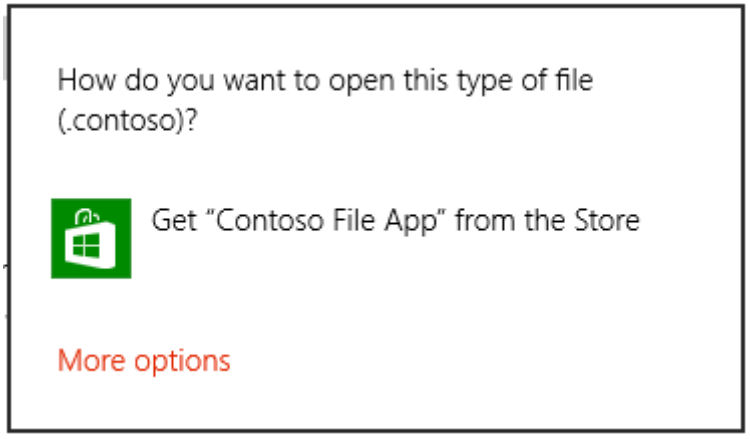
```
void MainPage::DefaultLaunch()
{
    auto installFolder = Windows::ApplicationModel::Package::Current-
>InstalledLocation;

    concurrency::task<Windows::Storage::StorageFile^>
getFileOperation(installFolder->GetFileAsync("images\\test.png"));
    getFileOperation.then([](Windows::Storage::StorageFile^ file)
    {
        if (file != nullptr)
        {
```

```
            // Set the option to show the picker
            auto launchOptions = ref new Windows::System::LauncherOptions();
            launchOptions->DisplayApplicationPicker = true;

            // Launch the retrieved file
            concurrency::task<bool>
launchFileOperation(Windows::System::Launcher::LaunchFileAsync(file,
launchOptions));
            launchFileOperation.then([](bool success)
            {
               if (success)
               {
                  // File launched
               }
               else
               {
                  // File launch failed
               }
            });
         }
         else
         {
            // Could not find file
         }
      });
}
```

Launch with a recommended app fallback

In some cases the user may not have an app installed to handle the file that you are launching. By default, Windows will handle these cases by providing the user with a link to search for an appropriate app on the store. If you would like to give the user a specific recommendation for which app to acquire in this scenario, you may do so by passing that recommendation along with the file that you are launching. To do this, call the Windows.System.Launcher.launchFileAsync(IStorageFile, LauncherOptions) method with LauncherOptions.PreferredApplicationPackageFamilyName set to the package family name of the app in the Store that you want to recommend. Then, set the LauncherOptions.PreferredApplicationDisplayName to the name of that app. Windows will use this information to replace the general option to search for an app in the store with a specific option to acquire the recommended app from the Store.

> **Note**
>
> You must set both of these options to recommend an app. Setting one without the other will result in a failure.



C#

```
async void DefaultLaunch()
{
   // Path to the file in the app package to launch
   string imageFile = @"images\test.contoso";

   // Get the image file from the package's image directory
   var file = await
Windows.ApplicationModel.Package.Current.InstalledLocation.GetFileAsync(imag
eFile);

   if (file != null)
   {
      // Set the recommended app
      var options = new Windows.System.LauncherOptions();
      options.PreferredApplicationPackageFamilyName =
"Contoso.FileApp_8wknc82po1e";
      options.PreferredApplicationDisplayName = "Contoso File App";

      // Launch the retrieved file pass in the recommended app
      // in case the user has no apps installed to handle the file
      bool success = await Windows.System.Launcher.LaunchFileAsync(file,
options);
      if (success)
      {
         // File launched
      }
      else
      {
         // File launch failed
      }
   }
   else
   {
      // Could not find file
   }
}
```

C++/WinRT

```
Windows::Foundation::IAsyncAction MainPage::DefaultLaunch()
{
    auto installFolder{
Windows::ApplicationModel::Package::Current().InstalledLocation() };

    Windows::Storage::StorageFile file{ co_await
installFolder.GetFileAsync(L"images\\test.png") };

    if (file)
    {
        // Set the recommended app
        Windows::System::LauncherOptions launchOptions;

launchOptions.PreferredApplicationPackageFamilyName(L"Contoso.FileApp_8wknc8
2po1e");
        launchOptions.PreferredApplicationDisplayName(L"Contoso File App");

        // Launch the retrieved file, and pass in the recommended app
        // in case the user has no apps installed to handle the file.
        bool success = co_await
Windows::System::Launcher::LaunchFileAsync(file, launchOptions);
        if (success)
        {
            // File launched
        }
        else
        {
            // File launch failed
        }
    }
    else
    {
        // Could not find file
    }
}
```

C++

```
void MainPage::DefaultLaunch()
{
   auto installFolder = Windows::ApplicationModel::Package::Current-
>InstalledLocation;

   concurrency::task<Windows::Storage::StorageFile^>
getFileOperation(installFolder->GetFileAsync("images\\test.contoso"));
   getFileOperation.then([](Windows::Storage::StorageFile^ file)
   {
      if (file != nullptr)
      {
         // Set the recommended app
         auto launchOptions = ref new Windows::System::LauncherOptions();
         launchOptions->PreferredApplicationPackageFamilyName =
```

```
"Contoso.FileApp_8wknc82po1e";
        launchOptions->PreferredApplicationDisplayName = "Contoso File
App";

        // Launch the retrieved file pass, and in the recommended app
        // in case the user has no apps installed to handle the file.
        concurrency::task<bool>
launchFileOperation(Windows::System::Launcher::LaunchFileAsync(file,
launchOptions));
        launchFileOperation.then([](bool success)
        {
           if (success)
           {
              // File launched
           }
           else
           {
              // File launch failed
           }
        });
     }
     else
     {
        // Could not find file
     }
  });
}
```

Launch with a Desired Remaining View (UWP-only)

Source apps that call LaunchFileAsync can request that they remain on screen after a file launch. By default, Windows attempts to share all available space equally between the source app and the target app that handles the file. Source apps can use the DesiredRemainingView property to indicate to the operating system that they prefer their app window to take up more or less of the available space. **DesiredRemainingView** can also be used to indicate that the source app does not need to remain on screen after the file launch and can be completely replaced by the target app. This property only specifies the preferred window size of the calling app. It doesn't specify the behavior of other apps that may happen to also be on screen at the same time.

> ⓘ **Note**
>
> Windows takes into account multiple different factors when it determines the source app's final window size, for example, the preference of the source app, the number of apps on screen, the screen orientation, and so on. By setting

DesiredRemainingView, you aren't guaranteed a specific windowing behavior for the source app.

C#

```
async void DefaultLaunch()
{
   // Path to the file in the app package to launch
   string imageFile = @"images\test.png";

   var file = await
Windows.ApplicationModel.Package.Current.InstalledLocation.GetFileAsync(imag
eFile);

   if (file != null)
   {
      // Set the desired remaining view
      var options = new Windows.System.LauncherOptions();
      options.DesiredRemainingView =
Windows.UI.ViewManagement.ViewSizePreference.UseLess;

      // Launch the retrieved file
      bool success = await Windows.System.Launcher.LaunchFileAsync(file,
options);
      if (success)
      {
         // File launched
      }
      else
      {
         // File launch failed
      }
   }
   else
   {
      // Could not find file
   }
}
```

C++/WinRT

```
Windows::Foundation::IAsyncAction MainPage::DefaultLaunch()
{
    auto installFolder{
Windows::ApplicationModel::Package::Current().InstalledLocation() };

    Windows::Storage::StorageFile file{ co_await
installFolder.GetFileAsync(L"images\\test.png") };

    if (file)
    {
```

```
        // Set the desired remaining view.
        Windows::System::LauncherOptions launchOptions;

launchOptions.DesiredRemainingView(Windows::UI::ViewManagement::ViewSizePref
erence::UseLess);

        // Launch the retrieved file.
        bool success = co_await
Windows::System::Launcher::LaunchFileAsync(file, launchOptions);
        if (success)
        {
            // File launched
        }
        else
        {
            // File launch failed
        }
    }
    else
    {
        // Could not find file
    }
}
```

C++

```
void MainPage::DefaultLaunch()
{
   auto installFolder = Windows::ApplicationModel::Package::Current-
>InstalledLocation;

   concurrency::task<Windows::Storage::StorageFile^>
getFileOperation(installFolder->GetFileAsync("images\\test.png"));
   getFileOperation.then([](Windows::Storage::StorageFile^ file)
   {
      if (file != nullptr)
      {
         // Set the desired remaining view.
         auto launchOptions = ref new Windows::System::LauncherOptions();
         launchOptions->DesiredRemainingView =
Windows::UI::ViewManagement::ViewSizePreference::UseLess;

         // Launch the retrieved file.
         concurrency::task<bool>
launchFileOperation(Windows::System::Launcher::LaunchFileAsync(file,
launchOptions));
         launchFileOperation.then([](bool success)
         {
            if (success)
            {
               // File launched
            }
            else
```

```
            {
                // File launch failed
            }
        });
    }
    else
    {
        // Could not find file
    }
});
}
```

Remarks

Your app can't select the app that is launched. The user determines which app is launched. The user can select either a UWP app or a Windows desktop app.

When launching a file, your app must be the foreground app, that is, it must be visible to the user. This requirement helps ensure that the user remains in control. To meet this requirement, make sure that you tie all file launches directly to the UI of your app. Most likely, the user must always take some action to initiate a file launch.

You can't launch file types that contain code or script if they are executed automatically by the operating system, such as, .exe, .msi, and .js files. This restriction protects users from potentially malicious files that could modify the operating system. You can use this method to launch file types that can contain script if they are executed by an app that isolates the script, such as, .docx files. Apps like Microsoft Word keep the script in .docx files from modifying the operating system.

If you try to launch a restricted file type, the launch will fail and your error callback will be invoked. If your app handles many different types of files and you expect that you will hit this error, we recommend that you provide a fallback experience to your user. For example, you could give the user an option to save the file to the desktop, and they could open it there.

Related content

- Launch the default app for a URI
- Handle file activation
- **Windows.System.Launcher.LaunchFileAsync**

Handle URI activation

Article • 02/13/2025

Learn how to register an app to become the default handler for a Uniform Resource Identifier (URI) scheme name. Both Windows desktop apps and Universal Windows Platform (UWP) apps can register to be a default handler for a URI scheme name. If the user chooses your app as the default handler for a URI scheme name, your app will be activated every time that type of URI is launched.

We recommend that you only register for a URI scheme name if you expect to handle all URI launches for that type of URI scheme. If you do choose to register for a URI scheme name, you must provide the end user with the functionality that is expected when your app is activated for that URI scheme. For example, an app that registers for the mailto: URI scheme name should open to a new e-mail message so that the user can compose a new e-mail. For more info on URI associations, see Files, folders, and libraries.

These steps show how to register for a custom URI scheme name, `alsdk://`, and how to activate your app when the user launches a `alsdk://` URI.

Important APIs

The following APIs are used in this topic:

- Windows.ApplicationModel.Activation.ProtocolActivatedEventArgs
- Windows.UI.Xaml.Application.OnActivated
- AppInstance.GetActivatedEventArgs

> **Note**
>
> In Windows, certain URIs and file extensions are reserved for use by built-in apps and the operating system. Attempts to register your app with a reserved URI or file extension will be ignored. See Reserved URI scheme names and file types for an alphabetic list of Uri schemes that you can't register for your apps because they are either reserved or forbidden.

Step 1: Specify the extension point in the package manifest

The app receives activation events only for the URI scheme names listed in the package manifest. Here's how you indicate that your app handles the `alsdk` URI scheme name.

1. In the **Solution Explorer**, double-click package.appxmanifest to open the manifest designer. Select the **Declarations** tab and in the **Available Declarations** drop-down, select **Protocol** and then click **Add**.

   Here is a brief description of each of the fields that you may fill in the manifest designer for the Protocol (see AppX Package Manifest for details):

Expand table

Field	Description
Logo	Specify the logo that is used to identify the URI scheme name in the Set Default Programs on the **Control Panel**. If no Logo is specified, the small logo for the app is used.
Display Name	Specify the display name to identify the URI scheme name in the Set Default Programs on the **Control Panel**.
Name	Choose a name for the Uri scheme.
	Note The Name must be in all lower case letters.
	Reserved and forbidden file types See Reserved URI scheme names and file types for an alphabetic list of Uri schemes that you can't register for your Windows apps because they are either reserved or forbidden.
Executable	Specifies the default launch executable for the protocol. If not specified, the app's executable is used. If specified, the string must be between 1 and 256 characters in length, must end with ".exe", and cannot contain these characters: >, <, :, ", |, ?, or *. If specified, the **Entry point** is also used. If the **Entry point** isn't specified, the entry point defined for the app is used.
Entry point	Specifies the task that handles the protocol extension. This is normally the fully namespace-qualified name of a Windows Runtime type. If not specified, the entry point for the app is used.
Start page	The web page that handles the extensibility point.
Resource group	A tag that you can use to group extension activations together for resource management purposes.
Desired View (Windows-only)	Specify the **Desired View** field to indicate the amount of space the app's window needs when it is launched for the URI scheme name. The possible values for **Desired View** are **Default**, **UseLess**, **UseHalf**, **UseMore**, or **UseMinimum**. **Note** Windows takes into account multiple different factors when determining the target app's final window size, for example, the preference of the source app, the number of apps on screen, the screen orientation, and so on. Setting **Desired**

Field	Description
	View doesn't guarantee a specific windowing behavior for the target app. **Mobile device family: Desired View** isn't supported on the mobile device family.

2. Enter `images\Icon.png` as the **Logo**.
3. Enter `SDK Sample URI Scheme` as the **Display name**
4. Enter `alsdk` as the **Name**.
5. Press Ctrl+S to save the change to package.appxmanifest.

   This adds an Extension element like this one to the package manifest. The **windows.protocol** category indicates that the app handles the `alsdk` URI scheme name.

XML

```
  <Applications>
      <Application Id= ... >
          <Extensions>
              <uap:Extension Category="windows.protocol">
                <uap:Protocol Name="alsdk">
                  <uap:Logo>images\icon.png</uap:Logo>
                  <uap:DisplayName>SDK Sample URI Scheme</uap:DisplayName>
                </uap:Protocol>
              </uap:Extension>
        </Extensions>
        ...
      </Application>
  </Applications>
```

Step 2: Add the proper icons

Apps that become the default for a URI scheme name have their icons displayed in various places throughout the system such as in the Default programs control panel. Include a 44x44 icon with your project for this purpose. Match the look of the app tile logo and use your app's background color rather than making the icon transparent. Have the logo extend to the edge without padding it. Test your icons on white backgrounds. See App icons and logos for more details about icons.

Step 3: Handle the activated event

> **Note**
>
> In a WinUI app, in App.OnLaunched (or in fact at any time) you can call (AppInstance.GetActivatedEventArgs) to retrieve the activated event args, and check them to determine how the app was activated. See Application lifecycle functionality migration for more information about lifecycle differences between UWP and WinUI apps.

In UWP apps, the OnActivated event handler receives all activation events. The **Kind** property indicates the type of activation event. This example is set up to handle Protocol activation events.

C#

```
public partial class App
{
   protected override void OnActivated(IActivatedEventArgs args)
  {
      if (args.Kind == ActivationKind.Protocol)
      {
         ProtocolActivatedEventArgs eventArgs = args as
ProtocolActivatedEventArgs;
         // TODO: Handle URI activation
         // The received URI is eventArgs.Uri.AbsoluteUri
      }
   }
}
```

C++/WinRT

```
void
App::OnActivated(Windows::ApplicationModel::Activation::IActivatedEventArgs
const& args)
{
    if (args.Kind() ==
Windows::ApplicationModel::Activation::ActivationKind::Protocol)
    {
        auto protocolActivatedEventArgs{
args.as<Windows::ApplicationModel::Activation::ProtocolActivatedEventArgs>()
};
        // TODO: Handle URI activation
        auto receivedURI{ protocolActivatedEventArgs.Uri().RawUri() };
    }
}
```

C++

```
void
App::OnActivated(Windows::ApplicationModel::Activation::IActivatedEventArgs^
args)
{
   if (args->Kind ==
Windows::ApplicationModel::Activation::ActivationKind::Protocol)
   {
      Windows::ApplicationModel::Activation::ProtocolActivatedEventArgs^
eventArgs =

dynamic_cast<Windows::ApplicationModel::Activation::ProtocolActivatedEventAr
gs^>(args);

      // TODO: Handle URI activation
      // The received URI is eventArgs->Uri->RawUri
   }
}
```

> **Note**
>
> When launched via Protocol Contract, make sure that Back button takes the user back to the screen that launched the app and not to the app's previous content.

The following code programmatically launches the app via its URI:

C#

```
   // Launch the URI
   var uri = new Uri("alsdk:");
   var success = await Windows.System.Launcher.LaunchUriAsync(uri)
```

For more details about how to launch an app via a URI, see Launch the default app for a URI.

It is recommended that apps create a new XAML Frame for each activation event that opens a new page. This way, the navigation backstack for the new XAML **Frame** will not contain any previous content that the app might have on the current window when suspended. Apps that decide to use a single XAML **Frame** for Launch and File Contracts should clear the pages on the **Frame** navigation journal before navigating to a new page.

When launched via Protocol activation, apps should consider including UI that allows the user to go back to the top page of the app.

Remarks

Any app or website can use your URI scheme name, including malicious ones. So any data that you get in the URI could come from an untrusted source. We recommend that you never perform a permanent action based on the parameters that you receive in the URI. For example, URI parameters could be used to launch the app to a user's account page, but we recommend that you never use them to directly modify the user's account.

> **Note**
>
> If you are creating a new URI scheme name for your app, be sure to follow the guidance in RFC 4395. This ensures that your name meets the standards for URI schemes.

> **Note**
>
> When a UWP app is launched via Protocol Contract, make sure that Back button takes the user back to the screen that launched the app and not to the app's previous content.

We recommend that apps create a new XAML **Frame** for each activation event that opens a new Uri target. This way, the navigation backstack for the new XAML **Frame** will not contain any previous content that the app might have on the current window when suspended.

If you decide that you want your apps to use a single XAML **Frame** for Launch and Protocol Contracts, clear the pages on the **Frame** navigation journal before navigating to a new page. When launched via Protocol Contract, consider including UI into your apps that allows the user to go back to the top of the app.

Related content

- Association UWP launching sample
- Default Programs
- Handle file activation

Launch the default Windows app for a URI

Article • 02/13/2025

Learn how to launch the default app for a Uniform Resource Identifier (URI). URIs allow you to launch another app to perform a specific task. This topic also provides an overview of the many URI schemes built into Windows. You can launch custom URIs too. For more info about registering a custom URI scheme and handling URI activation, see Handle URI activation.

URI schemes let you open apps by clicking hyperlinks. Just as you can start a new email using **mailto:**, you can open the default web browser using **http:** or **https:**.

This topic describes the following URI schemes built into Windows:

Expand table

URI Scheme	Launches
bingmaps:, ms-drive-to:, and ms-walk-to:	Maps app
http: and https:	Default web browser
mailto:	Default email app
microsoft-edge:	Microsoft Edge browser
ms-chat:	Messaging app
ms-people:	People app
ms-photos:	Photos app
ms-recall:	Click to do (part of Recall)
ms-settings:	Settings app
ms-store:	Store app
msnweather:	Weather app

For example, the following URI opens the default browser and displays the Microsoft Bing web site: `https://bing.com/`.

You can also launch custom URI schemes too. If there is no app installed to handle that URI, you can recommend an app for the user to install. For more info, see Recommend

an app if one is not available to handle the URI.

In general, your app can't select the app that is launched. The user determines which app is launched. More than one app can register to handle the same URI scheme. The exception to this is for reserved URI schemes. Registrations of reserved URI schemes are ignored. For the full list of reserved URI schemes, see Handle URI activation. In cases where more than one app may have registered the same URI scheme, your app can recommend a specific app to be launched. For more info, see Recommend an app if one is not available to handle the URI.

Important APIs

The following Windows Runtime (WinRT) APIs are used in this topic:

- LaunchUriAsync
- PreferredApplicationPackageFamilyName
- DesiredRemainingView

Note that many WinRT APIs will work with other desktop apps with package identity. There are some exceptions, and most of these are related to APIs that are specific to UI rendering or input. Some of the **LauncherOptions**, such as TreatAsUntrusted, only work in UWP apps. To read more about enabling your desktop app to work with WinRT APIs, see Call Windows Runtime APIs in desktop apps.

Call LaunchUriAsync to launch a URI

Use the LaunchUriAsync method to launch a URI. When you call this method, your app must be the foreground app, that is, it must be visible to the user. This requirement helps ensure that the user remains in control. To meet this requirement, make sure that you tie all URI launches directly to the UI of your app. The user must always take some action to initiate a URI launch. In a UWP app, if you attempt to launch a URI and your app isn't in the foreground, the launch will fail and your error callback will be invoked.

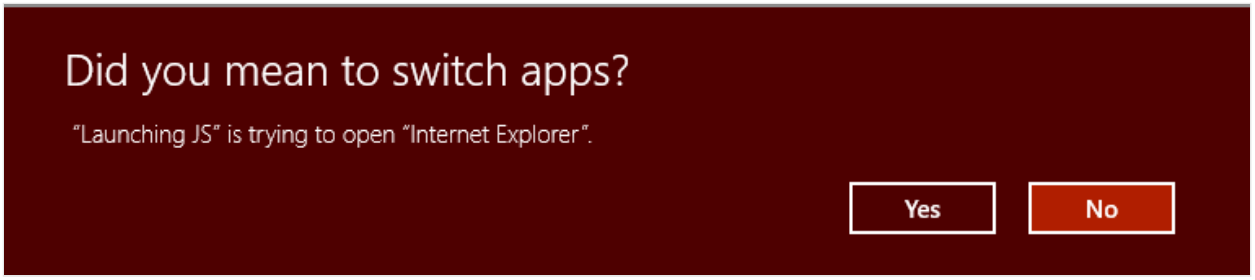
First create a System.Uri object to represent the URI, then pass that to the LaunchUriAsync method. Use the return result to see if the call succeeded, as shown in the following example.

cs

```
private async void launchURI_Click(object sender, RoutedEventArgs e)
{
    // The URI to launch
    var bingUri = new Uri(@"https://www.bing.com");
```

```
    // Launch the URI
    var success = await Windows.System.Launcher.LaunchUriAsync(bingUri);

    if (success)
    {
        // URI launched
    }
    else
    {
        // URI launch failed
    }
}
```

In some cases, the operating system will prompt the user to see if they actually want to switch apps.



> **ⓘ Important**
>
> This prompt is only supported by UWP apps. If Windows attempts to show this prompt for a desktop app, the launch will fail.

If your app is a UWP app, and you always want this prompt to occur, use the Windows.System.LauncherOptions.TreatAsUntrusted property to tell the operating system to display a warning.

cs

```
// The URI to launch
var uriBing = new Uri(@"http://www.bing.com");

// Set the option to show a warning
var promptOptions = new Windows.System.LauncherOptions
{
    TreatAsUntrusted = true
};

// Launch the URI
var success = await Windows.System.Launcher.LaunchUriAsync(uriBing,
promptOptions);
```

Recommend an app if one is not available to handle the URI

In some cases, the user might not have an app installed to handle the URI that you are launching. By default, the operating system handles these cases by providing the user with a link to search for an appropriate app on the store. If you want to give the user a specific recommendation for which app to acquire in this scenario, you can do so by passing that recommendation along with the URI that you are launching.

Recommendations are also useful when more than one app has registered to handle a URI scheme. By recommending a specific app, Windows will open that app if it is already installed.

To make a recommendation, call the Windows.System.Launcher.LaunchUriAsync(Uri, LauncherOptions) method with LauncherOptions.preferredApplicationPackageFamilyName set to the package family name of the app in the store that you want to recommend. The operating system uses this info to replace the general option to search for an app in the store with a specific option to acquire the recommended app from the store.

```cs
// Set the recommended app
var options = new Windows.System.LauncherOptions
{
    PreferredApplicationPackageFamilyName = "Contoso.URIApp_8wknc82po1e",
    PreferredApplicationDisplayName = "Contoso URI App"
};

// Launch the URI and pass in the recommended app
// in case the user has no apps installed to handle the URI
var success = await Windows.System.Launcher.LaunchUriAsync(uriContoso,
options);
```

Set remaining view preference

> **Important**
>
> This feature is only available in UWP apps. When used in a desktop app, the property will be ignored.

Source apps that call LaunchUriAsync can request that they remain on screen after a URI launch. By default, Windows attempts to share all available space equally between the

source app and the target app that handles the URI. Source apps can use the DesiredRemainingView property to indicate to the operating system that they prefer their app window to take up more or less of the available space. **DesiredRemainingView** can also be used to indicate that the source app doesn't need to remain on screen after the URI launch and can be completely replaced by the target app. This property only specifies the preferred window size of the calling app. It doesn't specify the behavior of other apps that may happen to also be on screen at the same time.

> **Note**
>
> Windows takes into account multiple different factors when it determines the source app's final window size, for example, the preference of the source app, the number of apps on screen, the screen orientation, and so on. By setting DesiredRemainingView, you aren't guaranteed a specific windowing behavior for the source app.

cs

```
// Set the desired remaining view.
var options = new Windows.System.LauncherOptions
{
    DesiredRemainingView =
Windows.UI.ViewManagement.ViewSizePreference.UseLess
};

// Launch the URI
var success = await Windows.System.Launcher.LaunchUriAsync(uriContoso,
options);
```

URI Schemes

The various URI schemes are described in this section.

Email URI scheme

Use the **mailto:** URI scheme to launch the default mail app.

Expand table

URI Scheme	Results
`mailto:`	Launches the default email app.

URI Scheme	Results
`mailto:\[email address\]`	Launches the email app and creates a new message with the specified email address on the To line. Note that the email is not sent until the user taps send.

HTTP URI scheme

Use the **http:** URI scheme to launch the default web browser.

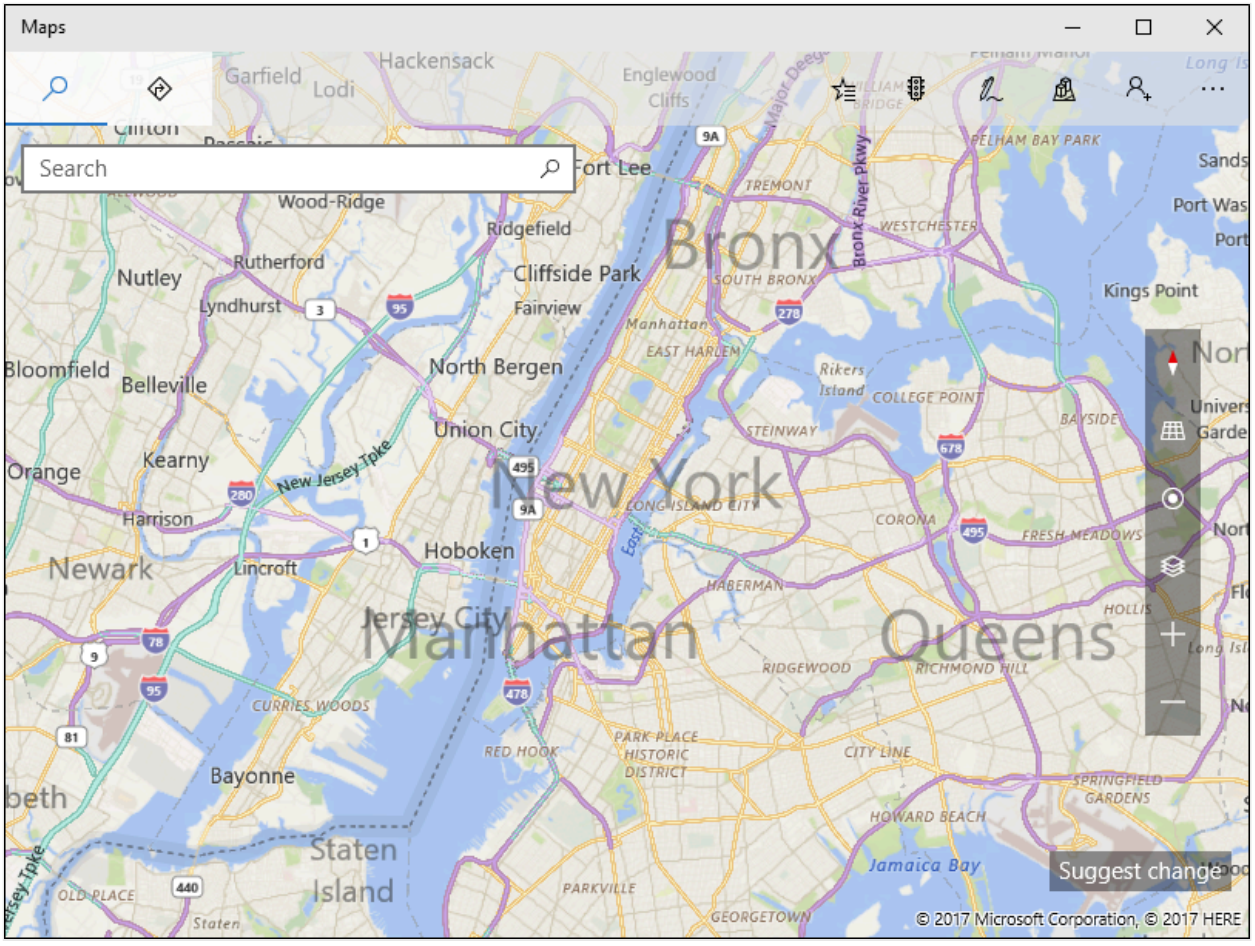
Expand table

URI Scheme	Results
`http:` or `https:`	Launches the default web browser.

Maps app URI schemes

Use the **bingmaps:**, **ms-drive-to:**, and **ms-walk-to:** URI schemes to launch the Windows Maps app to specific maps, directions, and search results. For example, the following URI opens the Windows Maps app and displays a map centered over New York City.

```
bingmaps:?cp=40.726966~-74.006076
```



For more info, see Launch the Windows Maps app. To use the map control in your UWP app, see Display maps with 2D, 3D, and Streetside views. If you're using Windows App SDK 1.5 or later in a WinUI app or other desktop app, you can use the MapControl.

Messaging app URI scheme

Use the **ms-chat:** URI scheme to launch the Microsoft Messaging app.

Expand table

URI scheme	Results
`ms-chat:`	Launches the Messaging app.
`ms-chat:?ContactID={contacted}`	Allows the messaging application to be launched with a particular contact's information.
`ms-chat:?Body={body}`	Allows the messaging application to be launched with a string to use as the content of the message.
`ms-chat:?Addresses={address}&Body={body}`	Allows the messaging application to be launched with a particular addresses' information, and with a string to use as the content of the message. Note: Addresses can be concatenated.
`ms-chat:?TransportId={transportId}`	Allows the messaging application to be launched with a particular transport ID.

People app URI scheme

Use the **ms-people:** URI scheme to launch the People app. For more info, see Launch the People app.

> **Important**
>
> The People app is moving to the new Outlook. You can take your contacts with you by selecting **Export contacts** on the People app's toolbar and then importing them into the new Outlook. See Manage your contacts and connect with people through the new Outlook for Windows (Preview) for more information.

Photos app URI scheme

Use the **ms-photos:** URI scheme to launch the Photos app to view an image or edit a video. For example:

Expand table

Operation	URI
View an image	`ms-photos:viewer?fileName=c:\users\userName\Pictures\image.jpg`
Edit a video	`ms-photos:videoedit?InputToken=123abc&Action=Trim&StartTime=01:02:03`

> **Note**
>
> If you are launching the Photos app from a UWP app, the URIs to edit a video or display an image are only available on desktop.

The following table lists additional supported URI schemes for the Photos app:

Expand table

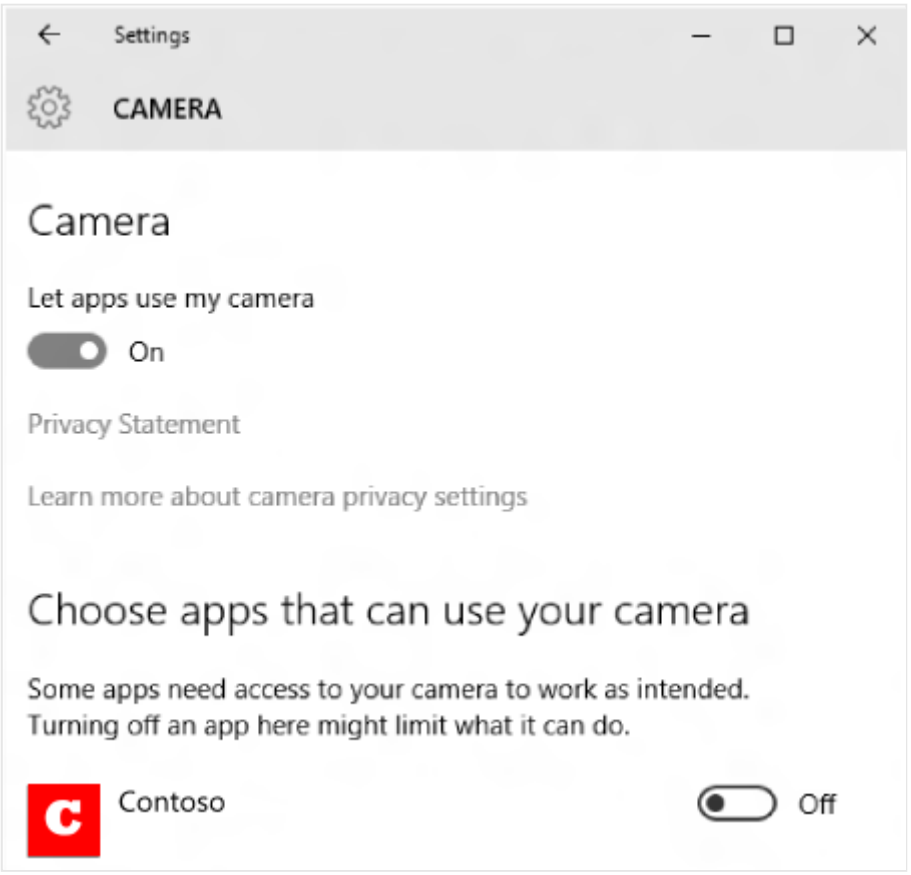
URI scheme	Results
`ms-photos:viewer?fileName={filename}`	Launches the Photos app to view the specified image where {filename} is a fully-qualified path name. For example: `c:\users\userName\Pictures\ImageToView.jpg`
`ms-photos:videoedit?InputToken={input token}`	Launches the Photos app in video editing mode for the file represented by the file token. **InputToken** is required. Use the SharedStorageAccessManager to get a token for a file.
`ms-photos:videoedit?Action={action}`	A parameter that indicates which video editing mode to open the Photos app in, where {action} is one of: **SlowMotion**, **FrameExtraction**, **Trim**, **View**, **Ink**. **Action** is required.
`ms-photos:videoedit?StartTime={timespan}`	An optional parameter that specifies where to start playing the video. `{timespan}` must be in the format `"hh:mm:ss.ffff"`. If not specified, defaults to `00:00:00.0000`

Settings app URI scheme

Use the **ms-settings:** URI scheme to launch the Windows Settings app. Launching to the Settings app is an important part of writing a privacy-aware app. If your app can't access a sensitive resource, we recommend providing the user a convenient link to the privacy settings for that resource.

For example, the following URI opens the Settings app and displays the camera privacy settings:

```
ms-settings:privacy-webcam
```



For more info, see Launch the Windows Settings app and Security and identity.

Store app URI scheme

Use the **ms-windows-store:** URI scheme to launch the Microsoft Store app. Open product detail pages, product review pages, and search pages, etc. For example, the following URI opens the Microsoft Store app and launches the home page of the Store.

```
ms-windows-store://home/
```

For more info, see Using ms-windows-store URIs.

Weather app URI scheme

Use the **msnweather:** URI scheme to launch the Weather app.

Expand table

URI Scheme	Results
`msnweather://forecast?la=\[latitude\]&lo=\[longitude\]`	Launches the Weather app in the Forecast page based on a location geographic coordinates.

URI Scheme	Results
	`latitude` refers to the latitude of the location. `longitude` refers to the longitude of the location.

Microsoft Edge URI scheme

Use the **microsoft-edge:** URI scheme to launch the Microsoft Edge browser to a specified URL.

Expand table

URI Scheme	Results
`microsoft-edge:https://example.com/`	Opens the Microsoft Edge browser and navigates to `https://example.com/`

You can use this URI scheme to launch the Microsoft Edge browser, regardless of the user's default browser setting.

Related content

Handle URI activation

Launch the Settings app

Launch the Windows Settings app

Article • 02/13/2025

Learn how to launch the Windows Settings app. This topic describes the `ms-settings:` URI scheme. Use this URI scheme to launch the Windows Settings app to specific settings pages.

Launching to the Settings app is an important part of writing a privacy-aware app. If your app can't access a sensitive resource, we recommend providing the user a convenient link to the privacy settings for that resource. For more information, see Security and identity.

Important APIs

The following Windows Runtime (WinRT) APIs are used in this topic:

- LaunchUriAsync
- PreferredApplicationPackageFamilyName
- DesiredRemainingView

Note that the WinRT APIs used in this topic can be used in both UWP apps, WinUI apps, and other desktop apps. To read more about enabling your desktop app to work with WinRT APIs, see Call Windows Runtime APIs in desktop apps.

How to launch the Settings app

To launch the **Settings** app, use the `ms-settings:` URI scheme as shown in the following examples.

XAML hyperlink control

In this example, a Hyperlink XAML control is used to launch the privacy settings page for the microphone using the `ms-settings:privacy-microphone` URI.

XML

```
<!--Set Visibility to Visible when access to the microphone is denied -->
<TextBlock x:Name="LocationDisabledMessage" FontStyle="Italic"
                Visibility="Collapsed" Margin="0,15,0,0"
TextWrapping="Wrap" >
          <Run Text="This app is not able to access the microphone. Go to "
/>
```

```
            <Hyperlink NavigateUri="ms-settings:privacy-microphone">
                <Run Text="Settings" />
            </Hyperlink>
        <Run Text=" to check the microphone privacy settings."/>
</TextBlock>
```

Calling LaunchUriAsync

Alternatively, your app can call the LaunchUriAsync method to launch the **Settings** app. This example shows how to launch to the privacy settings page for the camera using the `ms-settings:privacy-webcam` URI.

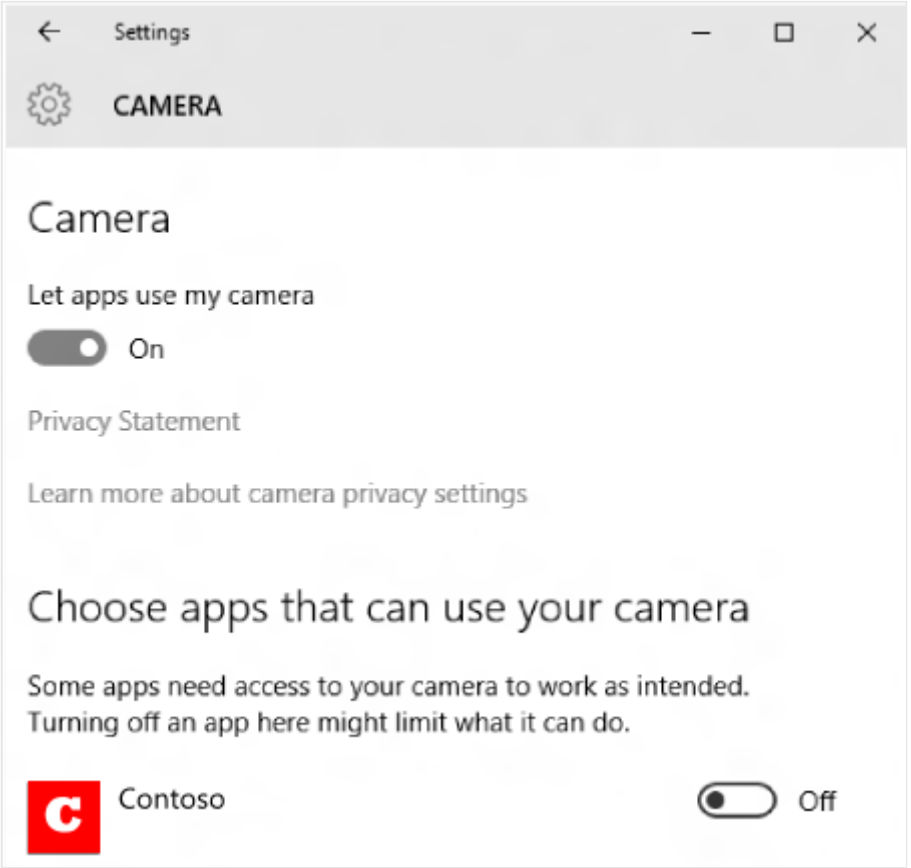
cs

```
bool result = await Windows.System.Launcher.LaunchUriAsync(new Uri("ms-settings:privacy-webcam"));
```

C++/WinRT

```
bool result = co_await Windows::System::Launcher::LaunchUriAsync(Windows::Foundation::Uri(L"ms-settings:privacy-webcam"));
```

The code above launches the privacy settings page for the camera:



For more info about launching URIs, see Launch the default app for a URI.

ms-settings: URI scheme reference

The following sections describe different categories of ms-settings URIs used to open various pages of the Settings app:

- Accounts
- Apps
- Control Center
- Cortana
- Devices
- Ease of access
- Extras
- Family Group
- Gaming
- Mixed reality
- Network and internet
- Personalization
- Phone
- Privacy
- Search
- Sound
- Surface Hub
- System
- Time and language
- Update and security

> **Note**
>
> The availability of some settings pages varies by Windows version and SKU. For some settings, the URI column also captures some usage information and any additional requirements that must be met for a page to be available.

Accounts

Expand table

Settings page	URI
Access work or school	ms-settings:workplace
Email & app accounts	ms-settings:emailandaccounts
Family & other people	ms-settings:otherusers
Provisioning	ms-settings:provisioning (only available on mobile and if the enterprise has deployed a provisioning package) ms-settings:workplace-provisioning (only available if enterprise has deployed a provisioning package)
Repair token	ms-settings:workplace-repairtoken
Set up a kiosk	ms-settings:assignedaccess
Sign-in options	ms-settings:signinoptions ms-settings:signinoptions-dynamiclock
Sync your settings	ms-settings:sync ms-settings:backup (**Backup page deprecated in Windows 11**)
Windows Anywhere	ms-settings:windowsanywhere (device must be Windows Anywhere-capable)
Windows Hello setup	ms-settings:signinoptions-launchfaceenrollment ms-settings:signinoptions-launchfingerprintenrollment
Your info	ms-settings:yourinfo

Apps

Expand table

Settings page	URI
Apps & Features	ms-settings:appsfeatures
App features	ms-settings:appsfeatures-app (Reset, manage add-on & downloadable content, etc. for the app) To access this page with a URI, use the ms-settings:appsfeatures-app URI and pass an optional parameter of the *package family name* of the app.

Settings page	URI
	Example: `ms-settings:appsfeatures-app?<PFN>`
Apps for websites	ms-settings:appsforwebsites
Default apps	ms-settings:defaultapps (**Behavior introduced in Windows 11, version 21H2 (with 2023-04 Cumulative Update) or 22H2 (with 2023-04 Cumulative Update), or later.**) Append the query string parameter in the following formats using the Uri-escaped name of an app to directly launch the default settings page for that app: - registeredAppMachine=<Uri-escaped per machine installed name of app> - registeredAppUser=<Uri-escaped per user installed name of app> - registeredAUMID=<Uri-escaped Application User Model ID> For more information, see Launch the Default Apps settings page.
Default browser settings	ms-settings:defaultbrowsersettings (**Deprecated in Windows 11**)
Manage optional features	ms-settings:optionalfeatures
Offline Maps	ms-settings:maps ms-settings:maps-downloadmaps (Download maps)
Startup apps	ms-settings:startupapps
Video playback	ms-settings:videoplayback

Control Center

Expand table

Settings page	URI
Control center	ms-settings:controlcenter

Cortana

Expand table

Settings page	URI
Cortana across my devices	ms-settings:cortana-notifications
More details	ms-settings:cortana-moredetails
Permissions & History	ms-settings:cortana-permissions
Searching Windows	ms-settings:cortana-windowssearch
Talk to Cortana	ms-settings:cortana-language ms-settings:cortana ms-settings:cortana-talktocortana

Important

Cortana voice assistance in Windows as a standalone app was retired in the spring of 2023. For more information, see End of support for Cortana.

Note

This Settings section on desktop will be called Search when the PC is set to regions where Cortana is not currently available or Cortana has been disabled. Cortana-specific pages (Cortana across my devices, and Talk to Cortana) will not be listed in this case.

Devices

Expand table

Settings page	URI
AutoPlay	ms-settings:autoplay
Bluetooth	ms-settings:bluetooth
Connected Devices	ms-settings:connecteddevices
Default camera	ms-settings:camera (**Behavior deprecated in Windows 10, version 1809 and later**)

Settings page	URI
Camera settings	ms-settings:camera (**Behavior introduced in Windows 11, build 22000 and later**) Append the query string parameter `cameraId` set to the Uri-escaped symbolic link name of a camera device to directly launch the settings for that camera. For more information, see Launch the camera settings page.
Mouse & touchpad	ms-settings:mousetouchpad (touchpad settings only available on devices that have a touchpad)
Pen & Windows Ink	ms-settings:pen
Printers & scanners	ms-settings:printers
Touch	ms-settings:devices-touch
Touchpad	ms-settings:devices-touchpad (only available if touchpad hardware is present)
Text Suggestions	ms-settings:devicestyping-hwkbtextsuggestions
Typing	ms-settings:typing
USB	ms-settings:usb
Wheel	ms-settings:wheel (only available if a **Surface Dial** device is paired)
Your phone	ms-settings:mobile-devices

Ease of access

Expand table

Settings page	URI
Audio	ms-settings:easeofaccess-audio
Closed captions	ms-settings:easeofaccess-closedcaptioning
Color filters	ms-settings:easeofaccess-colorfilter ms-settings:easeofaccess-colorfilter-adaptivecolorlink ms-settings:easeofaccess-colorfilter-bluelightlink
Display	ms-settings:easeofaccess-display
Eye control	ms-settings:easeofaccess-eyecontrol

Settings page	URI
Hearing devices	ms-settings:easeofaccess-hearingaids (**Added in Windows 11, Version 24H2**)
High contrast	ms-settings:easeofaccess-highcontrast
Keyboard	ms-settings:easeofaccess-keyboard
Magnifier	ms-settings:easeofaccess-magnifier
Mouse	ms-settings:easeofaccess-mouse
Mouse pointer & touch	ms-settings:easeofaccess-mousepointer
Narrator	ms-settings:easeofaccess-narrator ms-settings:easeofaccess-narrator-isautostartenabled
Speech	ms-settings:easeofaccess-speechrecognition
Text cursor	ms-settings:easeofaccess-cursor
Visual Effects	ms-settings:easeofaccess-visualeffects

Extras

Expand table

Settings page	URI
Extras	ms-settings:extras (only available if "settings apps" have been installed, for example, by a 3rd party)

Family Group

Expand table

Settings page	URI
Family Group	ms-settings:family-group

Gaming

Expand table

Settings page	URI
Game bar	ms-settings:gaming-gamebar
Game DVR	ms-settings:gaming-gamedvr
Game Mode	ms-settings:gaming-gamemode
Playing a game full screen	ms-settings:quietmomentsgame
TruePlay	ms-settings:gaming-trueplay (**As of Windows 10, version 1809 (10.0; Build 17763), this feature is removed from Windows**)

Mixed reality

> **Note**
>
> These settings are only available if the Mixed Reality Portal app is installed.

> **Important**
>
> Windows Mixed Reality devices are not supported with Windows 11, version 24H2 and newer.
>
> Windows Mixed Reality support is limited to Windows 10, version 20H2 through Windows 11, version 23H2.

Expand table

Settings page	URI
Audio and speech	ms-settings:holographic-audio
Environment	ms-settings:privacy-holographic-environment
Headset display	ms-settings:holographic-headset
Uninstall	ms-settings:holographic-management
Startup and desktop	ms-settings:holographic-startupandesktop

Network and internet

Expand table

Settings page	URI
Network & internet	ms-settings:network-status
Advanced settings	ms-settings:network-advancedsettings
Airplane mode	ms-settings:network-airplanemode ms-settings:proximity
Cellular & SIM	ms-settings:network-cellular
Dial-up	ms-settings:network-dialup
DirectAccess	ms-settings:network-directaccess (only available if DirectAccess is enabled)
Ethernet	ms-settings:network-ethernet
Manage known networks	ms-settings:network-wifisettings
Mobile hotspot	ms-settings:network-mobilehotspot
Proxy	ms-settings:network-proxy
VPN	ms-settings:network-vpn
Wi-Fi	ms-settings:network-wifi (only available if the device has a wifi adapter)
Wi-Fi provisioning	ms-settings:wifi-provisioning

Personalization

Expand table

Settings page	URI
Background	ms-settings:personalization-background
Choose which folders appear on Start	ms-settings:personalization-start-places
Colors	ms-settings:personalization-colors ms-settings:colors

Settings page	URI
Customize Copilot key on keyboard	ms-settings:personalization-textinput-copilot-hardwarekey
Dynamic Lighting	ms-settings:personalization-lighting
Fonts	ms-settings:fonts
Glance	ms-settings:personalization-glance (**Deprecated in Windows 10, version 1809 and later**)
Lock screen	ms-settings:lockscreen
Navigation bar	ms-settings:personalization-navbar (**Deprecated in Windows 10, version 1809 and later**)
Personalization (category)	ms-settings:personalization
Start	ms-settings:personalization-start
Taskbar	ms-settings:taskbar
Text input	ms-settings:personalization-textinput
Touch Keyboard	ms-settings:personalization-touchkeyboard
Themes	ms-settings:themes

Phone

Expand table

Settings page	URI
Your phone	ms-settings:mobile-devices ms-settings:mobile-devices-addphone ms-settings:mobile-devices-addphone-direct (Opens **Your Phone** app)
Device Usage	ms-settings:deviceusage

Privacy

Expand table

Settings page	URI
Accessory apps	ms-settings:privacy-accessoryapps (**Deprecated in Windows 10, version 1809 and later**)
Account info	ms-settings:privacy-accountinfo
Activity history	ms-settings:privacy-activityhistory
Advertising ID	ms-settings:privacy-advertisingid (**Deprecated in Windows 10, version 1809 and later**)
App diagnostics	ms-settings:privacy-appdiagnostics
Automatic file downloads	ms-settings:privacy-automaticfiledownloads
Background Apps	ms-settings:privacy-backgroundapps (**Deprecated in Windows 11, 21H2 and later**) **Note:** In Windows 11, the background app permissions are accessed individually. To view the permissions, go to **Apps->Installed apps** and then select **"..."** on a modern app and choose **Advanced options**. The advanced page is present for modern apps, and the **Background apps permissions** section will be present unless a group policy has been set or the user's global toggle value (the deprecated setting from Windows 10) is set. To access this page with a URI, use the `ms-settings:appsfeatures-app` URI and pass an optional parameter of the *package family name* of the app.
Background Spatial Perception	ms-settings:privacy-backgroundspatialperception
Calendar	ms-settings:privacy-calendar
Call history	ms-settings:privacy-callhistory
Camera	ms-settings:privacy-webcam
Contacts	ms-settings:privacy-contacts
Documents	ms-settings:privacy-documents
Downloads folder	ms-settings:privacy-downloadsfolder
Email	ms-settings:privacy-email
Eye tracker	ms-settings:privacy-eyetracker (requires eyetracker hardware)

Settings page	URI
Feedback & diagnostics	ms-settings:privacy-feedback
File system	ms-settings:privacy-broadfilesystemaccess
General	ms-settings:privacy or ms-settings:privacy-general
Graphics	ms-settings:privacy-graphicscaptureprogrammatic ms-settings:privacy-graphicscapturewithoutborder
Inking & typing	ms-settings:privacy-speechtyping
Location	ms-settings:privacy-location
Messaging	ms-settings:privacy-messaging
Microphone	ms-settings:privacy-microphone
Motion	ms-settings:privacy-motion
Music Library	ms-settings:privacy-musiclibrary
Notifications	ms-settings:privacy-notifications
Other devices	ms-settings:privacy-customdevices
Phone calls	ms-settings:privacy-phonecalls
Pictures	ms-settings:privacy-pictures
Radios	ms-settings:privacy-radios
Speech	ms-settings:privacy-speech
Tasks	ms-settings:privacy-tasks
Videos	ms-settings:privacy-videos
Voice activation	ms-settings:privacy-voiceactivation

Search

Expand table

Settings page	URI
Search	ms-settings:search
Search more details	ms-settings:search-moredetails
Search Permissions	ms-settings:search-permissions

Sound

Expand table

Settings page	URI
Volume mixer	ms-settings:apps-volume
Sound	ms-settings:sound
Sound devices	ms-settings:sound-devices
Default microphone	ms-settings:sound-defaultinputproperties
Default audio output	ms-settings:sound-defaultoutputproperties
Audio device properties (specific device)	ms-settings:sound-properties?endpointId={0.0.0.00000000}.{aaaaaaaa-0000-1111-2222-bbbbbbbbbbbb} **Note:** User of URI must know the `endpointId` string to use.
Audio device properties (specific device)	ms-settings:sound-properties?interfaceId=\\?\SWD#MMDEVAPI#{3.0.0.00000003}.{bbbbbbbb-1111-2222-3333-ccccccccccc}#{cccccccc-2222-3333-4444-dddddddddddd} **Note:** User of URI must know the `interfaceId` string to use and the string must be escaped correctly before sending.

Surface Hub

Expand table

Settings page	URI
Accounts	ms-settings:surfacehub-accounts
Session cleanup	ms-settings:surfacehub-sessioncleanup

Settings page	URI
Team Conferencing	ms-settings:surfacehub-calling
Team device management	ms-settings:surfacehub-devicemanagement
Welcome screen	ms-settings:surfacehub-welcome

System

Expand table

Settings page	URI
About	ms-settings:about
Advanced display settings	ms-settings:display-advanced (only available on devices that support advanced display options)
Battery Saver	ms-settings:batterysaver (only available on devices that have a battery, such as a tablet)
Battery Saver settings	ms-settings:batterysaver-settings (only available on devices that have a battery, such as a tablet)
Battery use	ms-settings:batterysaver-usagedetails (only available on devices that have a battery, such as a tablet)
Clipboard	ms-settings:clipboard
Default Save Locations	ms-settings:savelocations
Display	ms-settings:display ms-settings:screenrotation
Duplicating my display	ms-settings:quietmomentspresentation
During these hours	ms-settings:quietmomentsscheduled
Encryption	ms-settings:deviceencryption
Energy recommendations	ms-settings:energyrecommendations **(Available in February Moment update for Windows 11, Version 22H2, Build 22624 or later)**
Focus assist	ms-settings:quiethours
Graphics Settings	ms-settings:display-advancedgraphics (only available on devices that support advanced graphics options)

Settings page	URI
Graphics Default Settings	ms-settings:display-advancedgraphics-default
Multitasking	ms-settings:multitasking ms-settings:multitasking-sgupdate
Night light settings	ms-settings:nightlight
Projecting to this PC	ms-settings:project
Shared experiences	ms-settings:crossdevice
Tablet mode	ms-settings:tabletmode (**Removed in Windows 11**)
Taskbar	ms-settings:taskbar
Notifications & actions	ms-settings:notifications
Remote Desktop	ms-settings:remotedesktop
Phone	ms-settings:phone (**Deprecated in Windows 10, version 1809 and later**)
Power & sleep	ms-settings:powersleep
Presence sensing	ms-settings:presence (**Added in May Moment update for Windows 11, Version 22H2, Build 22624**)
Storage	ms-settings:storagesense
Storage Sense	ms-settings:storagepolicies
Storage recommendations	ms-settings:storagerecommendations
Disks & volumes	ms-settings:disksandvolumes

Time and language

Expand table

Settings page	URI
Date & time	ms-settings:dateandtime
Japan IME settings	ms-settings:regionlanguage-jpnime (available if the Microsoft Japan input method editor is installed)
Region	ms-settings:regionformatting

Settings page	URI
Language	ms-settings:keyboard ms-settings:keyboard-advanced ms-settings:regionlanguage ms-settings:regionlanguage-bpmfime ms-settings:regionlanguage-cangjieime ms-settings:regionlanguage-chsime-wubi-udp ms-settings:regionlanguage-quickime ms-settings:regionlanguage-korime
Pinyin IME settings	ms-settings:regionlanguage-chsime-pinyin (available if the Microsoft Pinyin input method editor is installed) ms-settings:regionlanguage-chsime-pinyin-domainlexicon ms-settings:regionlanguage-chsime-pinyin-keyconfig ms-settings:regionlanguage-chsime-pinyin-udp
Speech	ms-settings:speech
Wubi IME settings	ms-settings:regionlanguage-chsime-wubi (available if the Microsoft Wubi input method editor is installed)

Update and security

Expand table

Settings page	URI
Activation	ms-settings:activation
Backup	ms-settings:backup (page removed in Windows 11; opens Sync)
Delivery Optimization	ms-settings:delivery-optimization ms-settings:delivery-optimization-activity ms-settings:delivery-optimization-advanced
Find My Device	ms-settings:findmydevice
For developers	ms-settings:developers
Recovery	ms-settings:recovery
Launch Security Key Enrollment	ms-settings:signinoptions-launchsecuritykeyenrollment
Troubleshoot	ms-settings:troubleshoot
Windows Security	ms-settings:windowsdefender

Settings page	URI
Windows Insider Program	ms-settings:windowsinsider (only present if user is enrolled in WIP) ms-settings:windowsinsider-optin
Windows Update	ms-settings:windowsupdate ms-settings:windowsupdate-action
Windows Update-Active hours	ms-settings:windowsupdate-activehours
Windows Update-Advanced options	ms-settings:windowsupdate-options
Windows Update-Optional updates	ms-settings:windowsupdate-optionalupdates
Windows Update-Restart options	ms-settings:windowsupdate-restartoptions
Windows Update-Seeker on demand	ms-settings:windowsupdate-seekerondemand
Windows Update-View update history	ms-settings:windowsupdate-history

Related content

Launch the default app for a URI

Launching, resuming, and background tasks

Launch the Default Apps settings page

Article • 02/13/2025

Learn how to launch the Windows Settings app to display the Default Apps settings page from your app using the ms-settings URI scheme.

Windows defines a set of URIs that allow apps to launch the Windows Settings app and display a particular settings page. This article explains how to launch the Windows Settings app directly to the Default Apps settings page and, optionally, navigate directly to the settings for a specified default application. For more information, see Launch the Windows Settings app.

The Default Apps settings URL

`ms-settings:defaultapps` launches the Windows Settings app and navigates to the Default Apps settings page. Starting with Windows 11, version 21H2 (with 2023-04 Cumulative Update), 22H2 (with 2023-04 Cumulative Update), and 23H2 or later, you can append an additional query string parameter in escaped URI format to launch directly to the settings page for a specific application.

There are three query string parameters. The query string parameter to be used depends on how the application was installed.

Expand table

Query string parameter	Value to pass
registeredAppUser	Named value from HKEY_CURRENT_USER\Software\RegisteredApplications Use when the app was installed per user, and the registration for the app was written to HKEY_CURRENT_USER\Software\RegisteredApplications.
registeredAppMachine	Named value from HKEY_LOCAL_MACHINE\Software\RegisteredApplications Use when the app was installed per machine, and the registration for the app was written to HKEY_LOCAL_MACHINE\Software\RegisteredApplications.
registeredAUMID	Application User Model ID

Query string parameter	Value to pass
	Use when the app was registered with Package Manager using a manifest declaring that the app handles File Types (uap:FileTypeAssociation) or URI schemes (uap:Protocol).

> **Note**
>
> To get the registeredAUMID query string parameter to work after an OS upgrade, an app may need to increment its TargetDeviceFamily...MaxVersionTested value in its manifest. This will ensure that the app is reindexed for the user, which in turn will update the appropriate definitions used to process the deep link via protocol activation. MaxVersionTested should be updated to `10.0.22000.1817` for Windows 11, version 21H2 or `10.0.22621.1555` for Windows 11, version 22H2.

In the following example, `LaunchUriAsync` is called to launch the Windows Settings app. The `ms-settings:defaultapps` Uri specifies that the Default Apps settings page should be shown. Next, the app that should be launched is determined. As an example, "Microsoft Edge" was registered by the app in HKEY_LOCAL_MACHINE\Software\RegisteredApplications. Since it is a per machine installed app, `registeredAppMachine` is the query string parameter that should be used. The optional query string parameter `registeredAppMachine` is set to the registered name, escaped with a call to `Url.EscapeDataString`, to specify that the page for **Microsoft Edge** should be shown.

C#

```
private async void LaunchSettingsPage_Click(object sender, RoutedEventArgs
e)
{
    bool result = await Windows.System.Launcher.LaunchUriAsync(new Uri("ms-
settings:defaultapps?registeredAppMachine=" +
Uri.EscapeDataString("Microsoft Edge")));
}
```

Related content

Launch the Windows Settings app

Launch the default app for a URI

Using ms-windows-store URIs

Article • 02/19/2025

This topic describes the **ms-windows-store:** URI scheme. Your app can use this URI scheme to launch the Microsoft Store app to specific pages in the store by using the LaunchUriAsync method on Windows.

For example, you can open the Store to the Games page using the following code:

C#

```
bool result = await Windows.System.Launcher.LaunchUriAsync(new Uri("ms-windows-store://navigatetopage/?Id=Gaming"));
```

Opening the Store to specific sections

You can launch the Store App to a specific page or section using the following options:

Expand table

URI Scheme	Description	Notes
ms-windows-store://home	Launches the home page of the Store.	
ms-windows-store://navigatetopage/?Id=[vertical]	Launches a top-level vertical page of the store	Verticals include: `Home`, `Gaming`, `Entertainment`, `Productivity`, and `LOB`, but note that available verticals can vary by region.
ms-windows-store://downloadsandupdates	Launches the Updates and downloads page.	
ms-windows-store://mylibrary	Launches the Library page of the Store.	
ms-windows-store://settings	Launches the Store settings page.	

Opening to a specific product

You can launch the Store directly to the product detail page (PDP) for a specific product by using the Product ID for an App. While the Store app on Windows 10 and Windows 11 still supports Package Family Name (PFN) and App IDs, these are deprecated and may not be supported in the future. These values can be found in Partner Center on the App identity page in the Product management section for each app.

Starting with the October 2021 update to the Store app, there are two modes available for displaying the PDP. By default the Store app is opened to the product detail page. You can also launch the store into with a popup experience that displays a smaller PDP dialog with that only displays the essential details for your app and a single action button for users. For the popup experience, you can optionally specify the location of a window that the dialog should be centered above.

Expand table

URI Scheme	Description	Notes
ms-windows-store://pdp/?ProductId=9WZDNCRFHVJL	Launches the full product details page (PDP) for a product	This is the recommended way to link to a specific product.
ms-windows-store://pdp/?PFN=Microsoft.Office.OneNote_8wekyb3d8bbwe	Launches the full product details page (PDP) for a product	Using the package family name is deprecated.
ms-windows-store://pdp/?AppId=00001111-aaaa-2222-bbbb-3333cccc4444	Launches the full product details page (PDP) for a product	Using the App ID is deprecated.
ms-windows-store://pdp/?ProductId=9WZDNCRFHVJL&mode=mini	Launches the popup Store dialog experience	The popup experience only supports Product ID

Launching the rating and review experience for a product

To enable users to review your app, you can link to that PDP and launch directly into the rating and review dialog. Store ID is the recommended method to launch the Store app to a specific product detail page.

Expand table

URI Scheme	Description	Notes
ms-windows-store://review/?ProductId=9WZDNCRFHVJL	Launches the write a review experience for a product.	Using StoreId is recommended
ms-windows-store://review/?PFN=Microsoft.Office.OneNote_8wekyb3d8bbwe	Launches the write a review experience for a product.	Using product family name is deprecated.
ms-windows-store://review/?AppId=11112222-bbbb-3333-cccc-4444dddd5555	Launches the write a review experience for a product.	Using productid is deprecated

Searching the Store

You can launch the Store app directly into search results with several supported methods for searching store content.

Expand table

URI Scheme	Description	Notes
ms-windows-store://assoc/?Tags=Photos_Rich_Media_Edit,Camera_Capture_App	Launches a search for products associated with one or more tags.	Tags must be separated by commas.
ms-windows-store://search/?query=OneNote	Launches a search for the specified query.	Spaces in the query are allowed.
ms-windows-store://browse/?type=Apps&cat=Health+%26+fitness	Launches a search for products in a category.	
ms-windows-store://publisher/?name=Microsoft Corporation	Launches a search for products from the specified publisher.	Spaces in the name are allowed.
ms-windows-store://assoc/?FileExt=pdf	Launches a search for products associated with a file extension.	
ms-windows-store://assoc/?Protocol=ms-word	Launches a search for products associated with a protocol.	

Related content

Launching, resuming, and background tasks

Launch the Windows Maps app

Article • 02/13/2025

Learn how to launch the Windows Maps app from your app. This topic describes the **bingmaps:**, **ms-drive-to:**, **ms-walk-to:**, and **ms-settings:** Uniform Resource Identifier (URI) schemes. Use these URI schemes to launch the Windows Maps app to specific maps, directions, and search results or to download Windows Maps offline maps from the Settings app.

> **Tip**
>
> To learn more about launching the Windows Maps app from a Universal Windows Platform (UWP) app, download the UWP map sample from the Windows-universal-samples repo on GitHub. You can use the same Windows Runtime (WinRT) APIs for launching URIs in WinUI and other desktop apps to launch the Windows Maps app.

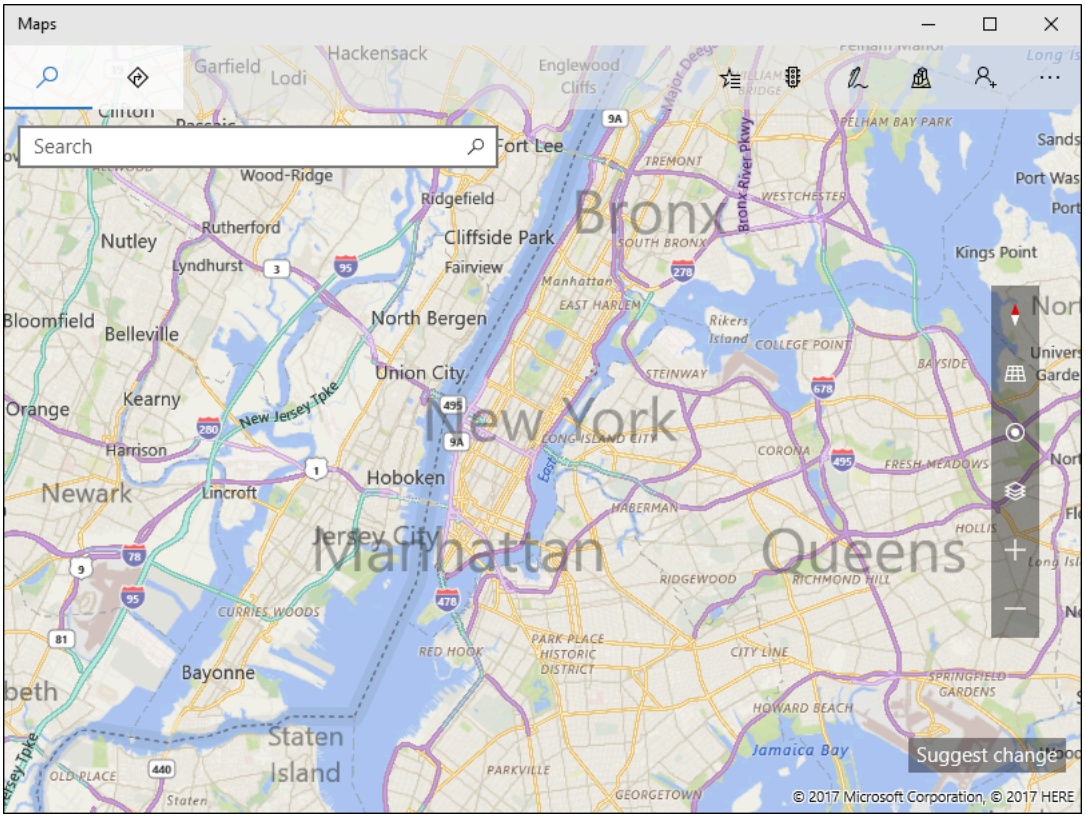
Introducing URIs

URI schemes let you open apps by clicking hyperlinks (or programmatically, in your app). Just as you can start a new email using **mailto:** or open a web browser using **http:**, you can open the Windows maps app using **bingmaps:**, **ms-drive-to:**, and **ms-walk-to:**.

- The **bingmaps:** URI provides maps for locations, search results, directions, and traffic.
- The **ms-drive-to:** URI provides turn-by-turn driving directions from your current location.
- The **ms-walk-to:** URI provides turn-by-turn walking directions from your current location.

For example, the following URI opens the Windows Maps app and displays a map centered over New York City:

```
bingmaps:?cp=40.726966~-74.006076
```



Here is a description of the URI scheme:

```
bingmaps:?query
```

In this URI scheme, *query* is a series of parameter name/value pairs:

```
&param1=value1&param2=value2 …
```

For a full list of the available parameters, see the bingmaps:, ms-drive-to:, and ms-walk-to: parameter reference. There are also examples later in this topic.

Launch a URI from your app

To launch the Windows Maps app from your app, call the LaunchUriAsync method with a **bingmaps:**, **ms-drive-to:**, or **ms-walk-to:** URI. The following example launches the same URI from the previous example. For more info about launching apps via URI, see Launch the default app for a URI.

```cs
// Center on New York City
var uriNewYork = new Uri(@"bingmaps:?cp=40.726966~-74.006076");

// Launch the Windows Maps app
var launcherOptions = new Windows.System.LauncherOptions();
launcherOptions.TargetApplicationPackageFamilyName = "Microsoft.WindowsMaps_8wekyb3d8bbwe";
var success = await Windows.System.Launcher.LaunchUriAsync(uriNewYork, launcherOptions);
```

In this example, the LauncherOptions class is used to help ensure the Windows Maps app is launched.

Display known locations

There are many options to control which part of the map to show. You can use the *cp* (center point) parameter with either the *rad* (radius) or the *lvl* (zoom level) parameters to show a location and choose how close to zoom in to it. When you use the *cp* parameter, you can also specify a *hdg* (heading) and *pit* (pitch) to control what direction to look. Another method is to use the *bb* (bounding box) parameter to provide the maximum south, east, north, and west coordinates of the area you want to show.

To control the type of view, use the *sty* (style) and *ss* (Streetside) parameters. The *sty* parameter lets you switch between road and aerial views. The *ss* parameter puts the map into a Streetside view. For more info about these and other parameters, see the bingmaps: parameter reference.

Expand table

Sample URI	Results
bingmaps:?	Opens the Maps app.
bingmaps:?cp=40.726966~-74.006076	Displays a map centered over New York City.
bingmaps:?cp=40.726966~-74.006076&lvl=10	Displays a map centered over New York City with a zoom level of 10.
bingmaps:?bb=39.719_-74.52~41.71_-73.5	Displays a map of New York City, which is the area specified in the **bb** argument.
bingmaps:?bb=39.719_-74.52~41.71_-73.5&cp=47~-122	Displays a map of New York City, which is the area specified in the bounding box argument. The center point for Seattle specified in the **cp** argument is ignored because *bb* is specified.
bingmaps:?collection=point.36.116584_-115.176753_Caesars%20Palace&lvl=16	Displays a map with a point named Caesars Palace (in Las Vegas) and sets the zoom level to 16.
bingmaps:?collection=point.40.726966_-74.006076_Some%255FBusiness	Displays a map with a point named Some_Business (in Las Vegas).
bingmaps:?cp=40.726966~-74.006076&trfc=1&sty=a	Displays a map of New York City with traffic on and aerial map style.
bingmaps:?cp=47.6204~-122.3491&sty=3d	Displays a 3D view of the Space Needle.
bingmaps:?cp=47.6204~-122.3491&sty=3d&rad=200&pit=75&hdg=165	Displays a 3D view of the Space Needle with a radius of 200m, a pitch of 75 degrees, and a heading of 165 degrees.
bingmaps:?cp=47.6204~-122.3491&ss=1	Displays a Streetside view of the Space Needle.

Display search results

When searching for places using the *q* parameter, we recommend making the terms as specific as possible and using the *cp*, *bb*, or *where* parameters to specify a search location. If you do not specify a search location and the user's current location isn't available, the search may not return meaningful results. Search results are displayed in the most appropriate map view. For more info about these and other parameters, see the bingmaps: parameter reference.

Expand table

Sample URI	Results
bingmaps:?q=1600%20Pennsylvania%20Ave,%20Washington,%20DC	Displays a map and searches for the address of the White House in Washington, D.C.
bingmaps:?q=coffee&where=Seattle	Searches for coffee in Seattle.
bingmaps:?cp=40.726966~-74.006076&where=New%20York	Searches for New York near the specified center point.
bingmaps:?bb=39.719_-74.52~41.71_-73.5&q=pizza	Searches for pizza in the specified bounding box (that is, in New York City).

Display multiple points

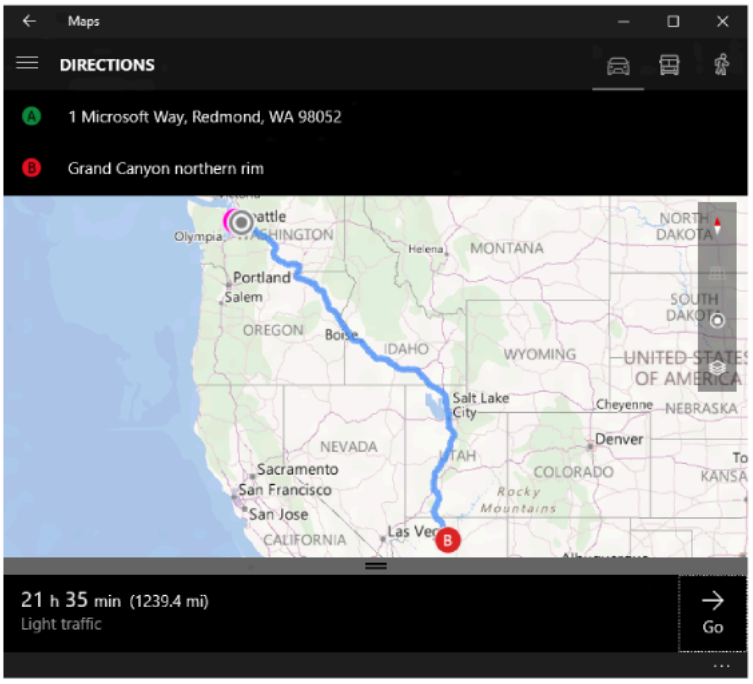
Use the *collection* parameter to show a custom set of points on the map. If there is more than one point, a list of points is displayed. There can be up to 25 points in a collection and they are listed in the order provided. The collection takes precedence over search and directions requests. For more info about this parameter and others, see the bingmaps: parameter reference.

Expand table

Sample URI
bingmaps:?collection=point.36.116584_-115.176753_Caesars%20Palace
bingmaps:?collection=point.36.116584_-115.176753_Caesars%20Palace&lvl=16
bingmaps:?collection=point.36.116584_-115.176753_Caesars%20Palace~point.36.113126_-115.175188_The%20Bellagio&lvl=16&cp=36.114902~-115.176669
bingmaps:?collection=point.40.726966_-74.006076_Fake%255FBusiness%255Fwith%255FUnderscore
bingmaps:? collection=name.Hotel%20List~point.36.116584_-115.176753_Caesars%20Palace~point.36.113126_-115.175188_The%20Bellagio&lvl=16&cp=36.114902~-115.176669

Display directions and traffic

You can display directions between two points using the *rtp* parameter; those points can be either addresses or latitude and longitude coordinates. Use the *trfc* parameter to show traffic information. To specify the type of directions: driving, walking, or transit, use the *mode* parameter. If *mode* isn't specified, directions will be provided using the user's preferred mode of transportation. For more info about these parameters and others, see the bingmaps: parameter reference.



Expand table

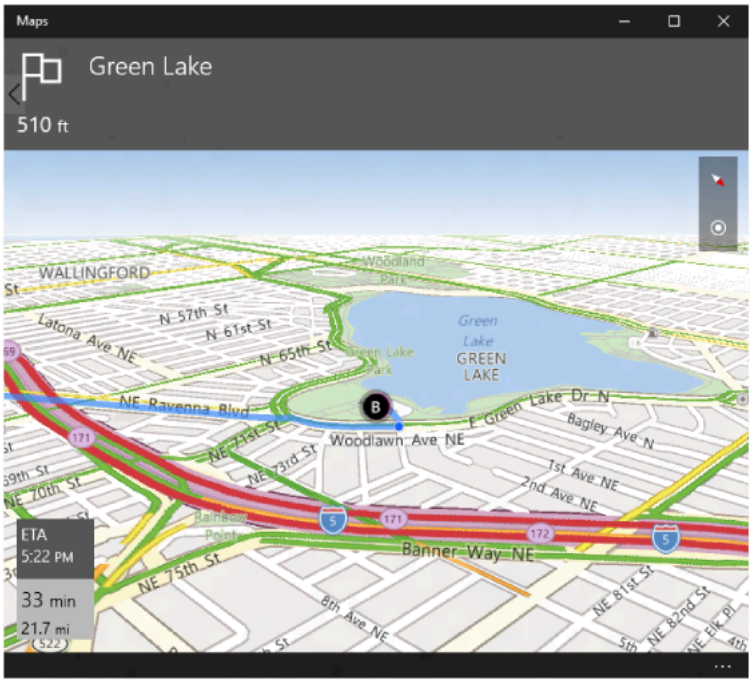
Sample URI	Results
bingmaps:?rtp=pos.44.9160_-110.4158~pos.45.0475_-109.4187	Displays a map with point-to-point directions. Because *mode* is not specified, directions will be provided using the user's mode of transportation preference.
bingmaps:?cp=43.0332~-87.9167&trfc=1	Displays a map centered over Milwaukee, WI with traffic.
bingmaps:?rtp=adr.One Microsoft Way, Redmond, WA 98052~pos.39.0731_-108.7238	Displays a map with directions from the specified address to the specified location.
bingmaps:? rtp=adr.1%20Microsoft%20Way,%20Redmond,%20WA,%2098052~pos.36.1223_-111.9495_Grand%20Canyon%20northern%20rim	Displays directions from 1 Microsoft Way, Redmond, WA, 98052 to the Grand Canyon's northern rim.
bingmaps:?rtp=adr.Davenport, CA~adr.Yosemite Village	Displays a map with driving directions from the specified location to the specified landmark.
bingmaps:?rtp=adr.Mountain%20View,%20CA~adr.San%20Francisco%20International%20Airport,%20CA&mode=d	Displays driving directions from Mountain View, CA to San Francisco International Airport, CA.
bingmaps:?rtp=adr.Mountain%20View,%20CA~adr.San%20Francisco%20International%20Airport,%20CA&mode=w	Displays walking directions from Mountain View, CA to San Francisco International Airport, CA.

Sample URI	Results
bingmaps:?rtp=adr.Mountain%20View,%20CA~adr.San%20Francisco%20International%20Airport,%20CA&mode=t	Displays transit directions from Mountain View, CA to San Francisco International Airport, CA.

Display turn-by-turn directions

The **ms-drive-to:** and **ms-walk-to:** URI schemes let you launch directly into a turn-by-turn view of a route. These URI schemes can only provide directions from the user's current location. If you must provide directions between points that do not include the user's current location, use the **bingmaps:** URI scheme as described in the previous section. For more info about these URI schemes, see the ms-drive-to: and ms-walk-to: parameter reference.

> **Important** When the **ms-drive-to:** or **ms-walk-to:** URI schemes are launched, the Maps app will check to see if the device has ever had a GPS location fix. If it has, then the Maps app will proceed to turn-by-turn directions. If it hasn't, the app will display the route overview, as described in Display directions and traffic.



Expand table

Sample URI	Results
ms-drive-to:? destination.latitude=47.680504&destination.longitude=-122.328262&destination.name=Green Lake	Displays a map with turn-by-turn driving directions to Green Lake from your current location.
ms-walk-to:? destination.latitude=47.680504&destination.longitude=-122.328262&destination.name=Green Lake	Displays a map with turn-by-turn walking directions to Green Lake from your current location.

Download offline maps

The **ms-settings:** URI scheme lets you launch directly into a particular page in the Settings app. While the **ms-settings:** URI scheme doesn't launch into the Maps app, it does allow you to launch directly to the Offline Maps page in the Settings app and displays a confirmation dialog to download the offline maps used by the Maps app. The URI scheme accepts a point specified by a latitude and longitude and automatically determines if there are offline maps available for a region containing that point. If the latitude and longitude passed happen to fall within multiple download regions, the confirmation dialog will let the user pick which of those regions to download. If offline maps are not available for a region containing that point, the offline Maps page in the Settings app is displayed with an error dialog.

Expand table

Sample URI	Results
ms-settings:maps-downloadmaps?latlong=47.6,-122.3	Opens the Settings app to the Offline Maps page with a confirmation dialog displayed to download maps for the region containing the specified latitude-longitude point.

bingmaps: parameter reference

The syntax for each parameter in this table is shown by using Augmented Backus–Naur Form (ABNF).

Expand table

Parameter	Definition	ABNF Definition and Example
cp	Center point	cp = "cp=" cpval cpval = degreeslat "~" degreeslon degreeslat = ["-"] 1*3DIGIT ["." 1*7DIGIT] degreeslon = ["-"] 1*2DIGIT ["." 1*7DIGIT] Example: cp=40.726966~-74.006076
bb	Bounding box	bb = "bb=" southlatitude "_" westlongitude "~" northlatitude "_" eastlongitude southlatitude = degreeslat northlatitude = degreeslat westlongitude = degreeslon eastlongitude = degreeslon degreeslat = ["-"] 13DIGIT ["." 17DIGIT] degreeslon = ["-"] 12DIGIT ["." 17DIGIT] Example: bb=39.719_-74.52~41.71_-73.5
where	Location	where = "where=" whereval whereval = 1*(ALPHA / DIGIT / "-" / "." / "_" / pct-encoded / "!" / "$" / "'" / "(" / ")" / "*" / "+" / "," / ";" / ":" / "@" / "/" / "?") Example: where=1600%20Pennsylvania%20Ave,%20Washington,%20DC

Parameter	Definition	ABNF Definition and Example
q	Query Term	q = "q=" whereval Example: q=mexican%20restaurants
lvl	Zoom Level	lvl = "lvl=" 1*2DIGIT [*"."* 1*2*DIGIT] Example: lvl=10.50
sty	Style	sty = "sty=" ("a" / "r"/"3d") Example: sty=a
rad	Radius	rad = "rad=" 1*8DIGIT Example: rad=1000
pit	Pitch	pit = "pit=" pitch Example: pit=60

Parameter	Definition	ABNF Definition and Example
hdg	Heading	hdg = "hdg=" heading Example: hdg=180
ss	Streetside	ss = "ss=" BIT Example: ss=1
trfc	Traffic	trfc = "trfc=" BIT Example: trfc=1
rtp	Route	rtp = "rtp=" (waypoint "~" [waypoint]) / ("~" waypoint) waypoint = ("pos." point) / ("adr." whereval) point = "point." pointval ["_" title] pointval = degreeslat "" degreeslon degreeslat = ["-"] 13DIGIT ["." 17DIGIT] degreeslon = ["-"] 12DIGIT ["." 17DIGIT] title = whereval whereval = 1(ALPHA / DIGIT / "-" / "." / "_" / pct-encoded / "!" / "$" / "'" / "(" / ")" / "" / "+" / "," / ";" / ":" / "@" / "/" / "?")

Parameter	Definition	ABNF Definition and Example
		Examples: rtp=adr.Mountain%20View,%20CA~adr.SFO rtp=adr.One%20Microsoft%20Way,%20Redmond,%20WA~pos.45.23423_-122.1232_My%20Picnic%20Spot

Parameter	Definition	ABNF Definition and Example
mode	Transportation mode	mode = "mode=" ("d" / "t" / "w") Example: mode=d
collection	Collection	collection = "collection="(name"~"/)point["~"point] name = "name." whereval whereval = 1(ALPHA / DIGIT / "-" / "." / "_" / pct-encoded / "!" / "$" / "'" / "(" / ")" / "" / "+" / "," / ";" / ":" / "@" / "/" / "?") point = "point." pointval ["_" title] pointval = degreeslat "" degreeslon degreeslat = ["-"] 13DIGIT ["." 17DIGIT]

Parameter	Definition	ABNF Definition and Example
		degreeslon = ["-"] 12DIGIT ["." 17DIGIT] title = whereval Example: collection=name.My%20Trip%20Stops~point.36.116584_-115.176753_Las%20Vegas~point.37.8268_-122.4798_Golden%20Gate%20Brid

ms-drive-to: parameter reference

The URI to launch a request for turn-by-turn driving directions does not need to be encoded and has the following format.

> **Note**
>
> You don't specify the starting point in this URI scheme. The starting point is always assumed to be the current location. If you need to specify a starting point other than the current location, see Display directions and traffic.

Expand table

Parameter	Definition	Example	Details
destination.latitude	Destination latitude	Example: destination.latitude=47.6451413797194	The latitude of the destination. Valid latitude values are between -90 and +90 inclusive.
destination.longitude	Destination longitude	Example: destination.longitude=-122.141964733601	The longitude of the destination. Valid longitude values are between -180 and +180 inclusive.
destination.name	Name of the destination	Example: destination.name=Redmond, WA	The name of the destination. You do not have to encode the **destination.name** value.

ms-walk-to: parameter reference

The URI to launch a request for turn-by-turn walking directions does not need to be encoded and has the following format.

> **Note**
>
> You don't specify the starting point in this URI scheme. The starting point is always assumed to be the current location. If you need to specify a starting point other than the current location, see Display directions and traffic.

Expand table

Parameter	Definition	Example	Details
destination.latitude	Destination latitude	Example: destination.latitude=47.6451413797194	The latitude of the destination. Valid latitude values are between -90 and +90 inclusive.
destination.longitude	Destination longitude	Example: destination.longitude=-122.141964733601	The longitude of the destination. Valid longitude values are between -180 and +180 inclusive.
destination.name	Name of the destination	Example: destination.name=Redmond, WA	The name of the destination. You do not have to encode the **destination.name** value.

ms-settings: parameter reference

The syntax for maps app specific parameters for the **ms-settings:** URI scheme is defined below. **maps-downloadmaps** is specified along with the **ms-settings:** URI in the form of **ms-settings:maps-downloadmaps?** to indicate the offline maps settings page.

Expand table

Parameter	Definition	Example	Details
latlong	Point defining offline map region.	Example: latlong=47.6,-122.3	The geopoint is specified by a comma separated latitude and longitude. Valid latitude values are between -90 and +90 inclusive. Valid longitude values are between -180 and +180 inclusive.

Related content

Launch the default app for a URI

Launch Snipping Tool

Article • 02/25/2025

This article specifies the protocol for integrating first and third-party applications with the Windows Snipping Tool using the **ms-screenclip:** URI (Uniform Resource Identifier) scheme. The protocol facilitates the capture of images and video (with audio) via Snipping Tool. App callers can customize and choose which Snipping Tool features their app will display. The protocol is designed to offer flexibility, security, and ease of use, aligning closely with familiar HTTP-based interactions. This shift can make the protocol more intuitive for developers and facilitate its integration with web technologies.

> **Note**
>
> This protocol launch replaces the previous existing experience documented here.

Supported features

The Snipping Tool protocol supports the following features:

- Rectangle Capture
- Freeform Capture
- Window Capture
- Ability to launch directly into a Screenshot or Recording
- Customizing features available
- Autosave feature is available, but can be disabled

Protocol Specification

URI Scheme: `{scheme}://{host}/{path}?{query}`

ms-screenclip: takes the following parameters:

- **Scheme:** `ms-screenclip` - The custom scheme for Snipping Tool's protocol.
- **Host:** Defines the Snipping Tool operation to perform (`capture` for snipping and recording, `discover` for querying capabilities).
- **Path:** Specifies the media type to capture, such as an image (`/image`) or video (`/video`).
- **Query:** Parameters for the specified schema.

Host (required)

- Capture
- Discover

Path (required)

- Image
- Video

Query

Capture Host

Expand table

Parameter	Type	Required	Description	Default
`api-version`	string	no	Protocol version (e.g., `"1.0"`). Ensures compatibility with future enhancements.	Latest version
`user-agent`	string	yes	Identifier for the calling application, used for logging and analytics.	n/a
`x-request-correlation-id`	string	no	Unique identifier for requests, allowing reference to a particular transaction or event chain.	Generated GUID if not provided.
`host`	string (enum)	yes	Specifies the **Snipping Tool action** to perform. Supported values: - `capture` – Opens Snipping Tool to take a screenshot or recording. - `discover` – Queries supported features.	`capture`
`path`	string (enum)	yes	Specifies the **type of media** being captured: - `image` – Screenshot capture. - `video` – Screen recording.	n/a
`enabledModes`	string (list)	no	Controls which snipping or recording modes are **enabled** in the UI. - `RectangleSnip` - `WindowSnip`	Defaults to the mode specified in the URI (`path`)

Parameter	Type	Required	Description	Default
			- `FreeformSnip` - `FullscreenSnip` - `RectangleRecord` - `SnippingAllModes` (all image modes) - `RecordAllModes` (all recording modes) - `All` (all supported modes)	
`auto-save`	boolean	no	Determines whether the captured **Screenshot or Recording** is automatically saved to the user's device. If set to `false`, the file is stored temporarily and can only be accessed using the token provided in the response. **Note:** The screenshot or recording will only be automatically saved if the user has enabled the corresponding Snipping Tool settings: - **"Automatically save original screenshots"** for screenshots. - **"Automatically save original screen recordings"** for recordings.	`false`
`redirect-uri`	URI	yes	Callback URI where the response will be sent. The calling application must register a handler for this protocol to receive and process the response.	n/a

Discover Host

The Snipping Tool protocol supports a `discover` endpoint that allows applications to retrieve available features, supported modes, and protocol versions dynamically. This is useful for ensuring compatibility with future updates or querying what capture methods are available.

How to Use

To retrieve Snipping Tool's supported capabilities, use the following URI:

```
ms-screenclip://discover?redirect-uri=my-snip-protocol-test-app://response
```

Response Format

The response is a JSON object containing:

- Version number (version) of the protocol.
- List of available capture paths (path), including:
  - capture/image (for snipping)
  - capture/video (for recording)
- Supported HTTP-like methods (methods).
- Available parameters (parameters) that can be used in requests.
- Description of each capability.

Example Response

JSON

```
{
  "version": 1.2,
  "capabilities": [
    {
      "path": "capture/image",
      "methods": ["GET"],
      "parameters": ["rectangle", "freeform", "window"],
      "description": "Captures an image with options for shape."
    },
    {
      "path": "capture/video",
      "methods": ["GET"],
      "parameters": [],
      "description": "Captures a video in a defined area."
    }
  ]
}
```

EnabledModes

The **enabledModes** parameter is designed to give developers granular control over the available UI options when invoking the ms-screenclip protocol. This allows for a tailored user experience that matches the specific requirements of the calling application. By specifying the **enabledModes** parameter, developers can restrict the user's choices in Snipping Tool's UI to ensure the output format meets their expectations.

Supported modes

The **enabledModes** parameter can accept the following modes:

- **RectangleSnip:** Enables rectangle capture mode.
- **WindowSnip:** Enables window capture mode.

- **FreeformSnip:** Enables freeform capture mode.
- **FullscreenSnip:** Enables fullscreen capture mode.
- **SnippingAllModes:** Enables all snipping (image capture) modes (RectangleSnip, WindowSnip, FreeformSnip, FullscreenSnip).
- **RectangleRecord:** Enables rectangle recording mode.
- **RecordAllModes:** Enables all recording modes (currently only RectangleRecord is available).
- **All:** Enables all supported modes (RectangleSnip, WindowSnip, FreeformSnip, FullscreenSnip, RectangleRecord).

> ⓘ **Important**
>
> If the **enabledModes** parameter is omitted, Snipping Tool will default to the mode explicitly specified in the URI (e.g., `rectangle`, `freeform`).
>
> - If a mode is explicitly specified in the URI (such as `rectangle`), only that mode will be available in Snipping Tool's UI.
> - If no default mode is specified in the URI, the request is considered invalid and will be ignored, even if **enabledModes** is provided.
> - The default mode parameter (e.g., `rectangle`, `freeform`) is required for **enabledModes** to function and for the request to be considered valid.

Examples

Example 1: Enable Only Rectangle Snip

`ms-screenclip://capture/image?rectangle&enabledModes=RectangleSnip&redirect-uri=my-snip-protocol-test-app://response`

Explanation: This command launches Snipping Tool's overlay with only the rectangle snipping mode enabled. The user will only be able to select a rectangular region for capture.

Example 2: Enable Rectangle Snip and Window Snip

`ms-screenclip://capture/image?rectangle&enabledModes=RectangleSnip,WindowSnip&redirect-uri=my-snip-protocol-test-app://response`

Explanation: This command launches Snipping Tool's overlay with both the rectangle and window snipping modes enabled. The user can choose between capturing a rectangular region or an entire window.

Example 3: Enable All Snipping Modes

`ms-screenclip://capture/image?rectangle&enabledModes=SnippingAllModes&redirect-uri=my-snip-protocol-test-app://response`

Explanation: This command launches Snipping Tool's overlay with all supported image snipping modes (RectangleSnip, WindowSnip, FreeformSnip, FullscreenSnip). The FullScreenSnip mode is excluded from interactive mode and will not be enabled.

Example 4: Enable Recording Mode Only

`ms-screenclip://capture/video?enabledModes=RecordAllModes&redirect-uri=my-snip-protocol-test-app://response`

Explanation: This command launches Snipping Tool's overlay with only the recording mode enabled. The user will be limited to selecting a rectangular region for recording, with no option to take a screenshot.

Example 5: Enable Multiple Snipping and Recording Modes

`ms-screenclip://capture/image?freeform&enabledModes=RectangleSnip,RectangleRecord&redirect-uri=my-snip-protocol-test-app://response`

Explanation: This command launches the Snipping Tool overlay with freeform snip, rectangle snip, and rectangle recording modes available. Since freeform mode is specified in the URI, it will be pre-selected by default. Users can choose to snip in freeform, rectangle, or record the selected region.

Example 6: Enable Rectangle Snipping Only

`ms-screenclip://capture/image?rectangle&redirect-uri=my-snip-protocol-test-app://response`

Explanation: Since rectangle is specified in the URI, only rectangle snipping mode will be available in Snipping Tool's UI.

Example 7: No `mode` specified

`ms-screenclip://capture/image?&enabledModes=All&redirect-uri=my-snip-protocol-test-app://response`

Explanation: This request is invalid because it does not specify a mode (e.g., `rectangle`, `window`, or `freeform`). Even though `enabledModes=All` is provided, a default mode must always be specified. As a result, Snipping Tool will ignore the call.

Key considerations

- **Mode Restrictions:** Developers should ensure that enabling specific modes aligns with the expected behavior of their application. Restricting UI options helps maintain a consistent user experience and ensures the resulting capture matches the application's needs.
- **Default Behavior:** If no `enabledModes` parameter is specified, only the `mode` specified in the URI will be available on the UI.

Security Considerations

Important: Snipping Tool does not validate the redirect-uri parameter. It is the responsibility of the calling application to ensure that:

- The `redirect-uri` points to a trusted destination.
- The `redirect-uri` does not expose the captured content to unauthorized applications.
- The application handling the response sanitizes and validates all incoming parameters before processing them.
- The protocol handler for the `redirect-uri` is properly registered and secured against misuse.
- **Required Fields:** Check that all required fields are provided and valid before proceeding with the snip operation.

Response if expected parameters **aren't** supplied or overloaded.

Responses to the caller

To ensure the response back to the caller via `redirect-uri` follows HTTP-based interaction principles, the response will mimic the structure of an HTTP response, but

will be conveyed through URI query parameters. This approach keeps the interaction web-standard compliant and familiar to developers.

Use of shared tokens

The use of the `SharedStorageAccessManager` class and of sharing tokens is subject to the following requirements and restrictions:

- A sharing token can only be redeemed once. After that, the token is no longer valid.
- A sharing token expires after 14 days and is no longer valid.
- The source app cannot provide more than 1000 sharing tokens. After a token is redeemed, removed, or expires, however, it no longer counts against the quota of the source app.

Response parameters

The following parameters are returned in the redirect URI:

Expand table

Parameter	Type	Description
Reason	String	The outcome and explanation for the snip.
Token (for success)	String	A token representing the captured media, which the application can use to access the file.
code	Int	The HTTP status code equivalent to provide a more granular understanding of the outcome.
x-request-correlation-id	String	A unique identifier value attached to requests and messages that allows reference to a particular transaction or event chain.

Retrieving a token

A sample application is available to test the process of calling the protocol and converting the response token into media. Use the SharedStorageAccessManager library to obtain the token.

Status codes

The following table lists the status codes that can be returned in the redirect URI:

Expand table

HTTP Status Code	Reason	Description
200	Success	The operation was successful.
400	Bad Request - Invalid or Missing Parameters	The request cannot be processed due to client error.
408	Request Timeout - Operation Took Too Long	The operation timed out before completion.
499	Client Closed Request - User Cancelled the Snip	The user cancelled the snip, closing the request.
500	Internal Server Error - Processing Failed	An error occurred on the server, preventing completion.

Example responses

Expand table

Use Case	Full URI Example
Successful Capture	`my-snip-protocol-test-app://response/?code=200&reason=Success&request-correlation-id=CF99C5A5-3907-461E-A9A9-D8AAFFAD5031&token=8930-ASDFA-ASDF4545`
Failed Capture	`my-snip-protocol-test-app://response/?code=400&reason=Bad%20Request%20-%20Invalid%20value%20Missing%20Parameters&request-correlation-id=C7696B38-52C8-419A-880F-F3350D7A6626`

Caller example

Below is a table displaying examples of full URIs constructed to initiate different types of snipping sessions using the screenclip: protocol. Each URI example demonstrates a combination of parameters to illustrate how you can customize Snipping Tool's behavior for different use cases.

Expand table

Use Case	Example URI	Description
Rectangle Mode	`ms-screenclip://capture/image?user-agent=TestApp&Rectangle&uri=my-snip-protocol-`	An application initiates an interactive rectangle capture

Use Case	Example URI	Description
	`test-app://response`	session, where the user selects the capture area. The result is redirected to a specific URI.
Video Mode	`ms-screenclip://capture/video?api-version=1.0&user-agent=TestApp&redirect-uri=my-snip-protocol-test-app://response`	A video capture. Always in rectangle mode.

Related content

- Use Snipping Tool to capture screenshots
- Launch screen snipping (Legacy)

Launch screen snipping (Legacy)

Article • 02/25/2025

**This is a legacy method for activating the Snipping Tool. It will be deprecated on 05/01/2025. Please refer to the updated documentation: Launch Snipping Tool.

The **ms-screenclip:** and **ms-screensketch:** URI schemes allows you to initiate snipping or editing screenshots.

Open a new snip from your app

The **ms-screenclip:** URI allows your app to automatically open up and start a new snip. The resulting snip is copied to the user's clipboard, but is not automatically passed back to the opening app.

ms-screenclip: takes the following parameters:

Expand table

Parameter	Type	Required	Description
source	string	no	A freeform string to indicate the source that launched the URI.
type	string	no	A string value to indicate which special type of capture is requested. This parameter can be omitted when starting a new snip. Values supported include: snapshot, recording*
clippingMode	string	no	A string value to indicate the clipping type for the snip. Values supported include: Rectangle, Freeform, Window
delayInSeconds	int	no	An integer value, from 1 to 30. Specifies the delay, in full seconds, between the URI call and when snipping begins.
callbackformat	string	no	This parameter is unavailable.

* `type=recording` is available only on Windows 11 PCs with Snipping Tool version 11.2307 or newer, and only when the default handler for ms-screenclip is set to "Snipping Tool" instead of "Screen Clipping".

Launching the Snipping Tool or Snip & Sketch App

The **ms-screensketch:** URI allows you to programmatically launch the Snipping Tool app (on Windows 11) or Snip & Sketch app (on Windows 10), and open a specific image in that app for annotation.

ms-screensketch: takes the following parameters:

Expand table

Parameter	Type	Required	Description
sharedAccessToken	string	no	A token identifying the file to open. Retrieved from SharedStorageAccessManager.AddFile. If this parameter is omitted, the app will be launched without a file open.
secondarySharedAccessToken	string	no	A string identifying a JSON file with metadata about the snip. The metadata may include a **clipPoints** field with an array of x,y coordinates, and/or a userActivity.
source	string	no	A freeform string to indicate the source that launched the URI.
isTemporary	bool	no	If set to True, Snipping Tool will try to delete the file after opening it.

The following example calls the LaunchUriAsync method to send an image to Snipping Tool from the user's app.

C#

```
bool result = await Windows.System.Launcher.LaunchUriAsync(new Uri("ms-screensketch:edit?source=MyApp&isTemporary=false&sharedAccessToken=2C37ADDA-B054-40B5-8B38-11CED1E1A2D"));
```

The following example illustrates what a file specified by the **secondarySharedAccessToken** parameter of **ms-screensketch** might contain:

JSON

```
{
  "clipPoints": [
    {
      "x": 0,
      "y": 0
    },
    {
```

```
      "x": 2080,
      "y": 0
    },
    {
      "x": 2080,
      "y": 780
    },
    {
      "x": 0,
      "y": 780
    }
  ],
  "userActivity": "{\"$schema\":\"http://activity.windows.com/user-
activity.json\",\"UserActivity\":\"type\",\"1.0\":\"version\",\"cross-
platform-identifiers\":
[{\"platform\":\"windows_universal\",\"application\":\"Microsoft.MicrosoftEd
ge_8wekyb3d8bbwe!MicrosoftEdge\"},
{\"platform\":\"host\",\"application\":\"edge.activity.windows.com\"}],\"act
ivationUrl\":\"microsoft-
edge:https://support.microsoft.com/help/13776/windows-use-snipping-tool-to-
capture-
screenshots\",\"contentUrl\":\"https://support.microsoft.com/help/13776/wind
ows-use-snipping-tool-to-capture-screenshots\",\"visualElements\":
{\"attribution\":{\"iconUrl\":\"https://www.microsoft.com/favicon.ico?
v2\",\"alternateText\":\"microsoft.com\"},\"description\":\"https://support.
microsoft.com/help/13776/windows-use-snipping-tool-to-capture-
screenshots\",\"backgroundColor\":\"#FF0078D7\",\"displayText\":\"Use
snipping tool to capture screenshots - Windows Help\",\"content\":
{\"$schema\":\"http://adaptivecards.io/schemas/adaptive-
card.json\",\"type\":\"AdaptiveCard\",\"version\":\"1.0\",\"body\":
[{\"type\":\"Container\",\"items\":[{\"type\":\"TextBlock\",\"text\":\"Use
snipping tool to capture screenshots - Windows
Help\",\"weight\":\"bolder\",\"size\":\"large\",\"wrap\":true,\"maxLines\":3
},
{\"type\":\"TextBlock\",\"text\":\"https://support.microsoft.com/help/13776/
windows-use-snipping-tool-to-capture-
screenshots\",\"size\":\"normal\",\"wrap\":true,\"maxLines\":3}]}]}},\"isRoa
mable\":true,\"appActivityId\":\"https://support.microsoft.com/help/13776/wi
ndows-use-snipping-tool-to-capture-screenshots\"}"
}
```


Related content

Launch a Windows app with a URI

Launch the People app

Article • 02/13/2025

This topic describes the **ms-people:** URI scheme. Your WinUI, UWP, and other Windows desktop apps can use this URI scheme to launch the People app for specific actions.

ms-people: URI scheme reference

Expand table

Results	URI scheme
Allows other apps to launch the People app Main page.	ms-people:
Allows other apps to launch the People app Settings page.	ms-people:settings
Allows other apps to provide a search string that will launch the People app with the result page of the search. The parameters are case sensitive. If you do not enter the syntax correctly, or are missing the search string value, the default behavior is to return a full list of contacts without any filtering.	ms-people:search?SearchString= <contactsearchinfo>
Launches to an existing contact card, if the contact is found. Or, launches to a temporary contact card, if no contact is found. If no input parameter is supplied, we will launch the People App with a contact list. The parameters are case sensitive. The order of the parameters doesn't matter. If there is more than one match, we will return the first match of the	ms-people:viewcontact?ContactId= <contactid>&AggregatedId= <aggid>&PhoneNumber= <phonenum>&Email= <email>&ContactName=<name>&Contact= <contactobj>

Results	URI scheme
contact.	
Launches to a Save-contact page within the People app to save the given contact with the supplied phone number or email address. The parameters are case sensitive. The order of the parameters doesn't matter.	ms-people:savetocontact?PhoneNumber=<phonenum>&Email=<email>&ContactName=<name>
Launches to the add a new contact page within the People app to save the given contact. Use **LaunchUriForResultsAsync** to open the save new contact page. Using **LaunchUriAsync** will only launch the People app Main page. The parameters are case sensitive. The order of the parameters doesn't matter. You can use any combination of supported parameters.	ms-people:savecontacttask?PhoneNumber=<phonenum>&Email=<email>&ContactName=<name>

ms-people:search: parameter reference

Expand table

Parameter	Description	Example
SearchString	Optional. The search string for the contact search information. The phone number or the contact name.	ms-people:search?SearchString=Smith

ms-people:viewcontact: parameter reference

Expand table

Parameter	Description	Example
ContactId	Optional. Contact Id of the contact.	ms-people:viewcontact?ContactId={ContactId}
PhoneNumber	Optional. Phone number of the contact.	ms-people:viewcontact?PhoneNumber=%2014257069326
Email	Optional. Email of the contact.	ms-people:viewcontact?Email=johnsmith@contsco.com
ContactName	Optional. Name of the contact.	ms-people:viewcontact?ContactName=John%20%Smith
Contact	Optional. Contact object.	ms-people:viewcontact?Contact={Serialized Contact}

ms-people:savetocontact: parameter reference

Expand table

Parameter	Description	Example
PhoneNumber	Optional. Phone number of the contact.	ms-people:savetocontact?PhoneNumber=%2014257069326
Email	Optional. Email of the contact.	ms-people:savetocontact?Email=johnsmith@contsco.com
ContactName	Optional. Name of the contact.	ms-people:savetocontact?Email=johnsmith@contsco.com&ContactName=John%20%Smith

ms-people:savecontacttask: parameter reference

Expand table

Parameter	Description
Company	Optional. Company name of the contact.
FirstName	Optional. First name of the contact.
HomeAddressCity	Optional. City of the home address.
HomeAddressCountry	Optional. Country of the home address.
HomeAddressState	Optional. State of the home address.
HomeAddressStreet	Optional. Street of the home address.
HomeAddressZipCode	Optional. Zip Code of the home address.
HomePhone	Optional. Home phone of the contact.
JobTitle	Optional. Job title of the contact.
LastName	Optional. Last name of the contact.
MiddleName	Optional. Middle name of the contact.
MobilePhone	Optional.

Parameter	Description
	Mobile phone number of the contact.
Nickname	Optional. Nickname of the contact.
Notes	Optional. Notes about the contact.
OtherEmail	Optional. Other Email of the contact.
PersonalEmail	Optional. Personal Email of the contact.
Suffix	Optional. Suffix of the contact.
Title	Optional. Title of the contact.
Website	Optional. Website of the contact.
WorkAddressCity	Optional. City of the work address.
WorkAddressCountry	Optional. Country of the work address.
WorkAddressState	Optional. State of the work address.
WorkAddressStreet	Optional. Street of work address.
WorkAddressZipCode	Optional. Zip Code of the work address.
WorkEmail	Optional.

Parameter	Description
	Work Email of the contact.
WorkPhone	Optional. Work phone number of the contact.

Related content

Launch a Windows app with a URI

Launch the default app for a URI

Enable apps for websites using app URI handlers

Article • 02/13/2025

Apps for Websites associates your app with a website so that when someone opens a link to your website, your app is launched instead of opening the browser. If your app is not installed, your website opens in the browser as usual. Users can trust this experience because only verified content owners can register for a link. Users will be able to check all of their registered web-to-app links by going to Settings > Apps > Apps for websites.

> **ⓘ Important**
>
> Starting with the Windows 10 Creators update and in all Windows 11 versions, supported links clicked in Microsoft Edge Legacy will launch the corresponding app. Supported links clicked in supported browsers (for example, Microsoft Edge Chromium, Firefox, Internet Explorer, etc.), will keep you in the browsing experience.

To enable web-to-app linking you will need to:

- Identify the URIs your app will handle in the manifest file
- A JSON file that defines the association between your app and your website. with the app Package Family Name at the same host root as the app manifest declaration.
- Handle the activation in the app.

Register to handle http and https links in the app manifest

Your app needs to identify the URIs for the websites it will handle. To do so, add the **Windows.appUriHandler** extension registration to your app's manifest file **Package.appxmanifest**.

For example, if your website's address is "msn.com" you would make the following entry in your app's manifest:

XML

```
<Applications>
  <Application ... >
```

```
      ...
      <Extensions>
         <uap3:Extension Category="windows.appUriHandler">
          <uap3:AppUriHandler>
            <uap3:Host Name="msn.com" />
          </uap3:AppUriHandler>
        </uap3:Extension>
      </Extensions>
  </Application>
</Applications>
```

The declaration above registers your app to handle links from the specified host. If your website has multiple addresses (for example: m.example.com, www.example.com, and example.com) then add a separate `<uap3:Host Name=... />` entry inside of the `<uap3:AppUriHandler>` for each address.

Associate your app and website with a JSON file

To ensure that only your app can open content on your website, include your app's package family name in a JSON file located in the web server root, or at the well-known directory on the domain. This signifies that your website gives consent for the listed apps to open content on your site. You can find the package family name in the Packages section in the app manifest designer.

> **ⓘ Important**
>
> The JSON file should not have a .json file suffix.

Create a JSON file (without the .json file extension) named **windows-app-web-link** and provide your app's package family name. For example:

JSON

```
[{
  "packageFamilyName" : "Your app's package family name, e.g
MyApp_9jmtgj1pbbz6e",
  "paths" : [ "*" ],
  "excludePaths" : [ "/news/*", "/blog/*" ]
 }]
```

Windows will make an https connection to your website and will look for the corresponding JSON file on your web server.

Wildcards

The JSON file example above demonstrates the use of wildcards. Wildcards allow you to support a wide variety of links with fewer lines of code. Web-to-app linking supports two types of wildcards in the JSON file:

Expand table

Wildcard	Description
*	Represents any substring
?	Represents a single character

For example, given `"excludePaths" : [ "/news/*", "/blog/*" ]` in the example above, your app will support all paths that start with your website's address (for example, msn.com), **except** those under `/news/` and `/blog/`. **msn.com/weather.html** will be supported, but not **msn.com/news/topnews.html**.

Multiple apps

If you have two apps that you would like to link to your website, list both of the application package family names in your **windows-app-web-link** JSON file. Both apps can be supported. The user will be presented with a choice of which is the default link if both are installed. If they want to change the default link later, they can change it in **Settings** > **Apps for Websites**. Developers can also change the JSON file at any time and see the change as early as the same day but no later than eight days after the update.

JSON

```
[{
  "packageFamilyName": "Your apps's package family name, e.g
MyApp_9jmtgj1pbbz6e",
  "paths": [ "*" ],
  "excludePaths" : [ "/news/*", "/blog/*" ]
 },
 {
  "packageFamilyName": "Your second app's package family name, for example,
MyApp2_8jmtgj2pbbz6e",
  "paths": [ "/example/*", "/links/*" ]
 }]
```

To provide the best experience for your users, use exclude paths to make sure that online-only content is excluded from the supported paths in your JSON file.

Exclude paths are checked first and if there is a match the corresponding page will be opened with the browser instead of the designated app. In the example above, '/news/*' includes any pages under that path while '/news*' (no forward slash trails 'news') includes any paths under 'news*' such as 'newslocal/', 'newsinternational/', and so on.

Handle links on Activation to link to content

Navigate to **App.xaml.cs** in your app's Visual Studio solution and in **OnActivated()** add handling for linked content. In the following example, the page that is opened in the app depends on the URI path:

CS

```
protected override void OnActivated(IActivatedEventArgs e)
{
    Frame rootFrame = Window.Current.Content as Frame;
    if (rootFrame == null)
    {
        ...
    }

    // Check ActivationKind, Parse URI, and Navigate user to content
    Type deepLinkPageType = typeof(MainPage);
    if (e.Kind == ActivationKind.Protocol)
    {
        var protocolArgs = (ProtocolActivatedEventArgs)e;
        switch (protocolArgs.Uri.AbsolutePath)
        {
            case "/":
                break;
            case "/index.html":
                break;
            case "/sports.html":
                deepLinkPageType = typeof(SportsPage);
                break;
            case "/technology.html":
                deepLinkPageType = typeof(TechnologyPage);
                break;
            case "/business.html":
                deepLinkPageType = typeof(BusinessPage);
                break;
            case "/science.html":
                deepLinkPageType = typeof(SciencePage);
                break;
        }
    }

    if (rootFrame.Content == null)
    {
        // Default navigation
```

```
            rootFrame.Navigate(deepLinkPageType, e);
        }

        // Ensure the current window is active
        Window.Current.Activate();
    }
```

> **ⓘ Important**
>
> Make sure to replace the final `if (rootFrame.Content == null)` logic with `rootFrame.Navigate(deepLinkPageType, e);` as shown in the example above.

Test in a local validation tool

You can test the configuration of your app and website by running the App host registration verifier tool which is available in:

%windir%\system32**AppHostRegistrationVerifier.exe**

Test the configuration of your app and website by running this tool with the following parameters:

AppHostRegistrationVerifier.exe *hostname packagefamilyname filepath*

- Hostname: Your website (for example, microsoft.com)
- Package Family Name (PFN): Your app's PFN
- File path: The JSON file for local validation (for example, C:\SomeFolder\windows-app-web-link)

If the tool does not return anything, validation will work on that file when uploaded. If there is an error code, it will not work.

You can enable the following registry key to force path matching for side-loaded apps as part of local validation:

`HKCU\Software\Classes\LocalSettings\Software\Microsoft\Windows\CurrentVersion\AppModel\SystemAppData\YourApp\AppUriHandlers`

Keyname: `ForceValidation` Value: `1`

Test it: Web validation

Close your application to verify that the app is activated when you click a link. Then, copy the address of one of the supported paths in your website. For example, if your website's address is "msn.com", and one of the support paths is "path1", you would use `http://msn.com/path1`

Verify that your app is closed. Press **Windows Key + R** to open the **Run** dialog box and paste the link in the window. Your app should launch instead of the web browser.

Additionally, you can test your app by launching it from another app using the LaunchUriAsync API. You can use this API to test on phones as well.

If you would like to follow the protocol activation logic, set a breakpoint in the **OnActivated** event handler.

AppUriHandlers tips

These tips will help you get the most out of the AppUriHandlers feature:

- Make sure to only specify links that your app can handle.
- List all of the hosts that you will support. Note that www.example.com and example.com are different hosts.
- Users can choose which app they prefer to handle websites in Settings.
- Your JSON file must be uploaded to an https server.
- If you need to change the paths that you wish to support, you can republish your JSON file without republishing your app. Users will see the changes in 1-8 days.
- All sideloaded apps with AppUriHandlers will have validated links for the host on install. You do not need to have a JSON file uploaded to test the feature.
- This feature works whenever your app is a UWP app launched with LaunchUriAsync or a Windows desktop app launched with ShellExecuteEx. If the URL corresponds to a registered App URI handler, the app will be launched instead of the browser.

Related content

Web-to-App example project

windows.protocol registration

Handle URI Activation

Reserved file and URI scheme names in Windows

Article • 02/13/2025

You can use URI associations to automatically launch your app when another app launches a specific URI scheme. But there are some URI associations that you can't use because they are reserved. If your app registers for a reserved association, that registration will be ignored. This topic lists the reserved file and URI scheme names that are not available to your app.

Reserved file types

There are two types of reserved file types: file types reserved for built-in apps and file types reserved for the operating system. When a file type reserved for a built-in app is launched, only the built-in app will launch. Any attempt to register your app with that file type is ignored. Similarly, any attempt to register your app with a file type reserved for the operating system also will be ignored.

File types reserved for built-in apps:

Expand table

Types			
.aac	.icon	.pem	.wdp
.aetx	.jpeg	.png	.wmv
.asf	.jxr	.pptm	.xap
.bmp	.m4a	.pptx	.xht
.cer	.m4r	.qcp	.xhtml
.dotm	.m4v	.rtf	.xltm
.dotx	.mov	.tif	.xltx
.gif	.mp3	.tiff	.xml
.hdp	.mp4	.txt	.xsl
.htm	.one	.url	.zip
.html	.onetoc2	.vcf	
.ico	.p7b	.wav	

File types reserved for the operating system

The following file types are reserved for the operating system:

Expand table

Types			
.accountpicture-ms	.its	.ops	.url
.ade	.jar	.pcd	.vb
.adp	.js	.pif	.vbe
.app	.jse	.pl	.vbp
.appx	.ksh	.plg	.vbs
.application	.lnk	.plsc	.vhd
.appref-ms	.mad	.prf	.vhdx
.asp	.maf	.prg	.vsmacros
.bas	.mag	.printerexport	.vsw
.bat	.mam	.provxml	.webpnp
.cab	.maq	.ps1	.ws
.chm	.mar	.ps1xml	.wsc
.cmd	.mas	.ps2	.wsf
.cnt	.mat	.ps2xml	.wsh
.com	.mau	.psc1	.xaml
.cpf	.mav	.psc2	.xdp
.cpl	.maw	.psm1	.xip
.crd	.mcf	.pst	.xnk
.crds	.mda	.pvw	
.crt	.mdb	.py	
.csh	.mde	.pyc	
.der	.mdt	.pyo	
.dll	.mdw	.rb	
.drv	.mdz	.rbw	
.exe	.msc	.rdp	
.fxp	.msh	.reg	
.fon	.msh1	.rgu	
.gadget	.msh1xml	.scf	
.grp	.msh2	.scr	
.hlp	.msh2xml	.shb	
.hme	.mshxml	.shs	
.hpj	.msi	.sys	

Types		
.hta	.msp	.theme
.inf	.mst	.tmp
.ins	.msu	.tsk
.isp	.ocx	.ttf

Reserved URI scheme names

The following URI scheme names are reserved and can't be used by your app:

Expand table

Types			
application.manifest	inffile	ms-settings:network-dialup	scrfile
application.reference	insfile	ms-settings:network-ethernet	scriptletfile
batfile	internetshortcut	ms-settings:network-mobilehotspot	shbfile
bing	javascript	ms-settings:network-proxy	shcmdfile
blob	jscript	ms-settings:network-wifi	shsfile
callto	jsefile	ms-settings:nfctransactions	smb
cerfile	ldap	ms-settings:notifications	stickynotes
chm.file	lnkfile	ms-settings:personalization	sysfile
cmdfile	mailto	ms-settings:privacy-calendar	tel
comfile	maps	ms-settings:privacy-contacts	telnet
cplfile	microsoft.powershellscript.1	ms-settings:privacy-customdevices	tn3270
dllfile	ms-accountpictureprovider	ms-settings:privacy-feedback	ttffile
drvfile	ms-appdata	ms-settings:privacy-location	unknown
dtmf	ms-appx	ms-settings:privacy-messaging	usertileprovider

Types			
exefile	ms-autoplay	ms-settings:privacy-microphone	vbefile
explorer.assocactionid.burnselection	ms-excel	ms-settings:privacy-speechtyping	vbscript
explorer.assocactionid.closesession	msi.package	ms-settings:privacy-webcam	vbsfile
explorer.assocactionid.erasedisc	msi.patch	ms-settings:proximity	wallet
explorer.assocactionid.zipselection	ms-powerpoint	ms-settings:regionlanguage	windows.gadget
explorer.assocprotocol.search-ms	ms-settings	ms-settings:screenrotation	windowsmediacenterapp
explorer.burnselection	ms-settings:batterysaver	ms-settings:speech	windowsmediacenterssl
explorer.burnselectionexplorer.closesession	ms-settings:batterysaver-settings	ms-settings:storagesense	windowsmediacenterweb
explorer.closesession	ms-settings:batterysaver-usagedetails	ms-settings:windowsupdate	wmp11.assocprotocol.mms
explorer.erasedisc	ms-settings:bluetooth	ms-settings:workplace	wsffile
explorer.zipselection	ms-settings:connecteddevices	ms-windows-store	wsfile
file	ms-settings:cortanasearch	ms-word	wshfile
fonfile	ms-settings:datasense	ocxfile	xbls
hlpfile	ms-settings:dateandtime	office	zune
htafile	ms-settings:display	onenote	
http	ms-settings:lockscreen	piffile	
https	ms-settings:maps	regfile	
iehistory	ms-settings:network-airplanemode	res	
ierss	ms-settings:network-cellular	rlogin	

Related content

- Launch a Windows app with a URI
- Handle URI activation

Working with background tasks in Windows apps

Article • 02/13/2025

Learn how to create and register a background task in your app with the Windows Runtime (WinRT) BackgroundTaskBuilder class.

Register a background task

See the BackgroundTask sample for a complete example of registering a background task in a Universal Windows Platform (UWP) app.

The following example shows the registration of a Win32 COM task that runs on a recurring 15 minute timer.

To register a background task, you must first create a new instance of the BackgroundTaskBuilder class. The `BackgroundTaskBuilder` class is used to create and register background tasks in your app. The following code example demonstrates how to create a new instance of the `BackgroundTaskBuilder` class:

C#

```
using System;
using Windows.ApplicationModel.Background;

public IBackgroundTaskRegistration
RegisterBackgroundTaskWithSystem(IBackgroundTrigger trigger, Guid
entryPointClsid, string taskName)
{
    BackgroundTaskBuilder builder = new BackgroundTaskBuilder();

    builder.SetTrigger(trigger);
    builder.SetTaskEntryPointClsid(entryPointClsid);

    BackgroundTaskRegistration registration;
    if (builder.Validate())
    {
        registration = builder.Register(taskName);
    }
    else
    {
        registration = null;
    }

    return registration;
}
```

```
RegisterBackgroundTaskWithSystem(new TimeTrigger(15, false),
typeof(TimeTriggeredTask).GUID, typeof(TimeTriggeredTask).Name);
```

The `RegisterBackgroundTaskWithSystem` method takes three parameters:

- `trigger`: The trigger that will start the background task.
- `entryPointClsid`: The class ID of the background task entry point.
- `taskName`: The name of the background task.

The `RegisterBackgroundTaskWithSystem` method creates a new instance of the `BackgroundTaskBuilder` class and sets the trigger and entry point class ID for the background task. The method then registers the background task with the system.

> ⓘ **Note**
>
> This class is not agile, which means that you need to consider its threading model and marshaling behavior. For more info, see Threading and Marshaling (C++/CX) and Using Windows Runtime objects in a multithreaded environment (.NET).

Handle modern standby in a background task

The **BackgroundTaskBuilder** and related APIs already allow packaged desktop applications to run background tasks. The API now extends these APIs to enable these appls to execute code in modern standby. The update also adds properties that can be queried by an app to determine if the system will throttle background tasks for the application in modern standby to conserve battery life. This enables scenarios like apps receiving VoIP calls or other push notifications from modern standby.

> ⓘ **Note**
>
> "Packaged desktop applications" in this section refers to Win32 applications that have package identity (i.e., are Desktop Bridge or Sparse Signed Packaged applications) and have a main (or wmain) function as their entry point.

The following example shows how an app developer can use the **BackgroundTaskBuilder** API to register at most one task with the specified task name. The sample also shows how to check and opt in the task registration to run in modern standby for the application's most critical tasks.

```
```

C#

```
// The following namespace is required for BackgroundTaskBuilder APIs.
using Windows.ApplicationModel.Background;

// The following namespace is required for API version checks.
using Windows.Foundation.Metadata;

// The following namespace is used for showing Toast Notifications. This
// namespace requires the Microsoft.Toolkit.Uwp.Notifications NuGet package
// version 7.0 or greater.
using Microsoft.Toolkit.Uwp.Notifications;

// Incoming calls are considered to be critical tasks to the operation of
the app.
const string IncomingCallTaskName = "IncomingCallTask";
const string NotificationTaskName = "NotificationTask";
const string PrefetchTaskName = "PrefetchTask";

public static bool IsAllowedInBackground(BackgroundAccessStatus status) {
    return ((status != BackgroundAccessStatus.Denied) &&
            (status != BackgroundAccessStatus.DeniedBySystemPolicy) &&
            (status != BackgroundAccessStatus.DeniedByUser) &&
            (status != BackgroundAccessStatus.Unspecified));
}

public async void RegisterTask(IBackgroundTrigger trigger,
                               Guid entryPointClsid,
                               string taskName,
                               bool isRunInStandbyRequested)
{
    var taskBuilder = new BackgroundTaskBuilder();
    taskBuilder.SetTrigger(trigger);
    taskBuilder.SetTaskEntryPointClsid(entryPointClsid);

    // Only the most critical background work should be allowed to proceed
in
    // modern standby. Additionally, some platforms may not support modern
    // or running background tasks in modern standby at all. Only attempt to
    // request modern standby execution if both are true. Requesting network
    // is necessary when running in modern standby to handle push
notifications.
    if (IsRunInStandbyRequested &&
taskBuilder.IsRunningTaskInStandbySupported)
    {
        var accessStatus =
BackgroundExecutionManager.GetAccessStatusForModernStandby();
        if (!IsAllowedInBackground(accessStatus)
        {
            await
BackgroundExecutionManager.RequestAccessKindForModernStandby(

BackgroundAccessRequestKind.AllowedSubjectToSystemPolicy,
                    "This app wants to receive incoming notifications while
```

```
your device is asleep");
        }

        accessStatus =
BackgroundExecutionManager.GetAccessStatusForModernStandby();

        if (IsAllowedInBackground(accessStatus)
        {
            taskBuilder.IsRunningTaskInStandbyRequested = true;
            taskBuilder.IsNetworkRequested = true;
        }
    }

    // Check that the registration is valid before attempting to register.
    if (taskBuilder.IsRegistrationValid)
    {
        // If a task with the specified name already exists, it is
unregistered
        // before a new one is registered. Note this API may still fail from
        // catastrophic failure (e.g., memory allocation failure).
        taskBuilder.Register(taskName);
    }

    return;
}

RegisterTask(new PushNotificationTrigger(), "{INSERT-YOUR-GUID-HERE}",
IncomingCallTaskName, true);
```

Check if background tasks have exceeded their budget in modern standby

The following sample code shows how an app developer may use the **BackgroundWorkCost.WasApplicationThrottledInStandby** and **BackgroundWorkCost.ApplicationEnergyUseLevel** to monitor and react to having their background tasks exhaust its app budget. The app developer may react by reducing lower priority work being done in modern standby. Note this relies on the code from the previous sample.

C#

```
public async void ReduceBackgroundCost()
{
    BackgroundTaskRegistration callTask;
    BackgroundTaskRegistration notificationTask;
    BackgroundTaskRegistration prefetchTask;

    // Nothing to do if the app was not or will not be throttled.
    if (!BackgroundWorkCost.WasApplicationThrottledInStandby &&
```

```
        (BackgroundWorkCost.ApplicationEnergyUseLevel !=
StandbyEnergyUseLevel.OverBudget))
    {
        return;
    }

    foreach (var task in BackgroundTaskRegistration.AllTasks)
    {
        switch (task.Value.Name) {
        case IncomingCallTaskName:
            callTask = task.Value;
            break;

        case NotificationTaskName:
            notificationTask = task.Value;
            break;

        case PrefetchTaskName:
            prefetchTask = task.Value;
            break;

        default:
        }
    }

    if (callTask.WasTaskThrottledInStandby)
    {
        // Unset the throttle flag after acknowledging it so the app can
        // react to the same task being throttled again in the future.
        task.Value.WasTaskThrottledInStandby = false;

        // Notify the user that the notification was missed.
        new ToastContentBuilder()
            .AddText("You missed a call")
            .AddText(task.Value.Name)
            .Show();

        // Because the incoming calls were not activated, demote less
notifications
        // tasks so the calls can be delivered promptly in the future.
        RegisterTask(notificationTask.Value.Trigger,
                     typeof(TimeTriggeredTask).GUID,
                     notificationTask.Value.Name,
                     false);
    }

    // Note that if incoming call tasks were throttled in some previous
modern
    // standby session, the application energy use was over budget for some
period.
    // Demote unimportant tasks like prefetch work to avoid calls and
notifications
    // from being throttled.
    if (callTask.WasTaskThrottledInStandby) ||
        (BackgroundWorkCost.ApplicationEnergyUseLevel ==
```

```
StandbyEnergyUseLevel.OverBudget))
    {
        RegisterTask(prefetchTask.Value.Trigger,
                     typeof(TimeTriggeredTask).GUID,
                     prefetchTask.Value.Name,
                     false);
    }

    return;
}
```

Monitor background task energy use trends

The following is an incremental end to end update to the following C++WinRT/C# sample code on GitHub.

The example shows how you can use the **BackgroundWorkCost.ApplicationEnergyUseTrend** to monitor how your background tasks trend towards exhausting their budget. You can also stop the most expensive background tasks from running in modern standby and prevent background tasks from running in modern standby if their app is using its budget too quickly. This sample relies on code from previous samples.

C#

```
public async void ReduceBackgroundCostPreemptively()
{
    BackgroundTaskRegistration mostExpensiveTask = null;

    // We can't do anything preemptively since the trend isn't known.
    if (!BackgroundWorkCost.IsApplicationEnergyUseTrendKnown)
    {
        return;
    }

    // The app is not trending towards being over budget, so this method can
    // return early.
    if ((BackgroundWorkCost.ApplicationEnergyUseTrend !=
EnergyUseTrend.OverBudget) &&
        (BackgroundWorkCost.ApplicationEnergyUseTrend !=
EnergyUseTrend.OverHalf))
    {
        return;
    }

    // The application is going exceeding its budget very quickly. Demote
the
    // most expensive task that is not the call task before call tasks start
being
```

```
    // throttled.
    if (BackgroundWorkCost.ApplicationEnergyUseTrend ==
EnergyUseTrend.OverBudget)
    {
        foreach (var task in BackgroundTaskRegistration.AllTasks)
        {
            if ((task.Value.Name != IncomingCallTaskName) &&
                ((mostExpensiveTask == null) ||

 (mostExpensiveTask.ApplicationEnergyUseTrendContributionPercentage <

task.Value.ApplicationEnergyUseTrendContributionPercentage)))
            {
                mostExpensiveTask = task.Value;
            }
        }
    }

    if (mostExpensiveTask != null)
    {
        RegisterTask(mostExpensiveTask.Trigger,
                     typeof(TimeTriggeredTask).GUID,
                     mostExpensiveTask.Name,
                     false);
    }

    // The application is trending toward eventually exceeding its budget.
Demote the
    // least important prefetch task before calls and notifications are
throttled.
    foreach (var task in BackgroundTaskRegistration.AllTasks)
    {
        if (task.Value.Name == PrefetchTaskName) {
            RegisterTask(task.Value.Trigger,
                         typeof(TimeTriggeredTask).GUID,
                         task.Value.Name,
                         false);
        }
    }

    return;
}
```

Background tasks and network connectivity

If your background task requires network connectivity, be aware of the following considerations.

Network related triggers

- Use a SocketActivityTrigger to activate the background task when a packet is received and you need to perform a short-lived task. After performing the task, the background task should terminate to save power.
- Use a ControlChannelTrigger to activate the background task when a packet is received and you need to perform a long-lived task.

Network related conditions and flags

- Add the InternetAvailable condition (BackgroundTaskBuilder.AddCondition) to your background task to delay triggering the background task until the network stack is running. This condition saves power because the background task won't execute until network access is available. This condition does not provide real-time activation.
- Regardless of the trigger you use, set IsNetworkRequested on your background task to ensure that the network stays up while the background task runs. This tells the background task infrastructure to keep the network up while the task is executing, even if the device has entered Connected Standby mode. If your background task does not use **IsNetworkRequested**, then your background task will not be able to access the network when in Connected Standby mode.

Related content

- BackgroundTaskBuilder class
- BackgroundTask sample
- Launching Windows apps and managing background tasks

Using background tasks in Windows apps

Article • 03/19/2025

This article provides an overview of using background tasks and describes how to create a new background task in a WinUI 3 app. For information about migrating your UWP apps with background tasks to WinUI, see the Windows App SDK Background task migration strategy.

BackgroundTaskBuilder in the Windows App SDK

Background tasks are app components that run in the background without a user interface. They can perform actions such as downloading files, syncing data, sending notifications, or updating tiles. They can be triggered by various events, such as time, system changes, user actions, or push notifications. These tasks can get executed when corresponding trigger occurs even when the app is not in running state.

The Windows Runtime (WinRT) BackgroundTaskBuilder was designed for UWP applications, and many of the background task triggers are not supported for full trust COM Components. They are supported only when registered with WinRT components that are launched with a `backgroundtaskhost` process. Due to this, Windows App SDK desktop applications can't directly register the full trust COM components to be launched with background task triggers. They require a workaround of including the WinRT components in the project. The BackgroundTaskBuilder in Windows App SDK API avoids this workaround so WinUI 3 and other desktop applications that use Windows App SDK can register the full trust COM components directly with background tasks.

Register a background task

The following example registers a background task for a full trust COM component using the Windows App SDK BackgroundTaskBuilder. See the Background task migration strategy guide for more information.

The C++ code to create and register a background task is as follows:

```
C++
```

```
//Using the Windows App SDK API for BackgroundTaskBuilder
winrt::Microsoft::Windows::ApplicationModel::Background::BackgroundTaskBuild
er builder;
SystemTrigger trigger = SystemTrigger(SystemTriggerType::TimeZoneChange,
false);
auto backgroundTrigger = trigger.as<IBackgroundTrigger>();
builder.SetTrigger(backgroundTrigger);
builder.AddCondition(SystemCondition(SystemConditionType::InternetAvailable)
);
builder.SetTaskEntryPointClsid(classGuid);
builder.Register();
```

To create and register the background task in C#, the code is as follows:

C#

```
//Using the Windows App SDK API for BackgroundTaskBuilder
var builder = new
Microsoft.Windows.ApplicationModel.Background.BackgroundTaskBuilder();
var trigger = new SystemTrigger(SystemTriggerType.TimeZoneChange, false);
var backgroundTrigger = trigger as IBackgroundTrigger;
builder.SetTrigger(backgroundTrigger);
builder.AddCondition(new
SystemCondition(SystemConditionType.InternetAvailable));
builder.SetTaskEntryPointClsid(classGuid);
builder.Register();
```

The corresponding package manifest entry for the background task is as follows:

XML

```
<Extension Category="windows.backgroundTasks"
EntryPoint="Microsoft.Windows.ApplicationModel.Background.UniversalBGTask.Ta
sk">
    <BackgroundTasks>
        <Task Type="general"/>
    </BackgroundTasks>
</Extension>
```

A full WinUI 3 background task registration sample can be found on GitHub.

Related content

- Guidelines for background tasks in UWP apps
- Background task migration strategy

Audio, video, and camera

Article • 01/16/2025

This article provides an index of development features that are related to scenarios involving audio, video, and camera in Windows apps.

Windows OS features

Windows 10 and later OS releases provide a wide variety of APIs related to audio, video, and camera scenarios for apps. These features are available via a combination of WinRT and Win32 (C++ and COM) APIs provided by the Windows SDK.

WinRT APIs

The following articles provide information about features available via WinRT APIs provided by the Windows SDK.

Expand table

Article	Description
Camera	Learn about how to use the camera or microphone to capture photos, video, or audio.
Media playback	Learn about how to play back audio and video in your apps.
Visual layer	The Visual layer provides a high performance, retained-mode API for graphics, effects and animations, and is the foundation for all UI across Windows devices.
XAML platform	Describes the XAML language and concepts to apps that use a WinRT XAML-based UI, including XAML UI components provided by Windows 10 and later as well as WinUI.

Win32 (C++ and COM) APIs

The following articles provide information about features available via Win32 (C++ and COM) APIs provided by the Windows SDK.

Expand table

Article	Description
Audio and video	Learn about features for integrating audio and video into your apps.
Graphics and gaming	Learn about features involving graphics, gaming, and imaging.
DirectX	Learn about how to use DirectX APIs to create games and other high-performance multimedia apps using 2D and 3D graphics.
Direct2D	Direct2D is a hardware-accelerated, immediate-mode, 2D graphics API that provides high performance and high-quality rendering for 2-D geometry, bitmaps, and text.
Direct3D	Direct3D is a low-level API for drawing primitives with the rendering pipeline, or for performing parallel operations with the compute shader.
GDI	The Windows Graphics Device Interface (GDI) is a low-level graphics API that enables apps to use two-dimensional vector graphics, imaging, and typography.
GDI+	GDI+ improves on GDI by adding new features and by optimizing existing features.

.NET features

The .NET SDK also provides APIs related to audio, video, and camera scenarios for WPF and Windows Forms apps.

Expand table

Article	Description
Graphics (WPF)	Learn about support for multimedia, vector graphics, animation, and content composition in WPF apps.
Multimedia (WPF)	Learn about integrating audio and video into your WPF apps.
Graphics and drawing (Windows Forms)	Learn how to create graphics, draw text, and manipulate graphical images in Windows Forms apps.
SoundPlayer class (Windows Forms)	Learn how to include sounds in your Windows Forms applications.

Camera

Article • 03/31/2025

This section provides guidance for creating WinUI 3 apps that use the camera or microphone to capture photos, video, or audio.

Basic MediaCapture tasks

Expand table

Topic	Description
Capture photos and video in a desktop app with the Windows built-in camera UI	Describes how to use the CameraCaptureUI class to capture photos or videos by using the camera UI built into Windows. This feature allows your app to get a user-captured photo or video with just a few lines of code.
Show the camera preview in a WinUI 3 app	Shows how to quickly display the camera preview stream within a XAML page in a WinUI 3 app.
Basic photo, video, and audio capture with MediaCapture	Shows the simplest way to capture photos and video using the **MediaCapture** class. The **MediaCapture** class exposes a robust set of APIs that provide low-level control over the capture pipeline and enable advanced capture scenarios, but this article is intended to help you add basic media capture to your app quickly and easily.

Advanced MediaCapture tasks

Expand table

Topic	Description
Discover and select camera capabilities with camera profiles	Learn how to use camera profiles to discover and manage the capabilities of different video capture devices. This includes tasks such as selecting profiles that support specific resolutions or frame rates, profiles that support simultaneous access to multiple cameras, and profiles that support HDR.
Set format, resolution, and frame rate for MediaCapture	Learn how to use the **IMediaEncodingProperties** interface to set the resolution and frame rate of the camera preview stream and captured photos and video. It also shows how to ensure that the aspect ratio of the preview stream matches that of the captured media.

Topic	Description
HDR and low-light photo capture	Learn how to use the **AdvancedPhotoCapture** class to capture High Dynamic Range (HDR) and low-light photos.
Manual camera controls for photo and video capture	Learn how to use manual device controls to enable enhanced photo and video capture scenarios including optical image stabilization and smooth zoom.
Manual camera controls for video capture	Learn how to use manual device controls to enable enhanced video capture scenarios including HDR video and exposure priority.
Video stabilization effect for video capture	Learn how to use the video stabilization effect.
Scene analysis for MediaCapture	Shows you how to use the **SceneAnalysisEffect** and the **FaceDetectionEffect** to analyze the content of the media capture preview stream.
Capture a photo sequence with VariablePhotoSequence	Learn how to capture a variable photo sequence, which allows you to capture multiple frames of images in rapid succession and configure each frame to use different focus, flash, ISO, exposure, and exposure compensation settings.
Process media frames with MediaFrameReader	Learn how to use a **MediaFrameReader** with **MediaCapture** to get media frames from one or more available sources, including color, depth, and infrared cameras, audio devices, or even custom frame sources such as those that produce skeletal tracking frames. This feature is designed to be used by apps that perform real-time processing of media frames, such as augmented reality and depth-aware camera apps.
Process audio frames with MediaFrameReader	Learn how to use a **MediaFrameReader** with **MediaCapture** to get audio data from a media frame source.
Detect and respond to audio level changes by the system	Learn how to detect and respond to audio level changes by the system.
Launch the camera settings page	Learn how to launch the Windows Settings app directly to the camera settings page.
Handle the Windows camera privacy setting	Learn how apps should handle the Windows camera privacy setting.
Connect to remote cameras	Learn how to connect to remote cameras and get a **MediaFrameSourceGroup** to retrieve frames from each camera.

WinUI 3 samples for camera

- MediaCapture WinUI 3 sample on github .

UWP app samples for camera

- Camera face detection sample
- Camera preview frame sample
- Camera HDR sample
- Camera manual controls sample
- Camera profile sample
- Camera resolution sample
- Camera starter kit
- Camera video stabilization sample

Related topics

- Audio, video, and camera

Capture photos and video in a desktop app with the Windows built-in camera UI

Article • 03/31/2025

This article describes how to use the CameraCaptureUI class to capture photos or videos by using the camera UI built into Windows. This feature allows your app to get a user-captured photo or video with just a few lines of code.

If you want to provide your own camera UI, or if your scenario requires more robust, low-level control of the capture operation, then you should use the MediaCapture class, and implement your own capture experience. For more information, see Basic photo, video, and audio capture with MediaCapture.

Note that the **CameraCaptureUI** class in the Microsoft.Windows.Media.Capture namespace is not supported for UWP apps. For information on using the UWP version of this feature, see Capture photos and video in a UWP app with the Windows built-in camera UI

Use the CameraCaptureUI class to capture photos

Create a new instance of **CameraCaptureUI**, passing in the AppWindow.Id property of your app window. The PhotoSettings property allows you to specify some constraints on the captured photo, including the file format and maximum resolution and whether UI allows the user to crop the photo after it's captured. The VideoSettings property provides similar properties for video capture, such as the maximum resolution and duration and whether the UI allows the user to trim the video after it's captured.

Call CaptureFileAsync to launch the camera capture UI asynchronously. Use one of the values from the CameraCaptureUIMode to specify whether the UI should allow photo capture, video capture, or both. When **CaptureFileAsync** completes, it will return a StorageFile file object containing the captured photo or video. If the returned object is null it means that either the user cancelled the capture operation or an error occurred.

The following example demonstrates launching the **CameraCaptureUI** for photo capture, specifying the image format as PNG and disabling cropping. In this example the captured photo is set as the source for an Image control.

C#

```
var cameraCaptureUI = new CameraCaptureUI(this.AppWindow.Id);
cameraCaptureUI.PhotoSettings.Format = CameraCaptureUIPhotoFormat.Png;
cameraCaptureUI.PhotoSettings.AllowCropping = false;

// Capture a photo asynchronously
StorageFile photo = await
cameraCaptureUI.CaptureFileAsync(CameraCaptureUIMode.Photo);

if (photo != null)
{
    // Photo capture was successful

    // Show the captured photo in a XAML Image control
    using (IRandomAccessStream fileStream = await
photo.OpenAsync(Windows.Storage.FileAccessMode.Read))
    {
        // Set the image source to the selected bitmap
        BitmapImage bitmapImage = new BitmapImage();
        await bitmapImage.SetSourceAsync(fileStream);
        iCapturedImage.Source = bitmapImage;
    }
} else
{
    // Photo capture failed or was cancelled
}
```

Use the CameraCaptureUI class to capture videos

The following example demonstrates launching the **CameraCaptureUI** for video capture, specifying the maximum video as standard definition and disabling trimming. In this example the captured photo is set as the source for an MediaPlayerElement control.

C#

```
var cameraCaptureUI = new CameraCaptureUI(this.AppWindow.Id);
cameraCaptureUI.VideoSettings.MaxResolution =
CameraCaptureUIMaxVideoResolution.StandardDefinition;
cameraCaptureUI.VideoSettings.AllowTrimming = true;
StorageFile videoFile = await
cameraCaptureUI.CaptureFileAsync(CameraCaptureUIMode.Video);

if (videoFile != null)
```

```
{
    // Video capture was successful

    // Show the captured video in a MediaPlayerElement control
    mediaPlayerElement.Source =
MediaSource.CreateFromStorageFile(videoFile);
    mediaPlayerElement.MediaPlayer.Play();
}
else
{
    // Video capture failed or was cancelled
}
```

Move and rename captured media files

The **CameraCaptureUI** creates randomized names for captured media files, so you may want to rename and move captured files to keep them organized. The following example moves and renames a captured file.

C#

```
StorageFile photo = await
cameraCaptureUI.CaptureFileAsync(CameraCaptureUIMode.Photo);

if (photo != null)
{
    // Move and rename captured photo
    StorageFolder destinationFolder =
    await
ApplicationData.Current.LocalFolder.CreateFolderAsync("ProfilePhotoFolde
r",
        CreationCollisionOption.OpenIfExists);

    await photo.CopyAsync(destinationFolder, "ProfilePhoto.jpg",
NameCollisionOption.ReplaceExisting);
    await photo.DeleteAsync();
}
```

Show the camera preview in a WinUI 3 app

Article • 01/16/2025

In this quickstart, you will learn how to create a basic WinUI 3 camera app that displays the camera preview. In a WinUI 3 app, you use the MediaPlayerElement control in the Microsot.UI.Xaml.Controls namespace to render the camera preview and the WinRT class MediaCapture to access the device's camera preview stream. **MediaCapture** provides APIs for performing a wide range of camera-related tasks such as such as capturing photos and videos and configuring the camera's device driver. See the other articles in this section for details about other **MediaCapture** features.

The code in this walkthrough is adapted from the MediaCapture WinUI 3 sample on github.

> **Tip**
>
> For the UWP version of this article, see **Display the camera preview** in the UWP documentation.

Prerequisites

- Your device must have developer mode enabled. For more information see Enable your device for development.
- Visual Studio 2022 or later with the **Windows application development** workload.

Create a new WinUI 3 app

In Visual Studio, create a new project. In the **Create a new project** dialog, set the language filter to "C#" and the platform filter to "Windows", then select the "Blank App, Packaged (WinUI 3 in desktop)" project template.

Create the UI

The simple UI for this example includes a **MediaPlayerElement** control for displaying the camera preview, a ComboBox that allows you to select from the device's cameras, and buttons for initializing the **MediaCapture** class, starting and stopping the camera

preview, and reseting the sample. We also include a TextBlock for displaying status messages.

In your project's MainWindow.xml file, replace the default **StackPanel** control with the following XAML.

XAML

```
<Grid ColumnDefinitions="4*,*" ColumnSpacing="4">
    <MediaPlayerElement x:Name="mpePreview" Grid.Row="0" Grid.Column="0"
AreTransportControlsEnabled="False" ManipulationMode="None"/>
    <StackPanel Orientation="Vertical"  Grid.Row="0" Grid.Column="1"
HorizontalAlignment="Stretch"  VerticalAlignment="Top">
        <TextBlock Text="Status:" Margin="0,0,10,0"/>
        <TextBlock x:Name="tbStatus" Text=""/>
        <TextBlock Text="Preview Source:" Margin="0,0,10,0"/>
        <ComboBox x:Name="cbDeviceList" HorizontalAlignment="Stretch"
SelectionChanged="cbDeviceList_SelectionChanged"></ComboBox>
        <Button x:Name="bStartMediaCapture" Content="Initialize
MediaCapture" IsEnabled="False" Click="bStartMediaCapture_Click"/>
        <Button x:Name="bStartPreview" Content="Start preview"
IsEnabled="False" Click="bStartPreview_Click"/>
        <Button x:Name="bStopPreview" Content="Stop preview"
IsEnabled="False" Click="bStopPreview_Click"/>
        <Button x:Name="bReset" Content="Reset" Click="bReset_Click" />
    </StackPanel>
</Grid>
```

Update the MainWindow class definition

The rest of the code in this article will be added to the **MainWindow** class definition in your project's MainWindow.xaml.cs file. First, add a few class variables that will persist throughout the lifetime of the window. These variables include:

- A DeviceInformationCollection that will store a DeviceInformation object for each available camera. The **DeviceInformation** object conveys information such as the unique identifier and the friendly name for the camera.
- A **MediaCapture** object that handles interactions with the selected camera's driver and allows you to retrieve the camera's video stream.
- A MediaFrameSource object that represents a source of media frames, such as a video stream.
- A boolean to track when the camera preview is running. Some camera settings can't be changed while the preview is running, so it's a good practice to track the state of the camera preview.

C#

```
private DeviceInformationCollection m_deviceList;
private MediaCapture m_mediaCapture;
private MediaFrameSource m_frameSource;
private MediaPlayer m_mediaPlayer;
private bool m_isPreviewing;
```

Populate the list of available cameras

Next we'll create a helper method to detect the cameras that are present on the current device and populate the **ComboBox** in the UI with the camera names, allowing the user to select a camera to preview.The DeviceInformation.FindAllAsync allows you to query for many different kinds of devices. We use MediaDevice.GetVideoCaptureSelector to retrieve the identifier that specifies that we only want to retrieve video capture devices.

C#

```
private async void PopulateCameraList()
{
    cbDeviceList.Items.Clear();

    m_deviceList = await
DeviceInformation.FindAllAsync(MediaDevice.GetVideoCaptureSelector());

    if(m_deviceList.Count == 0)
    {
        tbStatus.Text = "No video capture devices found.";
        return;
    }

    foreach (var device in m_deviceList)
    {
        cbDeviceList.Items.Add(device.Name);
        bStartMediaCapture.IsEnabled = true;
    }
}
```

Add a call to this helper method to the **MainWindow** class constructor so that the **ComboBox** gets populated when the window loads.

C#

```
public MainWindow()
{
    this.InitializeComponent();

    PopulateCameraList();
```

```
}
```

Initialize the MediaCapture object

Initialize the **MediaCapture** object by calling InitializeAsync, passing in a MediaCaptureInitializationSettings object containing the requested initialization parameters. There are a lot of optional initialization parameters that enable different scenarios. See the API reference page for the complete list. In this simple example we specify a few basic settings, including:

- The VideoDeviceId property specifies the unique identifier of the camera that the **MediaCapture** will attach to. We get the device ID from the **DeviceInformationCollection**, using the selected index of the **ComboBox**.
- The SharingMode property specifies whether the app is requesting shared, read-only access to the camera, which allows you to view and capture from the video stream, or exclusive control of the camera, which allows you to change the camera configuration. Multiple apps can read from a camera simultaneously, but only one app at a time can have exclusive control.
- The StreamingCaptureMode property specifies whether we want to capture video, audio, or audio and video.
- The MediaCaptureMemoryPreference allows us to request to specifically use CPU memory for video frames. The value **Auto** lets the system use GPU memory if it's available.

Before initializing the **MediaCapture** object we call AppCapability.CheckAccess method to determine if the user has denied our app access to the camera in Windows Settings.

> **Note**
>
> Windows allows users to grant or deny access to the device's camera in the Windows Settings app, under **Privacy & Security -> Camera**. When initializing the capture device, apps should check whether they have access to the camera and handle the case where access is denied by the user. For more information, see Handle the Windows camera privacy setting.

The **InitializeAsync** call is made from inside a **try** block so that we can recover if initialization fails. Apps should handle initialization failure gracefully. In this simple example, we'll just display an error message on failure.

```
C#
```

```
private async void bStartMediaCapture_Click(object sender, RoutedEventArgs
e)
{
    if (m_mediaCapture != null)
    {
        tbStatus.Text = "MediaCapture already initialized.";
        return;
    }

    // Supported in Windows Build 18362 and later
    if(AppCapability.Create("Webcam").CheckAccess() !=
AppCapabilityAccessStatus.Allowed)
    {
        tbStatus.Text = "Camera access denied. Launching settings.";

        bool result = await Windows.System.Launcher.LaunchUriAsync(new
Uri("ms-settings:privacy-webcam"));

        if (AppCapability.Create("Webcam").CheckAccess() !=
AppCapabilityAccessStatus.Allowed)
        {
            tbStatus.Text = "Camera access denied in privacy settings.";
            return;
        }
    }

    try
    {
        m_mediaCapture = new MediaCapture();
        var mediaCaptureInitializationSettings = new
MediaCaptureInitializationSettings()
        {
            VideoDeviceId = m_deviceList[cbDeviceList.SelectedIndex].Id,
            SharingMode = MediaCaptureSharingMode.ExclusiveControl,
            StreamingCaptureMode = StreamingCaptureMode.Video,
            MemoryPreference = MediaCaptureMemoryPreference.Auto
        };

        await
m_mediaCapture.InitializeAsync(mediaCaptureInitializationSettings);

        tbStatus.Text = "MediaCapture initialized successfully.";

        bStartPreview.IsEnabled = true;
    }
    catch (Exception ex)
    {
        tbStatus.Text = "Initialize media capture failed: " + ex.Message;
    }
}
```

Initialize the camera preview

When the user clicks the start preview button, we will attempt to create a **MediaFrameSource** for a video stream from the camera device with which the **MediaCapture** object was initialized. The available frame sources are exposed by the MediaCapture.FrameSources property.

To find a frame source that is color video data, as opposed to a depth camera for example, we look for a frame source that has a SourceKind of Color. Some camera drivers provide a dedicated preview stream that is separate from the record stream. To get the preview video stream, we try to select a frame source that has a MediaStreamType of VideoPreview. If no preview streams are found, we can get the record video stream by selecting a **MediaStreamType** of **VideoRecord**. If neither of these frame sources are available, then this capture device can't be used for video preview.

Once we have selected a frame source, we create a new MediaPlayer object that will be rendered by the **MediaPlayerElement** in our UI. We set the Source property of the **MediaPlayer** to a new MediaSource object we create from our selected **MediaFrameSource**.

Call Play on the **MediaPlayer** object to begin rendering the video stream.

C#

```
private void bStartPreview_Click(object sender, RoutedEventArgs e)
{

    m_frameSource = null;

    // Find preview source.
    // The preferred preview stream from a camera is defined by
MediaStreamType.VideoPreview on the RGB camera (SourceKind == color).
    var previewSource = m_mediaCapture.FrameSources.FirstOrDefault(source =>
source.Value.Info.MediaStreamType == MediaStreamType.VideoPreview

&& source.Value.Info.SourceKind == MediaFrameSourceKind.Color).Value;

    if (previewSource != null)
    {
        m_frameSource = previewSource;
    }
    else
    {
        var recordSource = m_mediaCapture.FrameSources.FirstOrDefault(source
=> source.Value.Info.MediaStreamType == MediaStreamType.VideoRecord

&& source.Value.Info.SourceKind == MediaFrameSourceKind.Color).Value;
```

```
        if (recordSource != null)
        {
            m_frameSource = recordSource;
        }
    }

    if (m_frameSource == null)
    {
        tbStatus.Text = "No video preview or record stream found.";
        return;
    }



    // Create MediaPlayer with the preview source
    m_mediaPlayer = new MediaPlayer();
    m_mediaPlayer.RealTimePlayback = true;
    m_mediaPlayer.AutoPlay = false;
    m_mediaPlayer.Source =
MediaSource.CreateFromMediaFrameSource(m_frameSource);
    m_mediaPlayer.MediaFailed += MediaPlayer_MediaFailed; ;

    // Set the mediaPlayer on the MediaPlayerElement
    mpePreview.SetMediaPlayer(m_mediaPlayer);

    // Start preview
    m_mediaPlayer.Play();


    tbStatus.Text = "Start preview succeeded!";
    m_isPreviewing = true;
    bStartPreview.IsEnabled = false;
    bStopPreview.IsEnabled = true;
}
```

Implement a handler for the MediaFailed event so that you can handle errors rendering the preview.

C#

```
private void MediaPlayer_MediaFailed(MediaPlayer sender,
MediaPlayerFailedEventArgs args)
{
    tbStatus.Text = "MediaPlayer error: " + args.ErrorMessage;
}
```

Stop the camera preview

To stop the camera preview, call Pause on the **MediaPlayer** object.

C#

```
private void bStopPreview_Click(object sender, RoutedEventArgs e)
{
    // Stop preview
    m_mediaPlayer.Pause();
    m_isPreviewing = false;
    bStartPreview.IsEnabled = true;
    bStopPreview.IsEnabled = false;
}
```

Reset the app

To make it easier to test out the sample app, add a method to reset the state of the app. Camera apps should always dispose of the camera and associated resources when the camera is no longer needed.

C#

```
private void bReset_Click(object sender, RoutedEventArgs e)
{
    if (m_mediaCapture != null)
    {
        m_mediaCapture.Dispose();
        m_mediaCapture = null;
    }

    if(m_mediaPlayer != null)
    {
        m_mediaPlayer.Dispose();
        m_mediaPlayer = null;
    }

    m_frameSource = null;


    bStartMediaCapture.IsEnabled = false;
    bStartPreview.IsEnabled = false;
    bStopPreview.IsEnabled = false;

    PopulateCameraList();

}
```

Basic photo, video, and audio capture with MediaCapture in a WinUI 3 app

Article • 01/16/2025

This article shows the simplest way to capture photos and video using the MediaCapture class. The **MediaCapture** class exposes a robust set of APIs that provide low-level control over the capture pipeline and enable advanced capture scenarios, but this article is intended to help you add basic media capture to your app quickly and easily. To learn about more of the features that **MediaCapture** provides, see Camera.

Initialize the MediaCapture object

All of the capture methods described in this article require the first step of initializing the MediaCapture object. This includes instantiating the object, selecting a capture device, setting initialization parameters, and then calling InitializeAsync. Typically camera apps will display the camera preview while capturing photos or video in their UI using the MediaPlayerElement. For a walkthrough of initializing **MediaCapture** and showing the preview in a XAML UI, see Show the camera preview in a WinUI 3 app. The code examples in this article will assume that an initialized instance of **MediaCapture** has already been created.

Capture a photo to a SoftwareBitmap

The SoftwareBitmap class provides a common representation of images across multiple features. If you want to capture a photo and then immediately use the captured image in your app, such as displaying it in XAML, instead of capturing to a file, then you should capture to a **SoftwareBitmap**. You still have the option of saving the image to disk later.

Capture a photo to a **SoftwareBitmap** using the LowLagPhotoCapture class. Get an instance of this class by calling PrepareLowLagPhotoCaptureAsync, passing in an ImageEncodingProperties object specifying the image format you want. CreateUncompressed creates an uncompressed encoding with the specified pixel format. Initiate photo capture by calling CaptureAsync, which returns a CapturedPhoto object. Get a **SoftwareBitmap** by accessing the Frame property and then the SoftwareBitmap property.

You can capture multiple photos by repeatedly calling **CaptureAsync**. When you are done capturing, call FinishAsync to shut down the **LowLagPhotoCapture** session and free up the associated resources. After calling **FinishAsync**, to begin capturing photos

again you will need to call PrepareLowLagPhotoCaptureAsync again to reinitialize the capture session before calling CaptureAsync.

```
C#

// Prepare and capture photo
var lowLagCapture = await
m_mediaCapture.PrepareLowLagPhotoCaptureAsync(ImageEncodingProperties.Create
Uncompressed(MediaPixelFormat.Bgra8));

var capturedPhoto = await lowLagCapture.CaptureAsync();
var softwareBitmap = capturedPhoto.Frame.SoftwareBitmap;

// Capture more photos, if desired.
// Then call FinishAsync to clean up resources
await lowLagCapture.FinishAsync();
```

Capture a photo to a memory stream

You can use **MediaCapture** to capture a photo to an in-memory stream, which you can then use to transcode the photo from the stream to a file on disk.

Create an InMemoryRandomAccessStream and then call CapturePhotoToStreamAsync to capture a photo to the stream, passing in the stream and an ImageEncodingProperties object specifying the image format that should be used. You can create custom encoding properties by initializing the object yourself, but the class provides static methods, like ImageEncodingProperties.CreateJpeg for common encoding formats.

Create a BitmapDecoder to decode the image from the in memory stream. Create a BitmapEncoder to encode the image to file by calling CreateForTranscodingAsync.

You can optionally create a BitmapPropertySet object and then call SetPropertiesAsync on the image encoder to include metadata about the photo in the image file. For more information about encoding properties, see Image metadata.

Finally, call FlushAsync on the encoder object to transcode the photo from the in-memory stream to the file.

```
C#

// Prepare and capture photo
var lowLagCapture = await
m_mediaCapture.PrepareLowLagPhotoCaptureAsync(ImageEncodingProperties.Create
Uncompressed(MediaPixelFormat.Bgra8));
```

```
var capturedPhoto = await lowLagCapture.CaptureAsync();
var softwareBitmap = capturedPhoto.Frame.SoftwareBitmap;

// Capture more photos, if desired.
// Then call FinishAsync to clean up resources
await lowLagCapture.FinishAsync();
```

Capture a video

Quickly add video capture to your app by using the **LowLagMediaRecording** class.

First, the **LowLagMediaRecording** needs to persist while video is being captured, so declare a class variable to for the object.

C#

```
LowLagMediaRecording m_mediaRecording;
```

Call **PrepareLowLagRecordToStorageFileAsync** to initialize the media recording, passing in the storage file and a **MediaEncodingProfile** object specifying the encoding for the video. The class provides static methods, like **CreateMp4**, for creating common video encoding profiles. Call **StartAsync** to begin capturing video.

C#

```
var myVideos = await
Windows.Storage.StorageLibrary.GetLibraryAsync(Windows.Storage.KnownLibraryI
d.Videos);
StorageFile file = await myVideos.SaveFolder.CreateFileAsync("video.mp4",
CreationCollisionOption.GenerateUniqueName);
m_mediaRecording = await
m_mediaCapture.PrepareLowLagRecordToStorageFileAsync(
        MediaEncodingProfile.CreateMp4(VideoEncodingQuality.Auto), file);

m_mediaCapture.RecordLimitationExceeded +=
m_mediaCapture_RecordLimitationExceeded;
await m_mediaRecording.StartAsync();
```

To stop recording video, call **StopAsync**.

C#

```
await m_mediaRecording.StopAsync();
```

You can continue to call **StartAsync** and **StopAsync** to capture additional videos. When you are done capturing videos, call **FinishAsync** to dispose of the capture session and clean up associated resources. After this call, you must call **PrepareLowLagRecordToStorageFileAsync** again to reinitialize the capture session before calling **StartAsync**.

```
C#

await m_mediaRecording.FinishAsync();
```

When capturing video, you should register a handler for the **RecordLimitationExceeded** event of the **MediaCapture** object, which will be raised by the operating system if you surpass the limit for a single recording, currently three hours. In the handler for the event, you should finalize your recording by calling **StopAsync**.

```
C#

private async void m_mediaCapture_RecordLimitationExceeded(MediaCapture
sender)
{
    await m_mediaRecording.StopAsync();
    DispatcherQueue.TryEnqueue(() =>
    {
        tbStatus.Text = "Record limitation exceeded.";
    });
}
```

You can pause a video recording and then resume recording without creating a separate output file by calling **PauseAsync** and then calling **ResumeAsync**.

```
C#

await
m_mediaRecording.PauseAsync(Windows.Media.Devices.MediaCapturePauseBehavior.
ReleaseHardwareResources);
```

```
C#

await m_mediaRecording.ResumeAsync();
```

Calling **PauseWithResultAsync** returns a **MediaCapturePauseResult** object. The **LastFrame** property is a **VideoFrame** object representing the last frame. To display the frame in XAML, get the **SoftwareBitmap** representation of the video frame. Currently, only images in BGRA8 format with premultiplied or empty alpha channel are supported,

so call **Convert** if necessary to get the correct format. **PauseWithResultAsync** also returns the duration of the video that was recorded in the preceding segment in case you need to track how much total time has been recorded.

You can also get a result frame when you stop the video by calling **StopWithResultAsync**.

Play and edit captured video files

Once you have captured a video to a file, you may want to load the file and play it back within your app's UI. You can do this using the **MediaPlayerElement** XAML control and an associated **MediaPlayer**. For information on playing media in a XAML page, see Play audio and video with MediaPlayer.

[TBD - Is the MediaComposition framework supported / recommended for WinUI?]

You can also create a **MediaClip** object from a video file by calling **CreateFromFileAsync**. A **MediaComposition** provides basic video editing functionality like arranging the sequence of **MediaClip** objects, trimming video length, creating layers, adding background music, and applying video effects. For more information on working with media compositions, see Media compositions and editing.

Capture audio

You can quickly add audio capture to your app by using the same technique shown above for capturing video. Call **PrepareLowLagRecordToStorageFileAsync** to initialize the capture session, passing in the file and a **MediaEncodingProfile** which is generated in this example by the **CreateMp3** static method. To begin recording, call **StartAsync**.

[TBD - This code is throwing 'The request is invalid in the current state.']

C#

```
m_mediaCapture.RecordLimitationExceeded +=
m_mediaCapture_RecordLimitationExceeded;

var myVideos = await
Windows.Storage.StorageLibrary.GetLibraryAsync(Windows.Storage.KnownLibraryI
d.Videos);
StorageFile file = await myVideos.SaveFolder.CreateFileAsync("audio.mp3",
CreationCollisionOption.GenerateUniqueName);
m_mediaRecording = await
m_mediaCapture.PrepareLowLagRecordToStorageFileAsync(
```

```
        MediaEncodingProfile.CreateMp3(AudioEncodingQuality.High), file);
await m_mediaRecording.StartAsync();
```

Call **StopAsync** to stop the audio recording.

```
C#

await m_mediaRecording.StopAsync();
```

You can call **StartAsync** and **StopAsync** multiple times to record several audio files. When you are done capturing audio, call **FinishAsync** to dispose of the capture session and clean up associated resources. After this call, you must call **PrepareLowLagRecordToStorageFileAsync** again to reinitialize the capture session before calling **StartAsync**.

For information about detecting when the system changes the audio level of the audio capture stream, see Detect and respond to audio level changes by the system.

Discover and select camera capabilities with camera profiles in a WinUI 3 app

Article • 01/16/2025

This article shows how to use camera profiles to discover and manage the capabilities of different video capture devices. This includes tasks such as selecting profiles that support specific resolutions or frame rates, profiles that support simultaneous access to multiple cameras, and profiles that support HDR.

About camera profiles

Cameras on different devices support different capabilities including the set of supported capture resolutions, frame rate for video captures, and whether HDR or variable frame rate captures are supported. A set of supported capabilities is defined in a **MediaCaptureVideoProfileMediaDescription** object. A camera profile, represented by a **MediaCaptureVideoProfile** object, has three collections of media descriptions; one for photo capture, one for video capture, and another for video preview.

Before initializing your **MediaCapture** object, you can query the capture devices on the current device to see what profiles are supported. When you select a supported profile, you know that the capture device supports all of the capabilities in the profile's media descriptions. This eliminates the need for a trial and error approach to determining which combinations of capabilities are supported on a particular device.

Find a camera that supports camera profiles

To use camera profiles, you must first check for a camera device that supports the use of camera profiles. The example below shows how to use the **DeviceInformation.FindAllAsync** method to retrieve a list of all available video capture devices on either the front or back panel of the current device. It loops through all of the devices in the list, calling the static method, **IsVideoProfileSupported**, for each device to see if it supports video profiles.

If a device that supports camera profiles is found on the specified panel, the **Id** value, containing the device's ID string, is returned.

XAML

```
public async Task<string>
GetVideoProfileSupportedDeviceIdAsync(Windows.Devices.Enumeration.Panel
```

```
panel)
{
    string deviceId = string.Empty;

    // Finds all video capture devices
    DeviceInformationCollection devices = await
DeviceInformation.FindAllAsync(DeviceClass.VideoCapture);

    foreach (var device in devices)
    {
        // Check if the device on the requested panel supports Video Profile
        if (MediaCapture.IsVideoProfileSupported(device.Id) &&
device.EnclosureLocation.Panel == panel)
        {
            // We've located a device that supports Video Profiles on
expected panel
            deviceId = device.Id;
            break;
        }
    }

    return deviceId;
}
```

If the device ID returned from the **GetVideoProfileSupportedDeviceIdAsync** helper method is null or an empty string, there is no device on the specified panel that supports camera profiles. In this case, you should initialize your media capture device without using profiles.

XAML

```
string videoDeviceId = await
GetVideoProfileSupportedDeviceIdAsync(Windows.Devices.Enumeration.Panel.Back
);

if (string.IsNullOrEmpty(videoDeviceId))
{
    // No devices on the specified panel support video profiles. .
    return;
}
```

Select a profile based on supported resolution and frame rate

To select a profile with particular capabilities, such as with the ability to achieve a particular resolution and frame rate, use the method shown previously in this article to get the ID of a capture device that supports using camera profiles.

Create a new MediaCaptureInitializationSettings object, passing in the selected device ID. Call the static method MediaCapture.FindAllVideoProfiles to get a list of all camera profiles supported by the device.

This example selects a profile that contains a SupportedRecordMediaDescription object where the Width, Height, and FrameRate properties match the requested values. If a match is found, VideoProfile and RecordMediaDescription of the **MediaCaptureInitializationSettings** are set to the values returned from the query. If no match is found, the default profile is used.

XAML

```
var mediaInitSettings = new MediaCaptureInitializationSettings {
VideoDeviceId = videoDeviceId };

IReadOnlyList<MediaCaptureVideoProfile> profiles =
MediaCapture.FindAllVideoProfiles(videoDeviceId);

var match = (from profile in profiles
             from desc in profile.SupportedRecordMediaDescription
             where desc.Width == 640 && desc.Height == 480 &&
Math.Round(desc.FrameRate) == 30
             select new { profile, desc }).FirstOrDefault();

if (match != null)
{
    mediaInitSettings.VideoProfile = match.profile;
    mediaInitSettings.RecordMediaDescription = match.desc;
}
else
{
    // Could not locate a WVGA 30FPS profile, use default video recording
profile
    mediaInitSettings.VideoProfile = profiles[0];
}
```

After you populate the **MediaCaptureInitializationSettings** with your desired camera profile, you simply call InitializeAsync on your media capture object to configure it to the desired profile.

XAML

```
await m_mediaCapture.InitializeAsync(mediaInitSettings);
```

Select devices with KnownVideoProfile

You can use the **MediaFrameSourceGroup** class to get camera profiles with specific capabilities before initializing the **MediaCapture** object. Frame source groups allow device manufacturers to represent groups of sensors or capture capabilities as a single virtual device. This enables computational photography scenarios such as using depth and color cameras together, but can also be used to select camera profiles for simple capture scenarios. For more information on using **MediaFrameSourceGroup**, see Process media frames with MediaFrameReader.

The example below shows how to use **MediaFrameSourceGroup** to find a camera profile that supports that supports the desired scenario. Call **MediaFrameSourceGroup.FindAllAsync** to get a list of all media frame source groups available on the current device. Loop through each source group and call **MediaCapture.FindKnownVideoProfiles** to get a list of all of the video profiles for the current source group that support the specified KnownCameraProfile, in this example the value **KnownVideoProfile.HighQualityPhoto** is used. Other values include support for HDR and variable photo sequences, for example. If a profile that meets the requested criteria is found, create a new **MediaCaptureInitializationSettings** object and set the **VideoProfile** to the select profile and the **VideoDeviceId** to the **Id** property of the current media frame source group. Use this **MediaCaptureInitializationSettings** object to initialize the **MediaCapture** object.

XAML

```
IReadOnlyList<MediaFrameSourceGroup> sourceGroups = await
MediaFrameSourceGroup.FindAllAsync();
MediaCaptureInitializationSettings settings = null;

foreach (MediaFrameSourceGroup sg in sourceGroups)
{

    IReadOnlyList<MediaCaptureVideoProfile> profileList =
MediaCapture.FindKnownVideoProfiles(
                                    sg.Id,
                                    KnownVideoProfile.HighQualityPhoto);

    if (profileList.Count > 0)
    {
        settings = new MediaCaptureInitializationSettings();
        settings.VideoProfile = profileList[0];
        settings.VideoDeviceId = sg.Id;
        break;
    }
}
```

Determine if a device supports simultaneous photo and video capture

Many devices support capturing photos and video simultaneously. To determine if a capture device supports this, call **MediaCapture.FindAllVideoProfiles** to get all of the camera profiles supported by the device. Use a link query to find a profile that has at least one entry for both **SupportedPhotoMediaDescription** and **SupportedRecordMediaDescription** which means that the profile supports simultaneous capture.

XAML

```
bool simultaneousPhotoAndVideoSupported = false;

IReadOnlyList<MediaCaptureVideoProfile> profiles =
MediaCapture.FindAllVideoProfiles(videoDeviceId);

var match = (from profile in profiles
             where profile.SupportedPhotoMediaDescription.Any() &&
             profile.SupportedRecordMediaDescription.Any()
             select profile).FirstOrDefault();

if (match != null)
{
    // Simultaneous photo and video supported
    simultaneousPhotoAndVideoSupported = true;
}
else
{
    // Simultaneous photo and video not supported
    simultaneousPhotoAndVideoSupported = false;
}
```

You can refine this query to look for profiles that support specific resolutions or other capabilities in addition to simultaneous video record. You can also use the **MediaCapture.FindKnownVideoProfiles** and specify the **BalancedVideoAndPhoto** value to retrieve profiles that support simultaneous capture, but querying all profiles will provide more complete results.

Related topics

- Camera
- Basic photo, video, and audio capture with MediaCapture

Set format, resolution, and frame rate for MediaCapture in a WinUI 3 app

Article • 01/16/2025

This article shows you how to use the **IMediaEncodingProperties** interface to set the resolution and frame rate of the camera preview stream and captured photos and video. It also shows how to ensure that the aspect ratio of the preview stream matches that of the captured media.

Camera profiles offer a simpler, higher level mechanism for discovering and setting the stream properties of the camera, but they are not supported for all devices. For more information, see Camera profiles.

Determine if the preview and capture streams are independent

On some devices, the same hardware pin is used for both preview and capture streams. On these devices, setting the encoding properties such as the format, resolution and frame rate on one will affect both. On devices that use different hardware pins for capture and preview, the properties can be set for each stream independently. Use the following code to determine if the preview and capture streams are independent. This example sets a boolean global variable that can be used to switch the app's behavior if the streams are shared or independent.

C#

```
if (m_mediaCapture.MediaCaptureSettings.VideoDeviceCharacteristic ==
VideoDeviceCharacteristic.AllStreamsIdentical ||
    m_mediaCapture.MediaCaptureSettings.VideoDeviceCharacteristic ==
VideoDeviceCharacteristic.PreviewRecordStreamsIdentical)
{
    m_captureAndPreviewStreamsIdentical = true;
}
```

A media encoding properties helper class

Creating a simple helper class to wrap the functionality of the **IMediaEncodingProperties** interface makes it easier to select a set of encoding properties that meet particular criteria. This helper class is particularly useful due to the following behavior of the encoding properties feature:

> **Note**
>
> The VideoDeviceController.GetAvailableMediaStreamProperties method takes a member of the MediaStreamType enumeration, such as **VideoRecord** or **Photo**, and returns a list of either ImageEncodingProperties or VideoEncodingProperties objects that convey the stream encoding settings, such as the resolution of the captured photo or video. The results of calling **GetAvailableMediaStreamProperties** may include **ImageEncodingProperties** or **VideoEncodingProperties** regardless of what **MediaStreamType** value is specified. For this reason, you should always check the type of each returned value and cast it to the appropriate type before attempting to access any of the property values.

The helper class defined below handles the type checking and casting for ImageEncodingProperties or VideoEncodingProperties so that your app code doesn't need to distinguish between the two types. In addition to this, the helper class exposes properties for the aspect ratio of the properties, the frame rate (for video encoding properties only), and a friendly name that makes it easier to display the encoding properties in your UI.

C#

```
class StreamPropertiesHelper
{
    private IMediaEncodingProperties _properties;

    public StreamPropertiesHelper(IMediaEncodingProperties properties)
    {
        if (properties == null)
        {
            throw new ArgumentNullException(nameof(properties));
        }

        // This helper class only uses ImageEncodingProperties or
VideoEncodingProperties
        if (!(properties is ImageEncodingProperties) && !(properties is
VideoEncodingProperties))
        {
            throw new ArgumentException("Argument is of the wrong type.
Required: " + typeof(ImageEncodingProperties).Name
                + " or " + typeof(VideoEncodingProperties).Name + ".",
nameof(properties));
        }

        // Store the actual instance of the IMediaEncodingProperties for
setting them later
        _properties = properties;
    }
```

```
    public uint Width
    {
        get
        {
            if (_properties is ImageEncodingProperties)
            {
                return (_properties as ImageEncodingProperties).Width;
            }
            else if (_properties is VideoEncodingProperties)
            {
                return (_properties as VideoEncodingProperties).Width;
            }

            return 0;
        }
    }

    public uint Height
    {
        get
        {
            if (_properties is ImageEncodingProperties)
            {
                return (_properties as ImageEncodingProperties).Height;
            }
            else if (_properties is VideoEncodingProperties)
            {
                return (_properties as VideoEncodingProperties).Height;
            }

            return 0;
        }
    }

    public uint FrameRate
    {
        get
        {
            if (_properties is VideoEncodingProperties)
            {
                if ((_properties as
VideoEncodingProperties).FrameRate.Denominator != 0)
                {
                    return (_properties as
VideoEncodingProperties).FrameRate.Numerator /
                           (_properties as
VideoEncodingProperties).FrameRate.Denominator;
                }
            }

            return 0;
        }
    }
```

```
    public double AspectRatio
    {
        get { return Math.Round((Height != 0) ? (Width / (double)Height) :
double.NaN, 2); }
    }

    public IMediaEncodingProperties EncodingProperties
    {
        get { return _properties; }
    }

    public string GetFriendlyName(bool showFrameRate = true)
    {
        if (_properties is ImageEncodingProperties ||
            !showFrameRate)
        {
            return Width + "x" + Height + " [" + AspectRatio + "] " +
_properties.Subtype;
        }
        else if (_properties is VideoEncodingProperties)
        {
            return Width + "x" + Height + " [" + AspectRatio + "] " +
FrameRate + "FPS " + _properties.Subtype;
        }

        return String.Empty;
    }

}
```

Get a list of available stream properties

Get a list of the available stream properties for a capture device by getting the VideoDeviceController for your app's MediaCapture object and then calling GetAvailableMediaStreamProperties and passing in one of the MediaStreamType values, **VideoPreview**, **VideoRecord**, or **Photo**. In this example, a list of **StreamPropertiesHelper** objects, defined previously in this article, is created for each of the IMediaEncodingProperties values returned from **GetAvailableMediaStreamProperties**. This example orders the returned properties based first on resolution and then on frame rate.

If your app has specific resolution or frame rate requirements, you can select a set of media encoding properties programmatically. A typical camera app will instead expose the list of available properties in the UI and allow the user to select their desired settings. A **ComboBoxItem** is created for each item in the list of **StreamPropertiesHelper** objects in the list. The content is set to the friendly name returned by the helper class and the tag is set to the helper class itself so it can be used

later to retrieve the associated encoding properties. Each **ComboBoxItem** is then added to a **ComboBox** defined in the UI.

C#

```
private void bGetStreamProperties_Click(object sender, RoutedEventArgs e)
{
    // Query all properties of the specified stream type
    IEnumerable<StreamPropertiesHelper> allStreamProperties =

m_mediaCapture.VideoDeviceController.GetAvailableMediaStreamProperties(Media
StreamType.VideoRecord).Select(x => new StreamPropertiesHelper(x));

    // Order them by resolution then frame rate
    allStreamProperties = allStreamProperties.OrderByDescending(x =>
x.Height * x.Width).ThenByDescending(x => x.FrameRate);

    // Populate the combo box with the entries
    foreach (var property in allStreamProperties)
    {
        ComboBoxItem comboBoxItem = new ComboBoxItem();
        comboBoxItem.Content = property.GetFriendlyName();
        comboBoxItem.Tag = property;
        cbStreamProperties.Items.Add(comboBoxItem);
    }
}
```

Set the desired stream properties

Tell the video device controller to use your desired encoding properties by calling SetMediaStreamPropertiesAsync, passing in the **MediaStreamType** value indicating whether the photo, video, or preview properties should be set. This example uses the **ComboBox** populated in the example from the previous section, where the media stream properties are retrieved from the tag property from the selected item.

C#

```
private async void bSetStreamProperties_Click(object sender, RoutedEventArgs
e)
{

    if (m_exclusiveCameraAccess)
    {
        var selectedItem = cbStreamProperties.SelectedItem as ComboBoxItem;
        var encodingProperties = (selectedItem.Tag as
StreamPropertiesHelper).EncodingProperties;
        await
m_mediaCapture.VideoDeviceController.SetMediaStreamPropertiesAsync(MediaStre
amType.VideoPreview, encodingProperties);
```

```
    }
}
```

Note that your app must have exclusive control of the capture device in order to change the media stream properties.

Match the aspect ratio of the preview and capture streams

A typical camera app will provide UI for the user to select the video or photo capture resolution but will programmatically set the preview resolution. There are a few different strategies for selecting the best preview stream resolution for your app:

- Select the highest available preview resolution, letting the UI framework perform any necessary scaling of the preview.
- Select the preview resolution closest to the capture resolution so that the preview displays the closest representation to the final captured media.
- Select the preview resolution closest to the size of the **CaptureElement** so that no more pixels than necessary are going through the preview stream pipeline.

> **Note**
>
> It is possible, on some devices, to set a different aspect ratio for the camera's preview stream and capture stream. Frame cropping caused by this mismatch can result in content being present in the captured media that was not visible in the preview which can result in a negative user experience. It is strongly recommended that you use the same aspect ratio, within a small tolerance window, for the preview and capture streams. It is fine to have entirely different resolutions enabled for capture and preview as long as the aspect ratio match closely.

To ensure that the photo or video capture streams match the aspect ratio of the preview stream, this example calls **VideoDeviceController.GetMediaStreamProperties** and passes in the **VideoPreview** enum value to request the current stream properties for the preview stream. Next a small aspect ratio tolerance window is defined so that we can include aspect ratios that are not exactly the same as the preview stream, as long as they are close. Next, the **StreamPropertiesHelper** objects where the aspect ratio is within the defined tolerance range of the preview stream are selected.

```
C#
```

```
// Query all properties of the specified stream type
IEnumerable<StreamPropertiesHelper> allVideoProperties =

m_mediaCapture.VideoDeviceController.GetAvailableMediaStreamProperties(Media
StreamType.VideoRecord).Select(x => new StreamPropertiesHelper(x));

// Query the current preview settings
StreamPropertiesHelper previewProperties = new
StreamPropertiesHelper(m_mediaCapture.VideoDeviceController.GetMediaStreamPr
operties(MediaStreamType.VideoPreview));

// Get all formats that have the same-ish aspect ratio as the preview
// Allow for some tolerance in the aspect ratio comparison
const double ASPECT_RATIO_TOLERANCE = 0.015;
var matchingFormats = allVideoProperties.Where(x => Math.Abs(x.AspectRatio -
previewProperties.AspectRatio) < ASPECT_RATIO_TOLERANCE);

// Order them by resolution then frame rate
allVideoProperties = matchingFormats.OrderByDescending(x => x.Height *
x.Width).ThenByDescending(x => x.FrameRate);

// Clear out old entries and populate the video combo box with new matching
entries
cbStreamProperties.Items.Clear();
foreach (var property in allVideoProperties)
{
    ComboBoxItem comboBoxItem = new ComboBoxItem();
    comboBoxItem.Content = property.GetFriendlyName();
    comboBoxItem.Tag = property;
    cbStreamProperties.Items.Add(comboBoxItem);
}
```

SnippetMatchPreviewAspectRatio

High dynamic range (HDR) and low-light photo capture

Article • 01/16/2025

This article shows you how to use the **AdvancedPhotoCapture** class to capture high dynamic range (HDR) photos. This API also allows you to obtain a reference frame from the HDR capture before the processing of the final image is complete.

Other articles related to HDR capture include:

- You can use the **SceneAnalysisEffect** to allow the system to evaluate the content of the media capture preview stream to determine if HDR processing would improve the capture result. For more information, see Scene analysis for MediaCapture.
- Use the **HdrVideoControl** to capture video using the Windows built-in HDR processing algorithm. For more information, see Capture device controls for video capture.
- You can use the **VariablePhotoSequenceCapture** to capture a sequence of photos, each with different capture settings, and implement your own HDR or other processing algorithm. For more information, see Variable photo sequence.

> **Note**
>
> Starting with Windows 10, version 1709, recording video and using **AdvancedPhotoCapture** concurrently is supported. This is not supported in previous versions. This change means that you can have a prepared LowLagMediaRecording and AdvancedPhotoCapture at the same time. You can start or stop video recording between calls to MediaCapture.PrepareAdvancedPhotoCaptureAsync and AdvancedPhotoCapture.FinishAsync. You can also call AdvancedPhotoCapture.CaptureAsync while video is recording. However, some **AdvancedPhotoCapture** scenarios, like capturing an HDR photo while recording video would cause some video frames to be altered by the HDR capture, resulting in a negative user experience. For this reason, the list of modes returned by the AdvancedPhotoControl.SupportedModes will be different while video is recording. You should check this value immediately after starting or stopping video recording to ensure that the desired mode is supported in the current video recording state.

> **Note**
>
> Starting with Windows 10, version 1709, when the **AdvancedPhotoCapture** is set to HDR mode, the setting of the FlashControl.Enabled property is ignored and the flash is never fired. For other capture modes, if the **FlashControl.Enabled**, it will override the **AdvancedPhotoCapture** settings and cause a normal photo to be captured with flash. If Auto is set to true, the **AdvancedPhotoCapture** may or may not use flash, depending on the camera driver's default behavior for the conditions in the current scene. On previous releases, the **AdvancedPhotoCapture** flash setting always overrides the **FlashControl.Enabled** setting.

There is a full sample demonstrating the use of the **AdvancedPhotoCapture** class that you can use to see the API used in context or as a starting point for your own app. For more information see, Camera Advanced Capture sample.

HDR photo capture

Determine if HDR photo capture is supported on the current device

The HDR capture technique described in this article is performed using the AdvancedPhotoCapture object. Not all devices support HDR capture with **AdvancedPhotoCapture**. Determine if the device on which your app is currently running supports the technique by getting the **MediaCapture** object's VideoDeviceController and then getting the AdvancedPhotoControl property. Check the video device controller's SupportedModes collection to see if it includes AdvancedPhotoMode.Hdr. If it does, HDR capture using **AdvancedPhotoCapture** is supported.

C#

```
m_hdrSupported =
m_mediaCapture.VideoDeviceController.AdvancedPhotoControl.SupportedModes.Con
tains(Windows.Media.Devices.AdvancedPhotoMode.Hdr);
```

Configure and prepare the AdvancedPhotoCapture object

Because you will need to access the AdvancedPhotoCapture instance from multiple places within your code, you should declare a member variable to hold the object.

C#

```
private AdvancedPhotoCapture m_advancedCapture;
```

In your app, after you have initialized the **MediaCapture** object, create an AdvancedPhotoCaptureSettings object and set the mode to AdvancedPhotoMode.Hdr. Call the AdvancedPhotoControl object's Configure method, passing in the **AdvancedPhotoCaptureSettings** object you created.

Call the **MediaCapture** object's PrepareAdvancedPhotoCaptureAsync, passing in an ImageEncodingProperties object specifying the type of encoding the capture should use. The **ImageEncodingProperties** class provides static methods for creating the image encodings that are supported by **MediaCapture**.

PrepareAdvancedPhotoCaptureAsync returns the AdvancedPhotoCapture object you will use to initiate photo capture. You can use this object to register handlers for the OptionalReferencePhotoCaptured and AllPhotosCaptured which are discussed later in this article.

C#

```
if (m_hdrSupported == false) return;

// Choose HDR mode
var settings = new AdvancedPhotoCaptureSettings { Mode =
AdvancedPhotoMode.Hdr };

// Configure the mode
m_mediaCapture.VideoDeviceController.AdvancedPhotoControl.Configure(settings
);

// Prepare for an advanced capture
m_advancedCapture =
    await
m_mediaCapture.PrepareAdvancedPhotoCaptureAsync(ImageEncodingProperties.Crea
teUncompressed(MediaPixelFormat.Nv12));

// Register for events published by the AdvancedCapture
m_advancedCapture.AllPhotosCaptured += M_advancedCapture_AllPhotosCaptured;
m_advancedCapture.OptionalReferencePhotoCaptured +=
M_advancedCapture_OptionalReferencePhotoCaptured;
```

Capture an HDR photo

Capture an HDR photo by calling the AdvancedPhotoCapture object's CaptureAsync method. This method returns an AdvancedCapturedPhoto object that provides the captured photo in its Frame property. Next, the photo is saved to disk.

C#

```
try
{

    // Start capture, and pass the context object
    AdvancedCapturedPhoto advancedCapturedPhoto = await
m_advancedCapture.CaptureAsync();

    using (var frame = advancedCapturedPhoto.Frame)
    {
        var fileName = String.Format("SimplePhoto_{0}_HDR.jpg",
DateTime.Now.ToString("HHmmss"));
        StorageFile photoFile = await
KnownFolders.PicturesLibrary.CreateFileAsync(fileName);
        IRandomAccessStream stream = await
photoFile.OpenAsync(FileAccessMode.ReadWrite);
        await
RandomAccessStream.CopyAndCloseAsync(advancedCapturedPhoto.Frame, stream);
    }
}
catch (Exception ex)
{
    Debug.WriteLine("Exception when taking an HDR photo: {0}",
ex.ToString());
}
```

Get optional reference frame

The HDR process captures multiple frames and then composites them into a single image after all of the frames have been captured. You can get access to a frame after it is captured but before the entire HDR process is complete by handling the OptionalReferencePhotoCaptured event. You don't need to do this if you are only interested in the final HDR photo result.

> ⓘ **Important**
>
> OptionalReferencePhotoCaptured is not raised on devices that support hardware HDR and therefore do not generate reference frames. Your app should handle the case where this event is not raised.

Because the reference frame arrives out of context of the call to **CaptureAsync**, a mechanism is provided to pass context information to the **OptionalReferencePhotoCaptured** handler. First you should call an object that will contain your context information. The name and contents of this object is up to you.

This example defines an object that has members to track the file name and camera orientation of the capture.

C#

```
public class MyAdvancedCaptureContextObject
{
    public string CaptureFileName;
    public PhotoOrientation CaptureOrientation;
}
```

Create a new instance of your context object, populate its members, and then pass it into the overload of **CaptureAsync** that accepts an object as a parameter.

C#

```
// Add the the capture time to the file name
var fileName = String.Format("SimplePhoto_{0}_HDR.jpg",
DateTime.Now.ToString("HHmmss"));

// Create a context object, to identify the capture in the
OptionalReferencePhotoCaptured event
var context = new MyAdvancedCaptureContextObject()
{
    CaptureFileName = fileName,
};

// Start capture, and pass the context object
AdvancedCapturedPhoto advancedCapturedPhoto = await
m_advancedCapture.CaptureAsync(context);
```

In the **OptionalReferencePhotoCaptured** event handler, cast the **Context** property of the **OptionalReferencePhotoCapturedEventArgs** object to your context object class. This example modifies the file name to distinguish the reference frame image from the final HDR image and then saves the image to disk.

C#

```
private async void
M_advancedCapture_OptionalReferencePhotoCaptured(AdvancedPhotoCapture
sender, OptionalReferencePhotoCapturedEventArgs args)
{
    // Retrieve the context (i.e. what capture does this belong to?)
    var context = args.Context as MyAdvancedCaptureContextObject;

    // Remove "_HDR" from the name of the capture to create the name of the
reference
    var referenceName = context.CaptureFileName.Replace("_HDR", "");
```

```
    using (var frame = args.Frame)
    {
        await SaveCapturedFrameAsync(frame, referenceName,
context.CaptureOrientation);
    }
}
```

Receive a notification when all frames have been captured

The HDR photo capture has two steps. First, multiple frames are captured, and then the frames are processed into the final HDR image. You can't initiate another capture while the source HDR frames are still being captured, but you can initiate a capture after all of the frames have been captured but before the HDR post-processing is complete. The **AllPhotosCaptured** event is raised when the HDR captures are complete, letting you know that you can initiate another capture. A typical scenario is to disable your UI's capture button when HDR capture begins and then reenable it when **AllPhotosCaptured** is raised.

C#

```
private void M_advancedCapture_AllPhotosCaptured(AdvancedPhotoCapture
sender, object args)
{
    // Update UI to enable capture button
}
```

Clean up the AdvancedPhotoCapture object

When your app is done capturing, before disposing of the **MediaCapture** object, you should shut down the **AdvancedPhotoCapture** object by calling **FinishAsync** and setting your member variable to null.

C#

```
await m_advancedCapture.FinishAsync();
m_advancedCapture = null;
```

Low-light photo capture

Starting with Windows 10, version 1607, **AdvancedPhotoCapture** can be used to capture photos using a built-in algorithm that enhances the quality of photos captured in low-

light settings. When you use the low-light feature of the AdvancedPhotoCapture class, the system will evaluate the current scene and, if needed, apply an algorithm to compensate for low-light conditions. If the system determines that the algorithm is not needed, a regular capture is performed instead.

Before using low-light photo capture, determine if the device on which your app is currently running supports the technique by getting the **MediaCapture** object's VideoDeviceController and then getting the AdvancedPhotoControl property. Check the video device controller's SupportedModes collection to see if it includes AdvancedPhotoMode.LowLight. If it does, low-light capture using **AdvancedPhotoCapture** is supported.

C#

```
m_lowLightSupported =
m_mediaCapture.VideoDeviceController.AdvancedPhotoControl.SupportedModes.Con
tains(Windows.Media.Devices.AdvancedPhotoMode.LowLight);
```

In your app, after you have initialized the **MediaCapture** object, create an AdvancedPhotoCaptureSettings object and set the mode to AdvancedPhotoMode.LowLight. Call the AdvancedPhotoControl object's Configure method, passing in the **AdvancedPhotoCaptureSettings** object you created.

Call the **MediaCapture** object's PrepareAdvancedPhotoCaptureAsync, passing in an ImageEncodingProperties object specifying the type of encoding the capture should use.

C#

```
if (m_lowLightSupported == false) return;

// Choose LowLight mode
var settings = new AdvancedPhotoCaptureSettings { Mode =
AdvancedPhotoMode.LowLight };
m_mediaCapture.VideoDeviceController.AdvancedPhotoControl.Configure(settings
);

// Prepare for an advanced capture
m_advancedCapture =
    await
m_mediaCapture.PrepareAdvancedPhotoCaptureAsync(ImageEncodingProperties.Crea
teJpeg());
```

To capture a photo, call CaptureAsync.

C#

```
AdvancedCapturedPhoto advancedCapturedPhoto = await
m_advancedCapture.CaptureAsync();
StorageFile photoFile = await
KnownFolders.PicturesLibrary.CreateFileAsync($"Photo_{advancedCapturedPhoto.
Mode}.jpg", CreationCollisionOption.GenerateUniqueName);
IRandomAccessStream stream = await
photoFile.OpenAsync(FileAccessMode.ReadWrite);
await RandomAccessStream.CopyAndCloseAsync(advancedCapturedPhoto.Frame,
stream);
```

You can capture multiple low-light photos without reconfiguring the **AdvancedPhotoCapture** object, but when you are done capturing, you should call FinishAsync to clean up the object and associated resources.

C#

```
await m_advancedCapture.FinishAsync();
m_advancedCapture = null;
```

Working with AdvancedCapturedPhoto objects

AdvancedPhotoCapture.CaptureAsync returns an AdvancedCapturedPhoto object representing the captured photo. This object exposes the Frame property which returns a CapturedFrame object representing the image. The OptionalReferencePhotoCaptured event also provides a **CapturedFrame** object in its event args. After you get an object of this type, there are a number of things you can do with it, including creating a SoftwareBitmap or saving the image to a file.

Get a SoftwareBitmap from a CapturedFrame

Get a **SoftwareBitmap** from a **CapturedFrame** object by simply accessing the SoftwareBitmap property of the object. However, most encoding formats do not support **SoftwareBitmap** with **AdvancedPhotoCapture**, so you should check and make sure the property is not null before using it.

C#

```
SoftwareBitmap bitmap;
if (advancedCapturedPhoto.Frame.SoftwareBitmap != null)
{
    bitmap = advancedCapturedPhoto.Frame.SoftwareBitmap;
}
```

In the current release, the only encoding format that supports **SoftwareBitmap** for **AdvancedPhotoCapture** is uncompressed NV12. So, if you want to use this feature, you must specify that encoding when you call **PrepareAdvancedPhotoCaptureAsync**.

C#

```
m_advancedCapture =
    await
m_mediaCapture.PrepareAdvancedPhotoCaptureAsync(ImageEncodingProperties.Crea
teUncompressed(MediaPixelFormat.Nv12));
```

Of course, you can always save the image to a file and then load the file into a **SoftwareBitmap** in a separate step. For more information about working with **SoftwareBitmap**, see **Create, edit, and save bitmap images**.

Manual camera controls for photo and video capture

Article • 01/16/2025

This article shows you how to use manual device controls to enable enhanced photo and video capture scenarios including optical image stabilization and smooth zoom.

The controls discussed in this article are all added to your app using the same pattern. First, check to see if the control is supported on the current device on which your app is running. If the control is supported, set the desired mode for the control. Typically, if a particular control is unsupported on the current device, you should disable or hide the UI element that allows the user to enable the feature.

> **Note**
>
> This article builds on concepts and code discussed in Basic photo, video, and audio capture with MediaCapture, which describes the steps for implementing basic photo and video capture. We recommend that you familiarize yourself with the basic media capture pattern in that article before moving on to more advanced capture scenarios. The code in this article assumes that your app already has an instance of **MediaCapture** that has been properly initialized.

Exposure

The **ExposureControl** allows you to set the shutter speed used during photo or video capture.

This example uses a **Slider** control to adjust the current exposure value and a checkbox to toggle automatic exposure adjustment.

XML

```
<Slider Name="slExposure" ValueChanged="slExposure_ValueChanged"/>
<TextBlock Name="tbExposure" Text="{Binding
ElementName=slExposure,Path=Value}"/>
<CheckBox Name="cbExposureAuto" Content="Auto"
Checked="cbExposure_CheckedChanged" Unchecked="cbExposure_CheckedChanged"/>
```

Check to see if the current capture device supports the **ExposureControl** by checking the **Supported** property. If the control is supported, you can show and enable the UI for

this feature. Set the checked state of the checkbox to indicate if automatic exposure adjustment is currently active to the value of the **Auto** property.

The exposure value must be within the range supported by the device and must be an increment of the supported step size. Get the supported values for the current device by checking the **Min**, **Max**, and **Step** properties, which are used to set the corresponding properties of the slider control.

Set the slider control's value to the current value of the **ExposureControl** after unregistering the **ValueChanged** event handler so that the event is not triggered when the value is set.

C#

```
var exposureControl = m_mediaCapture.VideoDeviceController.ExposureControl;

if (exposureControl.Supported)
{
    cbExposureAuto.Visibility = Visibility.Visible;
    slExposure.Visibility = Visibility.Visible;

    cbExposureAuto.IsChecked = exposureControl.Auto;

    slExposure.Minimum = exposureControl.Min.Ticks;
    slExposure.Maximum = exposureControl.Max.Ticks;
    slExposure.StepFrequency = exposureControl.Step.Ticks;

    slExposure.ValueChanged -= slExposure_ValueChanged;
    var value = exposureControl.Value;
    slExposure.Value = value.Ticks;
    slExposure.ValueChanged += slExposure_ValueChanged;
}
else
{
    cbExposureAuto.Visibility = Visibility.Collapsed;
    slExposure.Visibility = Visibility.Collapsed;
}
```

In the **ValueChanged** event handler, get the current value of the control and the set the exposure value by calling **SetValueAsync**.

C#

```
private async void slExposure_ValueChanged(object sender,
Microsoft.UI.Xaml.Controls.Primitives.RangeBaseValueChangedEventArgs e)
{
    var value = TimeSpan.FromTicks((long)(sender as Slider).Value);
    await
```

```
m_mediaCapture.VideoDeviceController.ExposureControl.SetValueAsync(value);
}
```

In the **CheckedChanged** event handler of the auto exposure checkbox, turn automatic exposure adjustment on or off by calling **SetAutoAsync** and passing in a boolean value.

C#

```
private async void cbExposure_CheckedChanged(object sender, RoutedEventArgs
e)
{
    if (!m_isPreviewing)
    {
        // Auto exposure only supported while preview stream is running.
        return;
    }

    var autoExposure = ((sender as CheckBox).IsChecked == true);
    await
m_mediaCapture.VideoDeviceController.ExposureControl.SetAutoAsync(autoExposu
re);
}
```

> **ⓘ Important**
>
> Automatic exposure mode is only supported while the preview stream is running. Check to make sure that the preview stream is running before turning on automatic exposure.

Exposure compensation

The **ExposureCompensationControl** allows you to set the exposure compensation used during photo or video capture.

This example uses a **Slider** control to adjust the current exposure compensation value.

XML

```
<Slider Name="slEV" ValueChanged="slEV_ValueChanged"/>
<TextBlock Text="{Binding ElementName=slEV,Path=Value}" Name="tbEV"/>
```

Check to see if the current capture device supports the **ExposureCompensationControl** by checking the Supported property. If the control is supported, you can show and enable the UI for this feature.

The exposure compensation value must be within the range supported by the device and must be an increment of the supported step size. Get the supported values for the current device by checking the **Min**, **Max**, and **Step** properties, which are used to set the corresponding properties of the slider control.

Set slider control's value to the current value of the **ExposureCompensationControl** after unregistering the **ValueChanged** event handler so that the event is not triggered when the value is set.

C#

```
var exposureCompensationControl =
m_mediaCapture.VideoDeviceController.ExposureCompensationControl;

if (exposureCompensationControl.Supported)
{
    slEV.Visibility = Visibility.Visible;
    slEV.Minimum = exposureCompensationControl.Min;
    slEV.Maximum = exposureCompensationControl.Max;
    slEV.StepFrequency = exposureCompensationControl.Step;

    slEV.ValueChanged -= slEV_ValueChanged;
    slEV.Value = exposureCompensationControl.Value;
    slEV.ValueChanged += slEV_ValueChanged;
}
else
{
    slEV.Visibility = Visibility.Collapsed;
}
```

In the **ValueChanged** event handler, get the current value of the control and the set the exposure value by calling **SetValueAsync**.

C#

```
private async void slEV_ValueChanged(object sender,
Microsoft.UI.Xaml.Controls.Primitives.RangeBaseValueChangedEventArgs e)
{
    var value = (sender as Slider).Value;
    await
m_mediaCapture.VideoDeviceController.ExposureCompensationControl.SetValueAsy
nc((float)value);
}
```

Flash

The FlashControl allows you to enable or disable the flash or to enable automatic flash, where the system dynamically determines whether to use the flash. This control also allows you to enable automatic red eye reduction on devices that support it. These settings all apply to capturing photos. The TorchControl is a separate control for turning the torch on or off for video capture.

This example uses a set of radio buttons to allow the user to switch between on, off, and auto flash settings. A checkbox is also provided to allow toggling of red eye reduction and the video torch.

XML

```
<RadioButton Name="rbFlashOn" Content="On" Checked="rbFlashOn_Checked"/>
<RadioButton Name="rbFlashAuto" Content="Auto"
Checked="rbFlashAuto_Checked"/>
<RadioButton Name="rbFlashOff" Content="Off" Checked="rbFlashOff_Checked"/>
<CheckBox Name="cbRedEyeFlash" Content="Red Eye" Visibility="Collapsed"
Checked="cbRedEyeFlash_CheckedChanged"
Unchecked="cbRedEyeFlash_CheckedChanged"/>
<CheckBox Name="cbTorch" Content="Video Light" Visibility="Collapsed"
Checked="cbTorch_CheckedChanged" Unchecked="cbTorch_CheckedChanged"/>
```

Check to see if the current capture device supports the **FlashControl** by checking the Supported property. If the control is supported, you can show and enable the UI for this feature. If the **FlashControl** is supported, automatic red eye reduction may or may not be supported, so check the RedEyeReductionSupported property before enabling the UI. Because the **TorchControl** is separate from the flash control, you must also check its Supported property before using it.

In the Checked event handler for each of the flash radio buttons, enable or disable the appropriate corresponding flash setting. Note that to set the flash to always be used, you must set the Enabled property to true and the Auto property to false.

C#

```
var flashControl = m_mediaCapture.VideoDeviceController.FlashControl;

if (flashControl.Supported)
{
    rbFlashAuto.Visibility = Visibility.Visible;
    rbFlashOn.Visibility = Visibility.Visible;
    rbFlashOff.Visibility = Visibility.Visible;

    rbFlashAuto.IsChecked = true;

    if (flashControl.RedEyeReductionSupported)
    {
        cbRedEyeFlash.Visibility = Visibility.Visible;
```

```
    }

    // Video light is not strictly part of flash, but users might expect to
find it there
    if (m_mediaCapture.VideoDeviceController.TorchControl.Supported)
    {
        cbTorch.Visibility = Visibility.Visible;
    }
}
else
{
    rbFlashAuto.Visibility = Visibility.Collapsed;
    rbFlashOn.Visibility = Visibility.Collapsed;
    rbFlashOff.Visibility = Visibility.Collapsed;
}
```

C#

```
private void rbFlashOn_Checked(object sender, RoutedEventArgs e)
{
    m_mediaCapture.VideoDeviceController.FlashControl.Enabled = true;
    m_mediaCapture.VideoDeviceController.FlashControl.Auto = false;
}

private void rbFlashAuto_Checked(object sender, RoutedEventArgs e)
{
    m_mediaCapture.VideoDeviceController.FlashControl.Enabled = true;
    m_mediaCapture.VideoDeviceController.FlashControl.Auto = true;
}

private void rbFlashOff_Checked(object sender, RoutedEventArgs e)
{
    m_mediaCapture.VideoDeviceController.FlashControl.Enabled = false;
}
```

In the handler for the red eye reduction checkbox, set the **RedEyeReduction** property to the appropriate value.

C#

```


//private void cbRedEyeFlash_CheckedChanged(object sender, RoutedEventArgs
e)
private void cbRedEyeFlash_CheckedChanged(object sender, RoutedEventArgs e)
{
    m_mediaCapture.VideoDeviceController.FlashControl.RedEyeReduction =
(cbRedEyeFlash.IsChecked == true);
}
```

Finally, in the handler for the video torch checkbox, set the **Enabled** property to the appropriate value.

C#

```
private void cbTorch_CheckedChanged(object sender, RoutedEventArgs e)
{
    m_mediaCapture.VideoDeviceController.TorchControl.Enabled =
(cbTorch.IsChecked == true);

    if(! (m_isPreviewing && m_isRecording))
    {
        System.Diagnostics.Debug.WriteLine("Torch may not emit light if
preview and video capture are not running.");
    }
}
```

> **Note**
>
> On some devices the torch will not emit light, even if TorchControl.Enabled is set to true, unless the device has a preview stream running and is actively capturing video. The recommended order of operations is to turn on the video preview, turn on the torch by setting **Enabled** to true, and then initiate video capture. On some devices the torch will light up after the preview is started. On other devices, the torch may not light up until video capture is started.

Focus

Three different commonly used methods for adjusting the focus of the camera are supported by the **FocusControl** object, continuous autofocus, tap to focus, and manual focus. A camera app may support all three of these methods, but for readability, this article discusses each technique separately. This section also discusses how to enable the focus assist light.

Continuous autofocus

Enabling continuous autofocus instructs the camera to adjust the focus dynamically to try to keep the subject of the photo or video in focus. This example uses a radio button to toggle continuous autofocus on and off.

XML

```
<RadioButton Content="CAF" Name="rbContinuousAutoFocus"
Checked="rbContinuousAutoFocus_Checked"/>
```

Check to see if the current capture device supports the **FocusControl** by checking the **Supported** property. Next, determine if continuous autofocus is supported by checking the **SupportedFocusModes** list to see if it contains the value **FocusMode.Continuous**, and if so, show the continuous autofocus radio button.

C#

```
var focusControl = m_mediaCapture.VideoDeviceController.FocusControl;

if (focusControl.Supported)
{
    rbContinuousAutoFocus.Visibility =
focusControl.SupportedFocusModes.Contains(FocusMode.Continuous)
        ? Visibility.Visible : Visibility.Collapsed;
}
else
{
    rbContinuousAutoFocus.Visibility = Visibility.Collapsed;
}
```

In the **Checked** event handler for the continuous autofocus radio button, use the **VideoDeviceController.FocusControl** property to get an instance of the control. Call **UnlockAsync** to unlock the control in case your app has previously called **LockAsync** to enable one of the other focus modes.

Create a new **FocusSettings** object and set the **Mode** property to **Continuous**. Set the **AutoFocusRange** property to a value appropriate for your app scenario or selected by the user from your UI. Pass your **FocusSettings** object into the **Configure** method, and then call **FocusAsync** to initiate continuous autofocus.

C#

```
private async void rbContinuousAutoFocus_Checked(object sender,
RoutedEventArgs e)
{
    if(! m_isPreviewing)
    {
        // Autofocus only supported while preview stream is running.
        return;
    }

    var focusControl = m_mediaCapture.VideoDeviceController.FocusControl;
    await focusControl.UnlockAsync();
    var settings = new FocusSettings { Mode = FocusMode.Continuous,
```

```
AutoFocusRange = AutoFocusRange.FullRange };
    focusControl.Configure(settings);
    await focusControl.FocusAsync();
}
```

> **ⓘ Important**
>
> Autofocus mode is only supported while the preview stream is running. Check to make sure that the preview stream is running before turning on continuous autofocus.

Tap to focus

The tap-to-focus technique uses the **FocusControl** and the **RegionsOfInterestControl** to specify a subregion of the capture frame where the capture device should focus. The region of focus is determined by the user tapping on the screen displaying the preview stream.

This example uses a radio button to enable and disable tap-to-focus mode.

XML

```
<RadioButton Content="rbTapToFocus" Name="TapFocusRadioButton"
Checked="rbTapToFocus_Checked"/>
```

Check to see if the current capture device supports the **FocusControl** by checking the **Supported** property. The **RegionsOfInterestControl** must be supported, and must support at least one region, in order to use this technique. Check the **AutoFocusSupported** and **MaxRegions** properties to determine whether to show or hide the radio button for tap-to-focus.

C#

```
var focusControl = m_mediaCapture.VideoDeviceController.FocusControl;

if (focusControl.Supported)
{
    TapFocusRadioButton.Visibility =
(m_mediaCapture.VideoDeviceController.RegionsOfInterestControl.AutoFocusSupp
orted &&

m_mediaCapture.VideoDeviceController.RegionsOfInterestControl.MaxRegions >
0)
                                        ? Visibility.Visible :
Visibility.Collapsed;
```

```
}
else
{
    TapFocusRadioButton.Visibility = Visibility.Collapsed;
}
```

In the **Checked** event handler for the tap-to-focus radio button, use the **VideoDeviceController.FocusControl** property to get an instance of the control. Call **LockAsync** to lock the control in case your app has previously called **UnlockAsync** to enable continuous autofocus, and then wait for the user to tap the screen to change the focus.

C#

```
private async void rbTapToFocus_Checked(object sender, RoutedEventArgs e)
{
    // Lock focus in case Continuous Autofocus was active when switching to
Tap-to-focus
    var focusControl = m_mediaCapture.VideoDeviceController.FocusControl;
    await focusControl.LockAsync();
    // Wait for user tap
}
```

This example focuses on a region when the user taps the screen, and then removes the focus from that region when the user taps again, like a toggle. Use a boolean variable to track the current toggled state.

C#

```
bool _isFocused = false;
```

The next step is to listen for the event when the user taps the screen by handling the **Tapped** event of the **CaptureElement** that is currently displaying the capture preview stream. If the camera isn't currently previewing, or if tap-to-focus mode is disabled, return from the handler without doing anything.

If the tracking variable *_isFocused* is toggled to false, and if the camera isn't currently in the process of focus (determined by the **FocusState** property of the **FocusControl**), begin the tap-to-focus process. Get the position of the user's tap from the event args passed into the handler. This example also uses this opportunity to pick the size of the region that will be focused upon. In this case, the size is 1/4 of the smallest dimension of the capture element. Pass the tap position and the region size into the **TapToFocus** helper method that is defined in the next section.

If the *_isFocused* toggle is set to true, the user tap should clear the focus from the previous region. This is done in the **TapUnfocus** helper method shown below.

C#

```
private async void mpePreview_Tapped(object sender, TappedRoutedEventArgs e)
{
    if (!m_isPreviewing || (TapFocusRadioButton.IsChecked != true)) return;

    if (!_isFocused &&
m_mediaCapture.VideoDeviceController.FocusControl.FocusState !=
MediaCaptureFocusState.Searching)
    {
        var smallEdge = Math.Min(Window.Current.Bounds.Width,
Window.Current.Bounds.Height);

        // Choose to make the focus rectangle 1/4th the length of the
shortest edge of the window
        var size = new Size(smallEdge / 4, smallEdge / 4);
        var position = e.GetPosition(sender as UIElement);

        // Note that at this point, a rect at "position" with size "size"
could extend beyond the preview area. The following method will reposition
the rect if that is the case
        await TapToFocus(position, size);
    }
    else
    {
        await TapUnfocus();
    }
}
```

In the **TapToFocus** helper method, first set the *_isFocused* toggle to true so that the next screen tap will release the focus from the tapped region.

The next task in this helper method is to determine the rectangle within the preview stream that will be assigned to the focus control. This requires two steps. The first step is to determine the rectangle that the preview stream takes up within the CaptureElement control. This depends on the dimensions of the preview stream and the orientation of the device. The helper method **GetPreviewStreamRectInControl**, shown at the end of this section, performs this task and returns the rectangle containing the preview stream.

The next task in **TapToFocus** is to convert the tap location and desired focus rectangle size, which were determined within the **CaptureElement.Tapped** event handler, into coordinates within capture stream. The **ConvertUiTapToPreviewRect** helper method, shown later in this section, performs this conversion and returns the rectangle, in capture stream coordinates, where the focus will be requested.

Now that the target rectangle has been obtained, create a new **RegionOfInterest** object, setting the **Bounds** property to the target rectangle obtained in the previous steps.

Get the capture device's **FocusControl**. Create a new **FocusSettings** object and set the **Mode** and **AutoFocusRange** to your desired values, after checking to make sure that they are supported by the **FocusControl**. Call **Configure** on the **FocusControl** to make your settings active and signal the device to begin focusing on the specified region.

Next, get the capture device's **RegionsOfInterestControl** and call **SetRegionsAsync** to set the active region. Multiple regions of interest can be set on devices that support it, but this example only sets a single region.

Finally, call **FocusAsync** on the **FocusControl** to initiate focusing.

> **ⓘ Important**
>
> When implementing tap to focus, the order of operations is important. You should call these APIs in the following order:
>
> 1. FocusControl.Configure
> 2. RegionsOfInterestControl.SetRegionsAsync
> 3. FocusControl.FocusAsync

C#

```
public async Task TapToFocus(Point position, Size size)
{
    _isFocused = true;

    var previewRect = GetPreviewStreamRectInControl();
    var focusPreview = ConvertUiTapToPreviewRect(position, size,
previewRect);

    // Note that this Region Of Interest could be configured to also
calculate exposure
    // and white balance within the region
    var regionOfInterest = new RegionOfInterest
    {
        AutoFocusEnabled = true,
        BoundsNormalized = true,
        Bounds = focusPreview,
        Type = RegionOfInterestType.Unknown,
        Weight = 100,
    };


    var focusControl = m_mediaCapture.VideoDeviceController.FocusControl;
```

```
    var focusRange =
focusControl.SupportedFocusRanges.Contains(AutoFocusRange.FullRange) ?
AutoFocusRange.FullRange :
focusControl.SupportedFocusRanges.FirstOrDefault();
    var focusMode =
focusControl.SupportedFocusModes.Contains(FocusMode.Single) ?
FocusMode.Single : focusControl.SupportedFocusModes.FirstOrDefault();
    var settings = new FocusSettings { Mode = focusMode, AutoFocusRange =
focusRange };
    focusControl.Configure(settings);

    var roiControl =
m_mediaCapture.VideoDeviceController.RegionsOfInterestControl;
    await roiControl.SetRegionsAsync(new[] { regionOfInterest }, true);

    await focusControl.FocusAsync();
}
```

In the **TapUnfocus** helper method, obtain the **RegionsOfInterestControl** and call **ClearRegionsAsync** to clear the region that was registered with the control within the **TapToFocus** helper method. Then, get the **FocusControl** and call **FocusAsync** to cause the device to refocus without a region of interest.

C#

```
private async Task TapUnfocus()
{
    _isFocused = false;

    var roiControl =
m_mediaCapture.VideoDeviceController.RegionsOfInterestControl;
    await roiControl.ClearRegionsAsync();

    var focusControl = m_mediaCapture.VideoDeviceController.FocusControl;
    await focusControl.FocusAsync();
}
```

The **GetPreviewStreamRectInControl** helper method uses the resolution of the preview stream and the orientation of the device to determine the rectangle within the preview element that contains the preview stream, trimming off any letterboxed padding that the control may provide to maintain the stream's aspect ratio. This method uses class member variables defined in the basic media capture example code found in Basic photo, video, and audio capture with MediaCapture.

C#

```
public Rect GetPreviewStreamRectInControl()
{
    var result = new Rect();
```

```
    var previewResolution =
m_mediaCapture.VideoDeviceController.GetMediaStreamProperties(MediaStreamTyp
e.VideoPreview) as VideoEncodingProperties;

    // In case this function is called before everything is initialized
correctly, return an empty result
    if (mpePreview == null || mpePreview.ActualHeight < 1 ||
mpePreview.ActualWidth < 1 ||
        previewResolution == null || previewResolution.Height == 0 ||
previewResolution.Width == 0)
    {
        return result;
    }

    var streamWidth = previewResolution.Width;
    var streamHeight = previewResolution.Height;

    // For portrait orientations, the width and height need to be swapped
    if (m_displayOrientation == DisplayOrientations.Portrait ||
m_displayOrientation == DisplayOrientations.PortraitFlipped)
    {
        streamWidth = previewResolution.Height;
        streamHeight = previewResolution.Width;
    }

    // Start by assuming the preview display area in the control spans the
entire width and height both (this is corrected in the next if for the
necessary dimension)
    result.Width = mpePreview.ActualWidth;
    result.Height = mpePreview.ActualHeight;

    // If UI is "wider" than preview, letterboxing will be on the sides
    if ((mpePreview.ActualWidth / mpePreview.ActualHeight > streamWidth /
(double)streamHeight))
    {
        var scale = mpePreview.ActualHeight / streamHeight;
        var scaledWidth = streamWidth * scale;

        result.X = (mpePreview.ActualWidth - scaledWidth) / 2.0;
        result.Width = scaledWidth;
    }
    else // Preview stream is "wider" than UI, so letterboxing will be on
the top+bottom
    {
        var scale = mpePreview.ActualWidth / streamWidth;
        var scaledHeight = streamHeight * scale;

        result.Y = (mpePreview.ActualHeight - scaledHeight) / 2.0;
        result.Height = scaledHeight;
    }

    return result;
}
```

The **ConvertUiTapToPreviewRect** helper method takes as arguments the location of the tap event, the desired size of the focus region, and the rectangle containing the preview stream obtained from the **GetPreviewStreamRectInControl** helper method. This method uses these values and the device's current orientation to calculate the rectangle within the preview stream that contains the desired region. Once again, this method uses class member variables defined in the basic media capture example code found in Capture Photos and Video with MediaCapture.

C#

```
private Rect ConvertUiTapToPreviewRect(Point tap, Size size, Rect
previewRect)
{
    // Adjust for the resulting focus rectangle to be centered around the
position
    double left = tap.X - size.Width / 2, top = tap.Y - size.Height / 2;

    // Get the information about the active preview area within the
CaptureElement (in case it's letterboxed)
    double previewWidth = previewRect.Width, previewHeight =
previewRect.Height;
    double previewLeft = previewRect.Left, previewTop = previewRect.Top;

    // Transform the left and top of the tap to account for rotation
    switch (m_displayOrientation)
    {
        case DisplayOrientations.Portrait:
            var tempLeft = left;

            left = top;
            top = previewRect.Width - tempLeft;
            break;
        case DisplayOrientations.LandscapeFlipped:
            left = previewRect.Width - left;
            top = previewRect.Height - top;
            break;
        case DisplayOrientations.PortraitFlipped:
            var tempTop = top;

            top = left;
            left = previewRect.Width - tempTop;
            break;
    }

    // For portrait orientations, the information about the active preview
area needs to be rotated
    if (m_displayOrientation == DisplayOrientations.Portrait ||
m_displayOrientation == DisplayOrientations.PortraitFlipped)
    {
        previewWidth = previewRect.Height;
        previewHeight = previewRect.Width;
        previewLeft = previewRect.Top;
```

```
            previewTop = previewRect.Left;
        }

        // Normalize width and height of the focus rectangle
        var width = size.Width / previewWidth;
        var height = size.Height / previewHeight;

        // Shift rect left and top to be relative to just the active preview
area
        left -= previewLeft;
        top -= previewTop;

        // Normalize left and top
        left /= previewWidth;
        top /= previewHeight;

        // Ensure rectangle is fully contained within the active preview area
horizontally
        left = Math.Max(left, 0);
        left = Math.Min(1 - width, left);

        // Ensure rectangle is fully contained within the active preview area
vertically
        top = Math.Max(top, 0);
        top = Math.Min(1 - height, top);

        // Create and return resulting rectangle
        return new Rect(left, top, width, height);
    }
```

Manual focus

The manual focus technique uses a **Slider** control to set the current focus depth of the capture device. A radio button is used to toggle manual focus on and off.

XML

```
<Slider Name="slFocus" IsEnabled="{Binding
ElementName=rbManualFocus,Path=IsChecked}"
ValueChanged="slFocus_ValueChanged"/>
<TextBlock Text="{Binding
ElementName=slFocus,Path=Value,FallbackValue='0'}"/>
<RadioButton Content="Manual" Name="rbManualFocus"
Checked="rbManualFocus_Checked" IsChecked="False"/>
```

Check to see if the current capture device supports the **FocusControl** by checking the **Supported** property. If the control is supported, you can show and enable the UI for this feature.

The focus value must be within the range supported by the device and must be an increment of the supported step size. Get the supported values for the current device by checking the **Min**, **Max**, and **Step** properties, which are used to set the corresponding properties of the slider control.

Set the slider control's value to the current value of the **FocusControl** after unregistering the **ValueChanged** event handler so that the event is not triggered when the value is set.

```
C#
```

```
var focusControl = m_mediaCapture.VideoDeviceController.FocusControl;

if (focusControl.Supported)
{
    slFocus.Visibility = Visibility.Visible;
    rbManualFocus.Visibility = Visibility.Visible;

    slFocus.Minimum = focusControl.Min;
    slFocus.Maximum = focusControl.Max;
    slFocus.StepFrequency = focusControl.Step;


    slFocus.ValueChanged -= slFocus_ValueChanged;
    slFocus.Value = focusControl.Value;
    slFocus.ValueChanged += slFocus_ValueChanged;
}
else
{
    slFocus.Visibility = Visibility.Collapsed;
    rbManualFocus.Visibility = Visibility.Collapsed;
}
```

In the **Checked** event handler for the manual focus radio button, get the **FocusControl** object and call **LockAsync** in case your app had previously unlocked the focus with a call to **UnlockAsync**.

```
C#
```

```
private async void rbManualFocus_Checked(object sender, RoutedEventArgs e)
{
    var focusControl = m_mediaCapture.VideoDeviceController.FocusControl;
    await focusControl.LockAsync();
}
```

In the **ValueChanged** event handler of the manual focus slider, get the current value of the control and the set the focus value by calling **SetValueAsync**.

```
C#
```

```
private async void slFocus_ValueChanged(object sender,
Microsoft.UI.Xaml.Controls.Primitives.RangeBaseValueChangedEventArgs e)
{
    var value = (sender as Slider).Value;
    await
m_mediaCapture.VideoDeviceController.FocusControl.SetValueAsync((uint)value)
;
}
```

Enable the focus light

On devices that support it, you can enable a focus assist light to help the device focus. This example uses a checkbox to enable or disable the focus assist light.

XML

```
<CheckBox Content="Assist Light" Name="cbFocusLight" IsEnabled="{Binding
ElementName=rbTapToFocus,Path=IsChecked}"
Checked="cbFocusLight_CheckedChanged"
Unchecked="cbFocusLight_CheckedChanged"/>
```

Check to see if the current capture device supports the **FlashControl** by checking the **Supported** property. Also check the **AssistantLightSupported** to make sure the assist light is also supported. If these are both supported, you can show and enable the UI for this feature.

C#

```
var focusControl = m_mediaCapture.VideoDeviceController.FocusControl;

if (focusControl.Supported)
{

    cbFocusLight.Visibility =
(m_mediaCapture.VideoDeviceController.FlashControl.Supported &&

m_mediaCapture.VideoDeviceController.FlashControl.AssistantLightSupported) ?
Visibility.Visible : Visibility.Collapsed;
}
else
{
    cbFocusLight.Visibility = Visibility.Collapsed;
}
```

In the **CheckedChanged** event handler, get the capture devices **FlashControl** object. Set the **AssistantLightEnabled** property to enable or disable the focus light.

C#

```
private void cbFocusLight_CheckedChanged(object sender, RoutedEventArgs e)
{
    var flashControl = m_mediaCapture.VideoDeviceController.FlashControl;

    flashControl.AssistantLightEnabled = (cbFocusLight.IsChecked == true);
}
```

ISO speed

The **IsoSpeedControl** allows you to set the ISO speed used during photo or video capture.

This example uses a **Slider** control to adjust the current exposure compensation value and a checkbox to toggle automatic ISO speed adjustment.

XML

```
<Slider Name="slIso" ValueChanged="slIso_ValueChanged"/>
<TextBlock Text="{Binding ElementName=slIso,Path=Value}" Visibility="
{Binding ElementName=slIso,Path=Visibility}"/>
<CheckBox Name="cbIsoAuto" Content="Auto" Checked="cbIsoAuto_CheckedChanged"
Unchecked="cbIsoAuto_CheckedChanged"/>
```

Check to see if the current capture device supports the **IsoSpeedControl** by checking the **Supported** property. If the control is supported, you can show and enable the UI for this feature. Set the checked state of the checkbox to indicate if automatic ISO speed adjustment is currently active to the value of the **Auto** property.

The ISO speed value must be within the range supported by the device and must be an increment of the supported step size. Get the supported values for the current device by checking the **Min**, **Max**, and **Step** properties, which are used to set the corresponding properties of the slider control.

Set the slider control's value to the current value of the **IsoSpeedControl** after unregistering the **ValueChanged** event handler so that the event is not triggered when the value is set.

C#

```
private void bUpdateIsoControlCapabilities_Click(object sender,
RoutedEventArgs e)
{
    var isoSpeedControl =
m_mediaCapture.VideoDeviceController.IsoSpeedControl;
```

```
    if (isoSpeedControl.Supported)
    {
        cbIsoAuto.Visibility = Visibility.Visible;
        slIso.Visibility = Visibility.Visible;

        cbIsoAuto.IsChecked = isoSpeedControl.Auto;

        slIso.Minimum = isoSpeedControl.Min;
        slIso.Maximum = isoSpeedControl.Max;
        slIso.StepFrequency = isoSpeedControl.Step;

        slIso.ValueChanged -= slIso_ValueChanged;
        slIso.Value = isoSpeedControl.Value;
        slIso.ValueChanged += slIso_ValueChanged;
    }
    else
    {
        cbIsoAuto.Visibility = Visibility.Collapsed;
        slIso.Visibility = Visibility.Collapsed;
    }
}
```

In the **ValueChanged** event handler, get the current value of the control and the set the ISO speed value by calling **SetValueAsync**.

C#

```
private async void slIso_ValueChanged(object sender,
Microsoft.UI.Xaml.Controls.Primitives.RangeBaseValueChangedEventArgs e)
{
    var value = (sender as Slider).Value;
    await
m_mediaCapture.VideoDeviceController.IsoSpeedControl.SetValueAsync((uint)val
ue);
}
```

In the **CheckedChanged** event handler of the auto ISO speed checkbox, turn on automatic ISO speed adjustment by calling **SetAutoAsync**. Turn automatic ISO speed adjustment off by calling **SetValueAsync** and passing in the current value of the slider control.

C#

```
private async void cbIsoAuto_CheckedChanged(object sender, RoutedEventArgs
e)
{
    var autoIso = (sender as CheckBox).IsChecked == true;

    if (autoIso)
```

```
    {
        await
m_mediaCapture.VideoDeviceController.IsoSpeedControl.SetAutoAsync();
    }
    else
    {
        await
m_mediaCapture.VideoDeviceController.IsoSpeedControl.SetValueAsync((uint)slI
so.Value);
    }
}
```

Optical image stabilization

Optical image stabilization (OIS) stabilizes a captured video stream by mechanically manipulating the hardware capture device, which can provide a superior result than digital stabilization. On devices that don't support OIS, you can use the VideoStabilizationEffect to perform digital stabilization on your captured video. For more information, see Effects for video capture.

Determine if OIS is supported on the current device by checking the **OpticalImageStabilizationControl.Supported** property.

The OIS control supports three modes: on, off, and automatic, which means that the device dynamically determines if OIS would improve the media capture and, if so, enables OIS. To determine if a particular mode is supported on a device, check to see if the **OpticalImageStabilizationControl.SupportedModes** collection contains the desired mode.

Enable or disable OIS by setting the **OpticalImageStabilizationControl.Mode** to the desired mode.

C#

```
private void SetOpticalImageStabilizationMode(OpticalImageStabilizationMode
mode)
{
    if
(!m_mediaCapture.VideoDeviceController.OpticalImageStabilizationControl.Supp
orted)
    {
        tbStatus.Text = "Optical image stabilization not available";
        return;
    }

    var stabilizationModes =
m_mediaCapture.VideoDeviceController.OpticalImageStabilizationControl.Suppor
tedModes;
```

```
    if (!stabilizationModes.Contains(mode))
    {
        tbStatus.Text = "Optical image stabilization setting not supported";
        return;
    }


m_mediaCapture.VideoDeviceController.OpticalImageStabilizationControl.Mode =
mode;
}
```

Powerline frequency

Some camera devices support anti-flicker processing that depends on knowing the AC frequency of the powerlines in the current environment. Some devices support automatic determination of the powerline frequency, while others require that the frequency be set manually. The following code example shows how to determine powerline frequency support on the device and, if needed, how to set the frequency manually.

First, call the **VideoDeviceController** method **TryGetPowerlineFrequency**, passing in an output parameter of type **PowerlineFrequency**; if this call fails, the powerline frequency control is not supported on the current device. If the feature is supported, you can determine if automatic mode is available on the device by trying to set auto mode. Do this by calling **TrySetPowerlineFrequency** and passing in the value **Auto**. If the call succeeds, that means that your auto powerline frequency is supported. If the powerline frequency controller is supported on the device but automatic frequency detection is not, you can still manually set the frequency by using **TrySetPowerlineFrequency**. In this example, **MyCustomFrequencyLookup** is a custom method that you implement to determine the correct frequency for the device's current location.

C#

```
 PowerlineFrequency getFrequency;

 if (! m_mediaCapture.VideoDeviceController.TryGetPowerlineFrequency(out
getFrequency))
 {
     // Powerline frequency is not supported on this device.
     return;
 }

 if (!
m_mediaCapture.VideoDeviceController.TrySetPowerlineFrequency(PowerlineFrequ
ency.Auto))
 {
```

```
        // Set the frequency manually
        PowerlineFrequency setFrequency = MyCustomFrequencyLookup();
        if
(m_mediaCapture.VideoDeviceController.TrySetPowerlineFrequency(setFrequency)
)
        {
            System.Diagnostics.Debug.WriteLine(String.Format("Powerline
frequency manually set to {0}.", setFrequency));
        }
    }
```

White balance

The **WhiteBalanceControl** allows you to set the white balance used during photo or video capture.

This example uses a **ComboBox** control to select from built-in color temperature presets and a **Slider** control for manual white balance adjustment.

XML

```
<Slider Name="slWhiteBalance" ValueChanged="slWhiteBalance_ValueChanged"/>
<TextBlock Name="tbWhiteBalance" Text="{Binding
ElementName=slWhiteBalance,Path=Value}" Visibility="{Binding
ElementName=slWhiteBalance,Path=Visibility}"/>
<ComboBox Name="cbWhiteBalance"
SelectionChanged="cbWhiteBalance_SelectionChanged"/>
```

Check to see if the current capture device supports the **WhiteBalanceControl** by checking the **Supported** property. If the control is supported, you can show and enable the UI for this feature. Set the items of the combo box to the values of the **ColorTemperaturePreset** enumeration. And set the selected item to the current value of the **Preset** property.

For manual control, the white balance value must be within the range supported by the device and must be an increment of the supported step size. Get the supported values for the current device by checking the **Min**, **Max**, and **Step** properties, which are used to set the corresponding properties of the slider control. Before enabling manual control, check to make sure that the range between the minimum and maximum supported values is greater than the step size. If it is not, manual control is not supported on the current device.

Set the slider control's value to the current value of the **WhiteBalanceControl** after unregistering the **ValueChanged** event handler so that the event is not triggered when the value is set.

C#

```
            var whiteBalanceControl =
m_mediaCapture.VideoDeviceController.WhiteBalanceControl;

            if (whiteBalanceControl.Supported)
            {
                slWhiteBalance.Visibility = Visibility.Visible;
                cbWhiteBalance.Visibility = Visibility.Visible;

                if (cbWhiteBalance.ItemsSource == null)
                {
                    cbWhiteBalance.ItemsSource =
Enum.GetValues(typeof(ColorTemperaturePreset)).Cast<ColorTemperaturePreset>
();
                }

                cbWhiteBalance.SelectedItem = whiteBalanceControl.Preset;

                if (whiteBalanceControl.Max - whiteBalanceControl.Min >
whiteBalanceControl.Step)
                {

                    slWhiteBalance.Minimum = whiteBalanceControl.Min;
                    slWhiteBalance.Maximum = whiteBalanceControl.Max;
                    slWhiteBalance.StepFrequency = whiteBalanceControl.Step;

                    slWhiteBalance.ValueChanged -=
slWhiteBalance_ValueChanged;
                    slWhiteBalance.Value = whiteBalanceControl.Value;
                    slWhiteBalance.ValueChanged +=
slWhiteBalance_ValueChanged;
                }
                else
                {
                    slWhiteBalance.Visibility = Visibility.Collapsed;
                }
            }
            else
            {
                slWhiteBalance.Visibility = Visibility.Collapsed;
                cbWhiteBalance.Visibility = Visibility.Collapsed;
            }
```

In the SelectionChanged event handler of the color temperature preset combo box, get the currently selected preset and set the value of the control by calling SetPresetAsync. If the selected preset value is not **Manual**, disable the manual white balance slider.

C#

```
private async void cbWhiteBalance_SelectionChanged(object sender,
SelectionChangedEventArgs e)
```

```
{
    if(!m_isPreviewing)
    {
        // Do not set white balance values unless the preview stream is
running.
        return;
    }

    var selected = (ColorTemperaturePreset)cbWhiteBalance.SelectedItem;
    slWhiteBalance.IsEnabled = (selected == ColorTemperaturePreset.Manual);
    await
m_mediaCapture.VideoDeviceController.WhiteBalanceControl.SetPresetAsync(sele
cted);

}
```

In the **ValueChanged** event handler, get the current value of the control and the set the white balance value by calling **SetValueAsync**.

C#

```
private async void slWhiteBalance_ValueChanged(object sender,
Microsoft.UI.Xaml.Controls.Primitives.RangeBaseValueChangedEventArgs e)
{
    if (!m_isPreviewing)
    {
        // Do not set white balance values unless the preview stream is
running.
        return;
    }

    var value = (sender as Slider).Value;
    await
m_mediaCapture.VideoDeviceController.WhiteBalanceControl.SetValueAsync((uint
)value);
}
```

ⓘ Important

Adjusting the white balance is only supported while the preview stream is running. Check to make sure that the preview stream is running before setting the white balance value or preset.

ⓘ Important

The **ColorTemperaturePreset.Auto** preset value instructs the system to automatically adjust the white balance level. For some scenarios, such as capturing

a photo sequence where the white balance levels should be the same for each frame, you may want to lock the control to the current automatic value. To do this, call SetPresetAsync and specify the **Manual** preset and do not set a value on the control using SetValueAsync. This will cause the device to lock the current value. Do not attempt to read the current control value and then pass the returned value into **SetValueAsync** because this value is not guaranteed to be correct.

Zoom

The ZoomControl allows you to set the zoom level used during photo or video capture.

This example uses a Slider control to adjust the current zoom level. The following section shows how to adjust zoom based on a pinch gesture on the screen.

XML

```
<Slider Name="slZoom" Grid.Row="0" Orientation="Vertical"
HorizontalAlignment="Center" VerticalAlignment="Stretch"
ValueChanged="slZoom_ValueChanged"/>
<TextBlock Grid.Row="1" HorizontalAlignment="Center" Text="{Binding
ElementName=slZoom,Path=Value}"/>
<Button x:Name="bRegisterPinchGestureHandler" Content="Register pinch
gesture handler" Click="bRegisterPinchGestureHandler_Click"/>
```

Check to see if the current capture device supports the **ZoomControl** by checking the Supported property. If the control is supported, you can show and enable the UI for this feature.

The zoom level value must be within the range supported by the device and must be an increment of the supported step size. Get the supported values for the current device by checking the Min, Max, and Step properties, which are used to set the corresponding properties of the slider control.

Set the slider control's value to the current value of the **ZoomControl** after unregistering the ValueChanged event handler so that the event is not triggered when the value is set.

C#

```
var zoomControl = m_mediaCapture.VideoDeviceController.ZoomControl;

if (zoomControl.Supported)
{
    slZoom.Visibility = Visibility.Visible;

    slZoom.Minimum = zoomControl.Min;
```

```
    slZoom.Maximum = zoomControl.Max;
    slZoom.StepFrequency = zoomControl.Step;

    slZoom.ValueChanged -= slZoom_ValueChanged;
    slZoom.Value = zoomControl.Value;
    slZoom.ValueChanged += slZoom_ValueChanged;
}
else
{
    slZoom.Visibility = Visibility.Collapsed;
}
```

In the **ValueChanged** event handler, create a new instance of the ZoomSettings class, setting the Value property to the current value of the zoom slider control. If the SupportedModes property of the **ZoomControl** contains ZoomTransitionMode.Smooth, it means the device supports smooth transitions between zoom levels. Since this modes provides a better user experience, you will typically want to use this value for the Mode property of the **ZoomSettings** object.

Finally, change the current zoom settings by passing your **ZoomSettings** object into the Configure method of the **ZoomControl** object.

C#

```
private void slZoom_ValueChanged(object sender,
Microsoft.UI.Xaml.Controls.Primitives.RangeBaseValueChangedEventArgs e)
{
    var level = (float)slZoom.Value;
    var settings = new ZoomSettings { Value = level };

    var zoomControl = m_mediaCapture.VideoDeviceController.ZoomControl;
    if (zoomControl.SupportedModes.Contains(ZoomTransitionMode.Smooth))
    {
        settings.Mode = ZoomTransitionMode.Smooth;
    }
    else
    {
        settings.Mode = zoomControl.SupportedModes.First();
    }

    zoomControl.Configure(settings);
}
```

Smooth zoom using pinch gesture

As discussed in the previous section, on devices that support it, smooth zoom mode allows the capture device to smoothly transition between digital zoom levels, allowing the user to dynamically adjust the zoom level during the capture operation without

discrete and jarring transitions. This section describes how to adjust the zoom level in response to a pinch gesture.

First, determine if the digital zoom control is supported on the current device by checking the **ZoomControl.Supported** property. Next, determine if smooth zoom mode is available by checking the **ZoomControl.SupportedModes** to see if it contains the value **ZoomTransitionMode.Smooth**.

C#

```
private bool IsSmoothZoomSupported()
{
    if (!m_mediaCapture.VideoDeviceController.ZoomControl.Supported)
    {
        tbStatus.Text = "Digital zoom is not supported on this device.";
        return false;
    }

    var zoomModes =
m_mediaCapture.VideoDeviceController.ZoomControl.SupportedModes;

    if (!zoomModes.Contains(ZoomTransitionMode.Smooth))
    {
        tbStatus.Text = "Smooth zoom not supported";
        return false;
    }

    return true;
}
```

On a multi-touch enabled device, a typical scenario is to adjust the zoom factor based on a two-finger pinch gesture. Set the **ManipulationMode** property of the **CaptureElement** control to **ManipulationModes.Scale** to enable the pinch gesture. Then, register for the **ManipulationDelta** event which is raised when the pinch gesture changes size.

C#

```
private void RegisterPinchGestureHandler()
{
    if (!IsSmoothZoomSupported())
    {
        return;
    }

    // Enable pinch/zoom gesture for the MediaCaptureElement
    mpePreview.ManipulationMode = ManipulationModes.Scale;
    mpePreview.ManipulationDelta += MpePreview_ManipulationDelta;
}
```

In the handler for the **ManipulationDelta** event, update the zoom factor based on the change in the user's pinch gesture. The ManipulationDelta.Scale value represents the change in scale of the pinch gesture such that a small increase in the size of the pinch is a number slightly larger than 1.0 and a small decrease in the pinch size is a number slightly smaller than 1.0. In this example, the current value of the zoom control is multiplied by the scale delta.

Before setting the zoom factor, you must make sure that the value is not less than the minimum value supported by the device as indicated by the ZoomControl.Min property. Also, make sure that the value is less than or equal to the ZoomControl.Max value. Finally, you must make sure that the zoom factor is a multiple of the zoom step size supported by the device as indicated by the Step property. If your zoom factor does not meet these requirements, an exception will be thrown when you attempt to set the zoom level on the capture device.

Set the zoom level on the capture device by creating a new ZoomSettings object. Set the Mode property to ZoomTransitionMode.Smooth and then set the Value property to your desired zoom factor. Finally, call ZoomControl.Configure to set the new zoom value on the device. The device will smoothly transition to the new zoom value.

C#

```
private void MpePreview_ManipulationDelta(object sender,
ManipulationDeltaRoutedEventArgs e)
{
    var zoomControl = m_mediaCapture.VideoDeviceController.ZoomControl;

    // Example zoom factor calculation based on size of scale gesture
    var zoomFactor = zoomControl.Value * e.Delta.Scale;

    if (zoomFactor < zoomControl.Min) zoomFactor = zoomControl.Min;
    if (zoomFactor > zoomControl.Max) zoomFactor = zoomControl.Max;
    zoomFactor = zoomFactor - (zoomFactor % zoomControl.Step);

    var settings = new ZoomSettings();
    settings.Mode = ZoomTransitionMode.Smooth;
    settings.Value = zoomFactor;

    m_mediaCapture.VideoDeviceController.ZoomControl.Configure(settings);

}
```

Related topics

- Camera
- Basic photo, video, and audio capture with MediaCapture

Manual camera controls for video capture

Article • 01/16/2025

This article shows you how to use manual device controls to enable enhanced video capture scenarios, including HDR video and exposure priority.

The video device controls discussed in this article are all added to your app by using the same pattern. First, check to see if the control is supported on the current device on which your app is running. If the control is supported, set the desired mode for the control. Typically, if a particular control is unsupported on the current device, you should disable or hide the UI element that allows the user to enable the feature.

> **Note**
>
> This article builds on concepts and code discussed in Basic photo, video, and audio capture with MediaCapture, which describes the steps for implementing basic photo and video capture. We recommend that you familiarize yourself with the basic media capture pattern in that article before moving on to more advanced capture scenarios. The code in this article assumes that your app already has an instance of **MediaCapture** that has been properly initialized.

HDR video

The high dynamic range (HDR) video feature applies HDR processing to the video stream of the capture device. Determine if HDR video is supported by selecting the **HdrVideoControl.Supported** property.

The HDR video control supports three modes: on, off, and automatic, which means that the device dynamically determines if HDR video processing would improve the media capture and, if so, enables HDR video. To determine if a particular mode is supported on the current device, check to see if the **HdrVideoControl.SupportedModes** collection contains the desired mode.

Enable or disable HDR video processing by setting the **HdrVideoControl.Mode** to the desired mode. This control requires that the stream is at a stopped state before the mode is set, see KSPROPERTY_CAMERACONTROL_EXTENDED_VIDEOHDR.

```
C#
```

```
private void SetHdrVideoMode(HdrVideoMode mode)
{
    if (!m_mediaCapture.VideoDeviceController.HdrVideoControl.Supported)
    {
        tbStatus.Text = "HDR Video not available";
        return;
    }

    var hdrVideoModes =
m_mediaCapture.VideoDeviceController.HdrVideoControl.SupportedModes;

    if (!hdrVideoModes.Contains(mode))
    {
        tbStatus.Text = "HDR Video setting not supported";
        return;
    }

    m_mediaCapture.VideoDeviceController.HdrVideoControl.Mode = mode;
}
```

Exposure priority

The **ExposurePriorityVideoControl**, when enabled, evaluates the video frames from the capture device to determine if the video is capturing a low-light scene. If so, the control lowers the frame rate of the captured video in order to increase the exposure time for each frame and improve the visual quality of the captured video.

Determine if the exposure priority control is supported on the current device by checking the **ExposurePriorityVideoControl.Supported** property.

Enable or disable the exposure priority control by setting the **ExposurePriorityVideoControl.Enabled** to the desired mode.

C#

```
if
(!m_mediaCapture.VideoDeviceController.ExposurePriorityVideoControl.Supporte
d)
{
    tbStatus.Text = "Exposure priority not available";
    return;
}
m_mediaCapture.VideoDeviceController.ExposurePriorityVideoControl.Enabled =
true;
```

Temporal denoising

Starting with Windows 10, version 1803, you can enable temporal denoising for video on devices that support it. This feature fuses the image data from multiple adjacent frames in real time to produce video frames that have less visual noise.

The **VideoTemporalDenoisingControl** allows your app to determine if temporal denoising is supported on the current device, and if so, which denoising modes are supported. The available denoising modes are **Off**, **On**, and **Auto**. A device may not support all modes, but every device must support either **Auto** or **On** and **Off**.

The following example uses a simple UI to provide radio buttons allowing the user to switch between denoising modes.

XML

```
<StackPanel Orientation="Vertical" HorizontalAlignment="Right" >
    <StackPanel x:Name="spDenoise" Visibility="Collapsed">
        <TextBlock>Temporal Denoising</TextBlock>
        <RadioButton x:Name="rbDenoiseOff" Checked="rbDenoise_Checked"
            GroupName="Denoise Group" Content="Off"/>
        <RadioButton x:Name="rbDenoiseOn" Checked="rbDenoise_Checked"
            GroupName="Denoise Group" Content="On" Visibility="Collapsed"/>
        <RadioButton x:Name="rbDenoiseAuto" Checked="rbDenoise_Checked"
            GroupName="Denoise Group" Content="Auto"
Visibility="Collapsed"/>
    </StackPanel>
</StackPanel>
```

In the following method, the **VideoTemporalDenoisingControl.Supported** property is checked to see if temporal denoising is supported at all on the current device. If so, then we check to make sure that **Off** and **Auto** or **On** is supported, in which case we make our radio buttons visible. Next, the **Auto** and **On** buttons are made visible if those methods are supported.

C#

```
private void bUpdateDenoiseCapabilities_Click(object sender, RoutedEventArgs
e)
{
    if
(m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Supporte
d)
    {
         IReadOnlyList<VideoTemporalDenoisingMode> modes =
m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Supported
Modes;

if(modes.Contains(Windows.Media.Devices.VideoTemporalDenoisingMode.Off) &&

(modes.Contains(Windows.Media.Devices.VideoTemporalDenoisingMode.On) ||
```

```
modes.Contains(Windows.Media.Devices.VideoTemporalDenoisingMode.Auto)))
        {
            spDenoise.Visibility = Visibility.Visible;

            if
(modes.Contains(Windows.Media.Devices.VideoTemporalDenoisingMode.On))
            {
                rbDenoiseOn.Visibility = Visibility.Visible;
            }
            if
(modes.Contains(Windows.Media.Devices.VideoTemporalDenoisingMode.Auto))
            {
                rbDenoiseAuto.Visibility = Visibility.Visible;
            }
        }
    }
}
```

In the **Checked** event handler for the radio buttons, the name of the button is checked and the corresponding mode is set by setting the VideoTemporalDenoisingControl.Mode property.

C#

```
private void rbDenoise_Checked(object sender, RoutedEventArgs e)
{
    var button = sender as RadioButton;
    if(button.Name == "denoiseOffButton")
    {

m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Mode =
Windows.Media.Devices.VideoTemporalDenoisingMode.Off;
    }
    else if (button.Name == "denoiseOnButton")
    {

m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Mode =
Windows.Media.Devices.VideoTemporalDenoisingMode.On;
    }
    else if (button.Name == "denoiseAutoButton")
    {

m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Mode =
Windows.Media.Devices.VideoTemporalDenoisingMode.Auto;
    }
}
```

Disabling temporal denoising while processing frames

Video that has been processed using temporal denoising can be more pleasing to the human eye. However, because temporal denoising can impact image consistency and decrease the amount of details in the frame, apps that perform image processing on the frames, such as registration or optical character recognition, may want to programmatically disable denoising when image processing is enabled.

The following example determines which denoising modes are supported and stores this information in some class variables.

C#

```
private bool _isVideoTemporalDenoisingOffSupported = false;
private bool _isProcessing = false;
private Windows.Media.Devices.VideoTemporalDenoisingMode?
_videoDenoisingEnabledMode = null;
```

C#

```
private void ConfigureDenoiseForFrameProcessing()
{
    if
(m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Supporte
d)
    {
        // Query support for the VideoTemporalDenoising control Off mode
        _isVideoTemporalDenoisingOffSupported =
m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Supported
Modes.Contains(Windows.Media.Devices.VideoTemporalDenoisingMode.Off);

        // Query support for a mode that would enable VideoTemporalDenoising
(On or Auto) and toggle it if supported
        if
(m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Supporte
dModes.Contains(Windows.Media.Devices.VideoTemporalDenoisingMode.On))
        {
            _videoDenoisingEnabledMode =
Windows.Media.Devices.VideoTemporalDenoisingMode.On;

m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Mode =
(Windows.Media.Devices.VideoTemporalDenoisingMode)_videoDenoisingEnabledMode
;
        }
        else if
(m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Supporte
dModes.Contains(Windows.Media.Devices.VideoTemporalDenoisingMode.Auto))
        {
            _videoDenoisingEnabledMode =
Windows.Media.Devices.VideoTemporalDenoisingMode.Auto;

m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Mode =
(Windows.Media.Devices.VideoTemporalDenoisingMode)_videoDenoisingEnabledMode
```

```
;
            }
        }

    }
```

When the app enables frame processing, it sets the denoising mode to **Off** if that mode is supported so that the frame processing can use raw frames that have not been denoised.

C#

```
public void EnableFrameProcessing()
{
    // Toggle Off VideoTemporalDenoising
    if (_isVideoTemporalDenoisingOffSupported)
    {

m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Mode =
Windows.Media.Devices.VideoTemporalDenoisingMode.Off;
    }

    _isProcessing = true;
}
```

When the app disables frame processing, it sets the denoising mode to **On** or **Auto**, depending on which mode is supported.

C#

```
public void DisableFrameProcessing()
{
    _isProcessing = false;

    // If a VideoTemporalDenoising mode to enable VideoTemporalDenoising is
supported, toggle it
    if (_videoDenoisingEnabledMode != null)
    {

m_mediaCapture.VideoDeviceController.VideoTemporalDenoisingControl.Mode =
(Windows.Media.Devices.VideoTemporalDenoisingMode)_videoDenoisingEnabledMode
;
    }
}
```

For more information on obtaining video frames for image processing, see Process media frames with MediaFrameReader.

Related topics

- Camera
- Basic photo, video, and audio capture with MediaCapture
- Process media frames with MediaFrameReader
- **VideoTemporalDenoisingControl**

Effects for video capture

Article • 01/16/2025

This topic shows you how to apply effects to the camera preview and recording video streams and shows you how to use the video stabilization effect.

> **Note**
>
> This article builds on concepts and code discussed in Basic photo, video, and audio capture with MediaCapture, which describes the steps for implementing basic photo and video capture. We recommend that you familiarize yourself with the basic media capture pattern in that article before moving on to more advanced capture scenarios. The code in this article assumes that your app already has an instance of **MediaCapture** that has been properly initialized.

Adding and removing effects from the camera video stream

To capture or preview video from the device's camera, use the **MediaCapture** class as described in Basic photo, video, and audio capture with MediaCapture. After you have initialized the **MediaCapture** object, you can add one or more video effects to the preview or capture stream by calling **AddVideoEffectAsync**, passing in an **IVideoEffectDefinition** object representing the effect to be added, and a member of the **MediaStreamType** enumeration indicating whether the effect should be added to the camera's preview stream or the record stream.

> **Note**
>
> On some devices, the preview stream and the capture stream are the same, which means that if you specify **MediaStreamType.VideoPreview** or **MediaStreamType.VideoRecord** when you call **AddVideoEffectAsync**, the effect will be applied to both preview and record streams. You can determine whether the preview and record streams are the same on the current device by checking the VideoDeviceCharacteristic property of the MediaCaptureSettings for the **MediaCapture** object. If the value of this property is **VideoDeviceCharacteristic.AllStreamsIdentical** or **VideoDeviceCharacteristic.PreviewRecordStreamsIdentical**, then the streams are the same and any effect you apply to one will affect the other.

The following example adds an effect to both the camera preview and record streams. This example illustrates checking to see if the record and preview streams are the same.

C#

```
if (m_mediaCapture.MediaCaptureSettings.VideoDeviceCharacteristic ==
VideoDeviceCharacteristic.AllStreamsIdentical ||
    m_mediaCapture.MediaCaptureSettings.VideoDeviceCharacteristic ==
VideoDeviceCharacteristic.PreviewRecordStreamsIdentical)
{
    // This effect will modify both the preview and the record streams,
because they are the same stream.
    myRecordEffect = await
m_mediaCapture.AddVideoEffectAsync(myEffectDefinition,
MediaStreamType.VideoRecord);
}
else
{
    myRecordEffect = await
m_mediaCapture.AddVideoEffectAsync(myEffectDefinition,
MediaStreamType.VideoRecord);
    myPreviewEffect = await
m_mediaCapture.AddVideoEffectAsync(myEffectDefinition,
MediaStreamType.VideoPreview);
}
```

Note that **AddVideoEffectAsync** returns an object that implements **IMediaExtension** that represents the added video effect. Some effects allow you to change the effect settings by passing a **PropertySet** into the **SetProperties** method.

Starting with Windows 10, version 1607, you can also use the object returned by **AddVideoEffectAsync** to remove the effect from the video pipeline by passing it into **RemoveEffectAsync**. **RemoveEffectAsync** automatically determines whether the effect object parameter was added to the preview or record stream, so you don't need to specify the stream type when making the call.

C#

```
if (myRecordEffect != null)
{
    await m_mediaCapture.RemoveEffectAsync(myRecordEffect);
}
if (myPreviewEffect != null)
{
    await m_mediaCapture.RemoveEffectAsync(myPreviewEffect);
}
```

You can also remove all effects from the preview or capture stream by calling ClearEffectsAsync and specifying the stream for which all effects should be removed.

```
C#

await m_mediaCapture.ClearEffectsAsync(MediaStreamType.VideoPreview);
await m_mediaCapture.ClearEffectsAsync(MediaStreamType.VideoRecord);
```

Video stabilization effect

The video stabilization effect manipulates the frames of a video stream to minimize shaking caused by holding the capture device in your hand. Because this technique causes the pixels to be shifted right, left, up, and down, and because the effect can't know what the content just outside the video frame is, the stabilized video is cropped slightly from the original video. A utility function is provided to allow you to adjust your video encoding settings to optimally manage the cropping performed by the effect.

On devices that support it, Optical Image Stabilization (OIS) stabilizes video by mechanically manipulating the capture device and, therefore, does not need to crop the edges of the video frames. For more information, see Capture device controls for video capture.

Set up your app to use video stabilization

Declare a member variable to store the VideoStabilizationEffect object. As part of the effect implementation, you will modify the encoding properties that you use to encode the captured video. Declare two variables to store a backup copy of the initial input and output encoding properties so that you can restore them later when the effect is disabled. Finally, declare a member variable of type MediaEncodingProfile because this object will be accessed from multiple locations within your code.

```
C#

//
private VideoStabilizationEffect m_videoStabilizationEffect;
private VideoEncodingProperties m_inputPropertiesBackup;
private VideoEncodingProperties m_outputPropertiesBackup;
private MediaEncodingProfile m_encodingProfile;
```

For this scenario, you should assign the media encoding profile object to a member variable so that you can access it later.

```
```

C#

```
m_encodingProfile =
MediaEncodingProfile.CreateMp4(VideoEncodingQuality.Auto);
```

Initialize the video stabilization effect

After your **MediaCapture** object has been initialized, create a new instance of the **VideoStabilizationEffectDefinition** object. Call **MediaCapture.AddVideoEffectAsync** to add the effect to the video pipeline and retrieve an instance of the **VideoStabilizationEffect** class. Specify **MediaStreamType.VideoRecord** to indicate that the effect should be applied to the video record stream.

Register an event handler for the **EnabledChanged** event and call the helper method **SetUpVideoStabilizationRecommendationAsync**, both of which are discussed later in this article. Finally, set the **Enabled** property of the effect to true to enable the effect.

C#

```
// Create the effect definition
VideoStabilizationEffectDefinition stabilizerDefinition = new
VideoStabilizationEffectDefinition();

// Add the video stabilization effect to media capture
m_videoStabilizationEffect =
    (VideoStabilizationEffect)await
m_mediaCapture.AddVideoEffectAsync(stabilizerDefinition,
MediaStreamType.VideoRecord);

m_videoStabilizationEffect.EnabledChanged +=
VideoStabilizationEffect_EnabledChanged;

await SetUpVideoStabilizationRecommendationAsync();

m_videoStabilizationEffect.Enabled = true;
```

Use recommended encoding properties

As discussed earlier in this article, the technique that the video stabilization effect uses necessarily causes the stabilized video to be cropped slightly from the source video. Define the following helper function in your code in order to adjust the video encoding properties to optimally handle this limitation of the effect. This step is not required in order to use the video stabilization effect, but if you don't perform this step, the resulting video will be upscaled slightly and therefore have slightly lower visual fidelity.

Call **GetRecommendedStreamConfiguration** on your video stabilization effect instance, passing in the **VideoDeviceController** object, which informs the effect about your current input stream encoding properties, and your **MediaEncodingProfile** which lets the effect know your current output encoding properties. This method returns a **VideoStreamConfiguration** object containing new recommended input and output stream encoding properties.

The recommended input encoding properties are, if it is supported by the device, a higher resolution that the initial settings you provided so that there is minimal loss in resolution after the effect's cropping is applied.

Call **VideoDeviceController.SetMediaStreamPropertiesAsync** to set the new encoding properties. Before setting the new properties, use the member variable to store the initial encoding properties so that you can change the settings back when you disable the effect.

If the video stabilization effect must crop the output video, the recommended output encoding properties will be the size of the cropped video. This means that the output resolution will match the cropped video size. If you do not use the recommended output properties, the video will be scaled up to match the initial output size, which will result in a loss of visual fidelity.

Set the **Video** property of the **MediaEncodingProfile** object. Before setting the new properties, use the member variable to store the initial encoding properties so that you can change the settings back when you disable the effect.

C#

```
private async Task SetUpVideoStabilizationRecommendationAsync()
{

    // Get the recommendation from the effect based on our current input and
output configuration
    var recommendation =
m_videoStabilizationEffect.GetRecommendedStreamConfiguration(m_mediaCapture.
VideoDeviceController, m_encodingProfile.Video);

    // Handle the recommendation for the input into the effect, which can
contain a larger resolution than currently configured, so cropping is
minimized
    if (recommendation.InputProperties != null)
    {
        // Back up the current input properties from before VS was activated
        m_inputPropertiesBackup =
m_mediaCapture.VideoDeviceController.GetMediaStreamProperties(MediaStreamTyp
e.VideoRecord) as VideoEncodingProperties;

        // Set the recommendation from the effect (a resolution higher than
```

```
the current one to allow for cropping) on the input
        await
m_mediaCapture.VideoDeviceController.SetMediaStreamPropertiesAsync(MediaStre
amType.VideoRecord, recommendation.InputProperties);
        await
m_mediaCapture.VideoDeviceController.SetMediaStreamPropertiesAsync(MediaStre
amType.VideoPreview, recommendation.InputProperties);
    }

    // Handle the recommendations for the output from the effect
    if (recommendation.OutputProperties != null)
    {
        // Back up the current output properties from before VS was
activated
        m_outputPropertiesBackup = m_encodingProfile.Video;

        // Apply the recommended encoding profile for the output
        m_encodingProfile.Video = recommendation.OutputProperties;
    }
}
```

Handle the video stabilization effect being disabled

The system may automatically disable the video stabilization effect if the pixel throughput is too high for the effect to handle or if it detects that the effect is running slowly. If this occurs, the EnabledChanged event is raised. The **VideoStabilizationEffect** instance in the *sender* parameter indicates the new state of the effect, enabled or disabled. The VideoStabilizationEffectEnabledChangedEventArgs has a VideoStabilizationEffectEnabledChangedReason value indicating why the effect was enabled or disabled. Note that this event is also raised if you programmatically enable or disable the effect, in which case the reason will be **Programmatic**.

Typically, you would use this event to adjust your app's UI to indicate the current status of video stabilization.

C#

```
private async void
VideoStabilizationEffect_EnabledChanged(VideoStabilizationEffect sender,
VideoStabilizationEffectEnabledChangedEventArgs args)
{
    await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
    {
        // Update your UI to reflect the change in status
        tbStatus.Text = "video stabilization status: " + sender.Enabled + ".
Reason: " + args.Reason;
    });
}
```

Clean up the video stabilization effect

To clean up the video stabilization effect, call **RemoveEffectAsync** to remove the effect from the video pipeline. If the member variables containing the initial encoding properties are not null, use them to restore the encoding properties. Finally, remove the **EnabledChanged** event handler and set the effect to null.

C#

```
// Clear all effects in the pipeline
await m_mediaCapture.RemoveEffectAsync(m_videoStabilizationEffect);

// If backed up settings (stream properties and encoding profile) exist,
restore them and clear the backups
if (m_inputPropertiesBackup != null)
{
    await
m_mediaCapture.VideoDeviceController.SetMediaStreamPropertiesAsync(MediaStre
amType.VideoRecord, m_inputPropertiesBackup);
    m_inputPropertiesBackup = null;
}

if (m_outputPropertiesBackup != null)
{
    m_encodingProfile.Video = m_outputPropertiesBackup;
    m_outputPropertiesBackup = null;
}

m_videoStabilizationEffect.EnabledChanged -=
VideoStabilizationEffect_EnabledChanged;

m_videoStabilizationEffect = null;
```

Related topics

- Camera
- Basic photo, video, and audio capture with MediaCapture

Scene analysis for MediaCapture

Article • 01/16/2025

This article describes how to use the **SceneAnalysisEffect** and the **FaceDetectionEffect** to analyze the content of the media capture preview stream.

Scene analysis effect

The **SceneAnalysisEffect** analyzes the video frames in the media capture preview stream and recommends processing options to improve the capture result. Currently, the effect supports detecting whether the capture would be improved by using High Dynamic Range (HDR) processing.

If the effect recommends using HDR, you can do this in the following ways:

- Use the **AdvancedPhotoCapture** class to capture photos using the Windows built-in HDR processing algorithm. For more information, see High dynamic range (HDR) and low-light photo capture.
- Use the **HdrVideoControl** to capture video using the Windows built-in HDR processing algorithm. For more information, see Capture device controls for video capture.
- Use the **VariablePhotoSequenceControl** to capture a sequence of frames that you can then composite using a custom HDR implementation. For more information, see Variable photo sequence.

Initialize the scene analysis effect and add it to the preview stream

Video effects are implemented using two APIs, an effect definition, which provides settings that the capture device needs to initialize the effect, and an effect instance, which can be used to control the effect. Since you may want to access the effect instance from multiple places within your code, you should typically declare a member variable to hold the object.

C#

```
private SceneAnalysisEffect m_sceneAnalysisEffect;
```

In your app, after you have initialized the **MediaCapture** object, create a new instance of SceneAnalysisEffectDefinition.

Register the effect with the capture device by calling AddVideoEffectAsync on your **MediaCapture** object, providing the **SceneAnalysisEffectDefinition** and specifying MediaStreamType.VideoPreview to indicate that the effect should be applied to the video preview stream, as opposed to the capture stream. **AddVideoEffectAsync** returns an instance of the added effect. Because this method can be used with multiple effect types, you must cast the returned instance to a SceneAnalysisEffect object.

To receive the results of the scene analysis, you must register a handler for the SceneAnalyzed event.

Currently, the scene analysis effect only includes the high dynamic range analyzer. Enable HDR analysis by setting the effect's HighDynamicRangeControl.Enabled to true.

C#

```
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml;
using System;
using Windows.Media.Devices;
using System.Linq;
using Microsoft.UI.Xaml.Input;
using System.Threading.Tasks;
using Windows.Foundation;
using Windows.Media.MediaProperties;
using Windows.Graphics.Display;
using Windows.Media.Capture;
using System.Collections.Generic;
using Windows.Media.Capture.Frames;
using Windows.Media.Core;
using Windows.Media.Effects;
using Windows.Media;
using Windows.UI.Core;

//using MyVideoEffect;
using Windows.Graphics.Imaging;

namespace CameraWinUI
{
    public sealed partial class MainWindow : Window
    {

        #region Basic add/remove

        IVideoEffectDefinition myEffectDefinition;
        IMediaExtension myPreviewEffect;
        IMediaExtension myRecordEffect;

        private async void bBasicAddEffect_Click(object sender,
```

```
RoutedEventArgs e)
        {


            myEffectDefinition = new
VideoEffectDefinition("MyVideoEffect.ExampleVideoEffect");

            // <SnippetBasicAddEffect>
            if
(m_mediaCapture.MediaCaptureSettings.VideoDeviceCharacteristic ==
VideoDeviceCharacteristic.AllStreamsIdentical ||

m_mediaCapture.MediaCaptureSettings.VideoDeviceCharacteristic ==
VideoDeviceCharacteristic.PreviewRecordStreamsIdentical)
            {
                // This effect will modify both the preview and the record
streams, because they are the same stream.
                myRecordEffect = await
m_mediaCapture.AddVideoEffectAsync(myEffectDefinition,
MediaStreamType.VideoRecord);
            }
            else
            {
                myRecordEffect = await
m_mediaCapture.AddVideoEffectAsync(myEffectDefinition,
MediaStreamType.VideoRecord);
                myPreviewEffect = await
m_mediaCapture.AddVideoEffectAsync(myEffectDefinition,
MediaStreamType.VideoPreview);
            }
            // </SnippetBasicAddEffect>
        }
        public async void RemoveOneEffect()
        {
            // <SnippetRemoveOneEffect>
            if (myRecordEffect != null)
            {
                await m_mediaCapture.RemoveEffectAsync(myRecordEffect);
            }
            if (myPreviewEffect != null)
            {
                await m_mediaCapture.RemoveEffectAsync(myPreviewEffect);
            }
            // </SnippetRemoveOneEffect>
        }
        public async void RemoveAllEffects()
        {
            // <SnippetClearAllEffects>
            await
m_mediaCapture.ClearEffectsAsync(MediaStreamType.VideoPreview);
            await
m_mediaCapture.ClearEffectsAsync(MediaStreamType.VideoRecord);
            // </SnippetClearAllEffects>
        }
```

```
        #endregion

        #region Video stabilization effect



        // <SnippetDeclareVideoStabilizationEffect>
        //
        private VideoStabilizationEffect m_videoStabilizationEffect;
        private VideoEncodingProperties m_inputPropertiesBackup;
        private VideoEncodingProperties m_outputPropertiesBackup;
        private MediaEncodingProfile m_encodingProfile;
        // </SnippetDeclareVideoStabilizationEffect>


        private async void bSetupVideoStabilizationEffect_Click(object
sender, RoutedEventArgs e)
        {

            // <SnippetEncodingProfileMember>
            m_encodingProfile =
MediaEncodingProfile.CreateMp4(VideoEncodingQuality.Auto);
            // </SnippetEncodingProfileMember>

            // <SnippetCreateVideoStabilizationEffect>
            // Create the effect definition
            VideoStabilizationEffectDefinition stabilizerDefinition = new
VideoStabilizationEffectDefinition();

            // Add the video stabilization effect to media capture
            m_videoStabilizationEffect =
                (VideoStabilizationEffect)await
m_mediaCapture.AddVideoEffectAsync(stabilizerDefinition,
MediaStreamType.VideoRecord);

            m_videoStabilizationEffect.EnabledChanged +=
VideoStabilizationEffect_EnabledChanged;

            await SetUpVideoStabilizationRecommendationAsync();

            m_videoStabilizationEffect.Enabled = true;
            // </SnippetCreateVideoStabilizationEffect>



        }
        // <SnippetSetUpVideoStabilizationRecommendationAsync>
        private async Task SetUpVideoStabilizationRecommendationAsync()
        {

            // Get the recommendation from the effect based on our current
input and output configuration
            var recommendation =
m_videoStabilizationEffect.GetRecommendedStreamConfiguration(m_mediaCapture.
VideoDeviceController, m_encodingProfile.Video);
```

```
            // Handle the recommendation for the input into the effect,
which can contain a larger resolution than currently configured, so cropping
is minimized
            if (recommendation.InputProperties != null)
            {
                // Back up the current input properties from before VS was
activated
                m_inputPropertiesBackup =
m_mediaCapture.VideoDeviceController.GetMediaStreamProperties(MediaStreamTyp
e.VideoRecord) as VideoEncodingProperties;

                // Set the recommendation from the effect (a resolution
higher than the current one to allow for cropping) on the input
                await
m_mediaCapture.VideoDeviceController.SetMediaStreamPropertiesAsync(MediaStre
amType.VideoRecord, recommendation.InputProperties);
                await
m_mediaCapture.VideoDeviceController.SetMediaStreamPropertiesAsync(MediaStre
amType.VideoPreview, recommendation.InputProperties);
            }

            // Handle the recommendations for the output from the effect
            if (recommendation.OutputProperties != null)
            {
                // Back up the current output properties from before VS was
activated
                m_outputPropertiesBackup = m_encodingProfile.Video;

                // Apply the recommended encoding profile for the output
                m_encodingProfile.Video = recommendation.OutputProperties;
            }
        }
        // </SnippetSetUpVideoStabilizationRecommendationAsync>
        // <SnippetVideoStabilizationEnabledChanged>
        private async void
VideoStabilizationEffect_EnabledChanged(VideoStabilizationEffect sender,
VideoStabilizationEffectEnabledChangedEventArgs args)
        {
            await Dispatcher.RunAsync(CoreDispatcherPriority.Normal, () =>
            {
                // Update your UI to reflect the change in status
                tbStatus.Text = "video stabilization status: " +
sender.Enabled + ". Reason: " + args.Reason;
            });
        }
        // </SnippetVideoStabilizationEnabledChanged>
        private async void bCleanupVideoStabilizationEffect_Click(object
sender, RoutedEventArgs e)
        {
            // <SnippetCleanUpVisualStabilizationEffect>
            // Clear all effects in the pipeline
            await
m_mediaCapture.RemoveEffectAsync(m_videoStabilizationEffect);
```

```
            // If backed up settings (stream properties and encoding
profile) exist, restore them and clear the backups
            if (m_inputPropertiesBackup != null)
            {
                await
m_mediaCapture.VideoDeviceController.SetMediaStreamPropertiesAsync(MediaStre
amType.VideoRecord, m_inputPropertiesBackup);
                m_inputPropertiesBackup = null;
            }

            if (m_outputPropertiesBackup != null)
            {
                m_encodingProfile.Video = m_outputPropertiesBackup;
                m_outputPropertiesBackup = null;
            }

            m_videoStabilizationEffect.EnabledChanged -=
VideoStabilizationEffect_EnabledChanged;

            m_videoStabilizationEffect = null;
            // </SnippetCleanUpVisualStabilizationEffect>
        }

        #endregion Video stabilization effect

        #region scene analyis effect
        // <SnippetDeclareSceneAnalysisEffect>
        private SceneAnalysisEffect m_sceneAnalysisEffect;
        // </SnippetDeclareSceneAnalysisEffect>

        private async void bCreateSceneAnalysisEffect_Click(object sender,
RoutedEventArgs e)
        {
            // <SnippetCreateSceneAnalysisEffectAsync>
            // Create the definition
            var definition = new SceneAnalysisEffectDefinition();

            // Add the effect to the video record stream
            m_sceneAnalysisEffect = (SceneAnalysisEffect)await
m_mediaCapture.AddVideoEffectAsync(definition,
MediaStreamType.VideoPreview);

            // Subscribe to notifications about scene information
            m_sceneAnalysisEffect.SceneAnalyzed +=
SceneAnalysisEffect_SceneAnalyzed;

            // Enable HDR analysis
            m_sceneAnalysisEffect.HighDynamicRangeAnalyzer.Enabled = true;
            // </SnippetCreateSceneAnalysisEffectAsync>

        }

        double MyCertaintyCap = .5;
        // <SnippetSceneAnalyzed>
        private void SceneAnalysisEffect_SceneAnalyzed(SceneAnalysisEffect
```

```
sender, SceneAnalyzedEventArgs args)
        {
            double hdrCertainty =
args.ResultFrame.HighDynamicRange.Certainty;

            // Certainty value is between 0.0 and 1.0
            if (hdrCertainty > MyCertaintyCap)
            {
                DispatcherQueue.TryEnqueue(() =>
                {
                    tbStatus.Text = "Enabling HDR capture is recommended.";
                });
            }
        }
        // </SnippetSceneAnalyzed>

        private async void bCleanupSceneAnalysisEffect_Click(object sender,
RoutedEventArgs e)
        {
            // <SnippetCleanUpSceneAnalysisEffectAsync>
            // Disable detection
            m_sceneAnalysisEffect.HighDynamicRangeAnalyzer.Enabled = false;

            m_sceneAnalysisEffect.SceneAnalyzed -=
SceneAnalysisEffect_SceneAnalyzed;

            // Remove the effect from the preview stream
            await
m_mediaCapture.ClearEffectsAsync(MediaStreamType.VideoPreview);

            // Clear the member variable that held the effect instance
            m_sceneAnalysisEffect = null;
            // </SnippetCleanUpSceneAnalysisEffectAsync>
        }

        #endregion scene analyis effect


        #region Face detection

        // <SnippetDeclareFaceDetectionEffect>
        FaceDetectionEffect m_faceDetectionEffect;
        // </SnippetDeclareFaceDetectionEffect>




        private async void bCreateFaceDetectionEffect_Click(object sender,
RoutedEventArgs e)
        {
            // <SnippetCreateFaceDetectionEffectAsync>

            // Create the definition, which will contain some initialization
settings
```

```
            var definition = new FaceDetectionEffectDefinition();

            // To ensure preview smoothness, do not delay incoming samples
            definition.SynchronousDetectionEnabled = false;

            // In this scenario, choose detection speed over accuracy
            definition.DetectionMode = FaceDetectionMode.HighPerformance;

            // Add the effect to the preview stream
            m_faceDetectionEffect = (FaceDetectionEffect)await
m_mediaCapture.AddVideoEffectAsync(definition,
MediaStreamType.VideoPreview);

            // Choose the shortest interval between detection events
            m_faceDetectionEffect.DesiredDetectionInterval =
TimeSpan.FromMilliseconds(33);

            // Start detecting faces
            m_faceDetectionEffect.Enabled = true;

            // </SnippetCreateFaceDetectionEffectAsync>


            // <SnippetRegisterFaceDetectionHandler>
            // Register for face detection events
            m_faceDetectionEffect.FaceDetected +=
FaceDetectionEffect_FaceDetected;
            // </SnippetRegisterFaceDetectionHandler>


            // <SnippetAreFaceFocusAndExposureSupported>
            var regionsControl =
m_mediaCapture.VideoDeviceController.RegionsOfInterestControl;
            bool faceDetectionFocusAndExposureSupported =
                regionsControl.MaxRegions > 0 &&
                (regionsControl.AutoExposureSupported ||
regionsControl.AutoFocusSupported);
            // </SnippetAreFaceFocusAndExposureSupported>
        }

        private async void bCleanipFaceDetectionEffect_Click(object sender,
RoutedEventArgs e)
        {
            // <SnippetCleanUpFaceDetectionEffectAsync>
            // Disable detection
            m_faceDetectionEffect.Enabled = false;

            // Unregister the event handler
            m_faceDetectionEffect.FaceDetected -=
FaceDetectionEffect_FaceDetected;

            // Remove the effect from the preview stream
            await
m_mediaCapture.ClearEffectsAsync(MediaStreamType.VideoPreview);
```

```
            // Clear the member variable that held the effect instance
            m_faceDetectionEffect = null;
            // </SnippetCleanUpFaceDetectionEffectAsync>
        }



        // <SnippetFaceDetected>
        private void FaceDetectionEffect_FaceDetected(FaceDetectionEffect
sender, FaceDetectedEventArgs args)
        {
            foreach (Windows.Media.FaceAnalysis.DetectedFace face in
args.ResultFrame.DetectedFaces)
            {
                BitmapBounds faceRect = face.FaceBox;

                // Draw a rectangle on the preview stream for each face
            }
        }
        // </SnippetFaceDetected>

        #endregion Face detection


    }
}
```

Implement the SceneAnalyzed event handler

The results of the scene analysis are returned in the **SceneAnalyzed** event handler. The SceneAnalyzedEventArgs object passed into the handler has a SceneAnalysisEffectFrame object which has a HighDynamicRangeOutput object. The Certainty property of the high dynamic range output provides a value between 0 and 1.0 where 0 indicates that HDR processing would not help improve the capture result and 1.0 indicates that HDR processing would help. You can decide the threshold point at which you want to use HDR or show the results to the user and let the user decide.

C#

```
private void SceneAnalysisEffect_SceneAnalyzed(SceneAnalysisEffect sender,
SceneAnalyzedEventArgs args)
{
    double hdrCertainty = args.ResultFrame.HighDynamicRange.Certainty;

    // Certainty value is between 0.0 and 1.0
    if (hdrCertainty > MyCertaintyCap)
    {
        DispatcherQueue.TryEnqueue(() =>
        {
            tbStatus.Text = "Enabling HDR capture is recommended.";
```

```
        });
    }
}
```

The **HighDynamicRangeOutput** object passed into the handler also has a **FrameControllers** property which contains suggested frame controllers for capturing a variable photo sequence for HDR processing. For more information, see Variable photo sequence.

Clean up the scene analysis effect

When your app is done capturing, before disposing of the **MediaCapture** object, you should disable the scene analysis effect by setting the effect's **HighDynamicRangeAnalyzer.Enabled** property to false and unregister your **SceneAnalyzed** event handler. Call **MediaCapture.ClearEffectsAsync**, specifying the video preview stream since that was the stream to which the effect was added. Finally, set your member variable to null.

C#

```
// Disable detection
m_sceneAnalysisEffect.HighDynamicRangeAnalyzer.Enabled = false;

m_sceneAnalysisEffect.SceneAnalyzed -= SceneAnalysisEffect_SceneAnalyzed;

// Remove the effect from the preview stream
await m_mediaCapture.ClearEffectsAsync(MediaStreamType.VideoPreview);

// Clear the member variable that held the effect instance
m_sceneAnalysisEffect = null;
```

Face detection effect

The **FaceDetectionEffect** identifies the location of faces within the media capture preview stream. The effect allows you to receive a notification whenever a face is detected in the preview stream and provides the bounding box for each detected face within the preview frame. On supported devices, the face detection effect also provides enhanced exposure and focus on the most important face in the scene.

Initialize the face detection effect and add it to the preview stream

Video effects are implemented using two APIs, an effect definition, which provides settings that the capture device needs to initialize the effect, and an effect instance, which can be used to control the effect. Since you may want to access the effect instance from multiple places within your code, you should typically declare a member variable to hold the object.

C#

```
FaceDetectionEffect m_faceDetectionEffect;
```

In your app, after you have initialized the **MediaCapture** object, create a new instance of FaceDetectionEffectDefinition. Set the DetectionMode property to prioritize faster face detection or more accurate face detection. Set SynchronousDetectionEnabled to specify that incoming frames are not delayed waiting for face detection to complete as this can result in a choppy preview experience.

Register the effect with the capture device by calling AddVideoEffectAsync on your **MediaCapture** object, providing the **FaceDetectionEffectDefinition** and specifying MediaStreamType.VideoPreview to indicate that the effect should be applied to the video preview stream, as opposed to the capture stream. **AddVideoEffectAsync** returns an instance of the added effect. Because this method can be used with multiple effect types, you must cast the returned instance to a FaceDetectionEffect object.

Enable or disable the effect by setting the FaceDetectionEffect.Enabled property. Adjust how often the effect analyzes frames by setting the FaceDetectionEffect.DesiredDetectionInterval property. Both of these properties can be adjusted while media capture is ongoing.

C#

```
// Create the definition, which will contain some initialization settings
var definition = new FaceDetectionEffectDefinition();

// To ensure preview smoothness, do not delay incoming samples
definition.SynchronousDetectionEnabled = false;

// In this scenario, choose detection speed over accuracy
definition.DetectionMode = FaceDetectionMode.HighPerformance;

// Add the effect to the preview stream
m_faceDetectionEffect = (FaceDetectionEffect)await
m_mediaCapture.AddVideoEffectAsync(definition,
MediaStreamType.VideoPreview);

// Choose the shortest interval between detection events
m_faceDetectionEffect.DesiredDetectionInterval =
```

```
TimeSpan.FromMilliseconds(33);

// Start detecting faces
m_faceDetectionEffect.Enabled = true;
```

Receive notifications when faces are detected

If you want to perform some action when faces are detected, such as drawing a box around detected faces in the video preview, you can register for the **FaceDetected** event.

C#

```
// Register for face detection events
m_faceDetectionEffect.FaceDetected += FaceDetectionEffect_FaceDetected;
```

In the handler for the event, you can get a list of all faces detected in a frame by accessing the **FaceDetectionEffectFrame.DetectedFaces** property of the **FaceDetectedEventArgs**. The **FaceBox** property is a **BitmapBounds** structure that describes the rectangle containing the detected face in units relative to the preview stream dimensions. To view sample code that transforms the preview stream coordinates into screen coordinates, see the face detection UWP sample.

C#

```
private void FaceDetectionEffect_FaceDetected(FaceDetectionEffect sender,
FaceDetectedEventArgs args)
{
    foreach (Windows.Media.FaceAnalysis.DetectedFace face in
args.ResultFrame.DetectedFaces)
    {
        BitmapBounds faceRect = face.FaceBox;

        // Draw a rectangle on the preview stream for each face
    }
}
```

Clean up the face detection effect

When your app is done capturing, before disposing of the **MediaCapture** object, you should disable the face detection effect with **FaceDetectionEffect.Enabled** and unregister your **FaceDetected** event handler if you previously registered one. Call **MediaCapture.ClearEffectsAsync**, specifying the video preview stream since that was the stream to which the effect was added. Finally, set your member variable to null.

C#

```
// Disable detection
m_faceDetectionEffect.Enabled = false;

// Unregister the event handler
m_faceDetectionEffect.FaceDetected -= FaceDetectionEffect_FaceDetected;

// Remove the effect from the preview stream
await m_mediaCapture.ClearEffectsAsync(MediaStreamType.VideoPreview);

// Clear the member variable that held the effect instance
m_faceDetectionEffect = null;
```

Check for focus and exposure support for detected faces

Not all devices have a capture device that can adjust its focus and exposure based on detected faces. Because face detection consumes device resources, you may only want to enable face detection on devices that can use the feature to enhance capture. To see if face-based capture optimization is available, get the **VideoDeviceController** for your initialized **MediaCapture** and then get the video device controller's **RegionsOfInterestControl**. Check to see if the **MaxRegions** supports at least one region. Then check to see if either **AutoExposureSupported** or **AutoFocusSupported** are true. If these conditions are met, then the device can take advantage of face detection to enhance capture.

C#

```
var regionsControl =
m_mediaCapture.VideoDeviceController.RegionsOfInterestControl;
bool faceDetectionFocusAndExposureSupported =
    regionsControl.MaxRegions > 0 &&
    (regionsControl.AutoExposureSupported ||
regionsControl.AutoFocusSupported);
```

Related topics

- Camera
- Basic photo, video, and audio capture with MediaCapture

Variable photo sequence

Article • 01/16/2025

This article shows you how to capture a variable photo sequence, which allows you to capture multiple frames of images in rapid succession and configure each frame to use different focus, flash, ISO, exposure, and exposure compensation settings. This feature enables scenarios like creating High Dynamic Range (HDR) images.

If you want to capture HDR images but don't want to implement your own processing algorithm, you can use the **AdvancedPhotoCapture** API to use the HDR capabilities built-in to Windows. For more information, see High dynamic range (HDR) and low-light photo capture.

> **Note**
>
> This article builds on concepts and code discussed in Basic photo, video, and audio capture with MediaCapture, which describes the steps for implementing basic photo and video capture. It is recommended that you familiarize yourself with the basic media capture pattern in that article before moving on to more advanced capture scenarios. The code in this article assumes that your app already has an instance of MediaCapture that has been properly initialized.

Set up your app to use variable photo sequence capture

Declare a member variable to store the **VariablePhotoSequenceCapture** object, which is used to initiate the photo sequence capture. Declare an array of **SoftwareBitmap** objects to store each captured image in the sequence. Also, declare an array to store the **CapturedFrameControlValues** object for each frame. This can be used by your image processing algorithm to determine what settings were used to capture each frame. Finally, declare an index that will be used to track which image in the sequence is currently being captured.

C#

```
VariablePhotoSequenceCapture m_photoSequenceCapture;
SoftwareBitmap[] m_images;
CapturedFrameControlValues[] m_frameControlValues;
int m_photoIndex;
```

Prepare the variable photo sequence capture

After you have initialized your MediaCapture, make sure that variable photo sequences are supported on the current device by getting an instance of the VariablePhotoSequenceController from the media capture's VideoDeviceController and checking the Supported property.

```
C#

var varPhotoSeqController =
m_mediaCapture.VideoDeviceController.VariablePhotoSequenceController;

if (!varPhotoSeqController.Supported)
{
    tbStatus.Text = "Variable Photo Sequence is not supported";
    return;
}
```

Get a FrameControlCapabilities object from the variable photo sequence controller. This object has a property for every setting that can be configured per frame of a photo sequence. These include:

- Exposure
- ExposureCompensation
- Flash
- Focus
- IsoSpeed
- PhotoConfirmation

This example will set a different exposure compensation value for each frame. To verify that exposure compensation is supported for photo sequences on the current device, check the Supported property of the FrameExposureCompensationCapabilities object accessed through the **ExposureCompensation** property.

```
C#

var frameCapabilities = varPhotoSeqController.FrameCapabilities;

if (!frameCapabilities.ExposureCompensation.Supported)
{
    tbStatus.Text = "EVCompenstaion is not supported in FrameController";
    return;
}
```

Create a new **FrameController** object for each frame you want to capture. This example captures three frames. Set the values for the controls you want to vary for each frame. Then, clear the **DesiredFrameControllers** collection of the **VariablePhotoSequenceController** and add each frame controller to the collection.

C#

```
var frame0 = new FrameController();
var frame1 = new FrameController();
var frame2 = new FrameController();

frame0.ExposureCompensationControl.Value = -1.0f;
frame1.ExposureCompensationControl.Value = 0.0f;
frame2.ExposureCompensationControl.Value = 1.0f;

varPhotoSeqController.DesiredFrameControllers.Clear();
varPhotoSeqController.DesiredFrameControllers.Add(frame0);
varPhotoSeqController.DesiredFrameControllers.Add(frame1);
varPhotoSeqController.DesiredFrameControllers.Add(frame2);
```

Create an **ImageEncodingProperties** object to set the encoding you want to use for the captured images. Call the static method **MediaCapture.PrepareVariablePhotoSequenceCaptureAsync**, passing in the encoding properties. This method returns a **VariablePhotoSequenceCapture** object. Finally, register event handlers for the **PhotoCaptured** and **Stopped** events.

C#

```
try
{
    var imageEncodingProperties = ImageEncodingProperties.CreateJpeg();

    m_photoSequenceCapture = await
m_mediaCapture.PrepareVariablePhotoSequenceCaptureAsync(imageEncodingPropert
ies);

    m_photoSequenceCapture.PhotoCaptured += OnPhotoCaptured;
    m_photoSequenceCapture.Stopped += OnStopped;
}
catch (Exception ex)
{
    tbStatus.Text = "Exception in PrepareVariablePhotoSequence: " +
ex.Message;
}
```

Start the variable photo sequence capture

To start the capture of the variable photo sequence, call VariablePhotoSequenceCapture.StartAsync. Be sure to initialize the arrays for storing the captured images and frame control values and set the current index to 0. Set your app's recording state variable and update your UI to disable starting another capture while this capture is in progress.

C#

```
private async void bStartVPS_Click(object sender, RoutedEventArgs e)
{
    m_images = new SoftwareBitmap[3];
    m_frameControlValues = new CapturedFrameControlValues[3];
    m_photoIndex = 0;
    m_isRecording = true;

    await m_photoSequenceCapture.StartAsync();
}
```

Receive the captured frames

The PhotoCaptured event is raised for each captured frame. Save the frame control values and captured image for the frame and then increment the current frame index. This example shows how to get a SoftwareBitmap representation of each frame. For more information on using **SoftwareBitmap**, see Imaging.

C#

```
void OnPhotoCaptured(VariablePhotoSequenceCapture s,
VariablePhotoCapturedEventArgs args)
{

    m_images[m_photoIndex] = args.Frame.SoftwareBitmap;
    m_frameControlValues[m_photoIndex] = args.CapturedFrameControlValues;
    m_photoIndex++;
}
```

Handle the completion of the variable photo sequence capture

The Stopped event is raised when all of the frames in the sequence have been captured. Update the recording state of your app and update your UI to allow the user to initiate new captures. At this point, you can pass the captured images and frame control values to your image processing code.

C#

```
void OnStopped(object s, object e)
{
    m_isRecording = false;
    MyPostProcessingFunction(m_images, m_frameControlValues, 3);
}
```

Update frame controllers

If you want to perform another variable photo sequence capture with different per frame settings, you don't need to completely reinitialize the **VariablePhotoSequenceCapture**. You can either clear the DesiredFrameControllers collection and add new frame controllers or you can modify the existing frame controller values. The following example checks the FrameFlashCapabilities object to verify that the current device supports flash and flash power for variable photo sequence frames. If so, each frame is updated to enable the flash at 100% power. The exposure compensation values that were previously set for each frame are still active.

C#

```
var varPhotoSeqController =
m_mediaCapture.VideoDeviceController.VariablePhotoSequenceController;

if (varPhotoSeqController.FrameCapabilities.Flash.Supported &&
    varPhotoSeqController.FrameCapabilities.Flash.PowerSupported)
{
    for (int i = 0; i < varPhotoSeqController.DesiredFrameControllers.Count;
i++)
    {
        varPhotoSeqController.DesiredFrameControllers[i].FlashControl.Mode =
FrameFlashMode.Enable;

varPhotoSeqController.DesiredFrameControllers[i].FlashControl.PowerPercent =
100;
    }
}
```

Clean up the variable photo sequence capture

When you are done capturing variable photo sequences or your app is suspending, clean up the variable photo sequence object by calling FinishAsync. Unregister the object's event handlers and set it to null.

C#

```
await m_photoSequenceCapture.FinishAsync();
m_photoSequenceCapture.PhotoCaptured -= OnPhotoCaptured;
m_photoSequenceCapture.Stopped -= OnStopped;
m_photoSequenceCapture = null;
```

Related topics

- Camera
- Basic photo, video, and audio capture with MediaCapture

Process media frames with MediaFrameReader

Article • 01/16/2025

This article shows you how to use a MediaFrameReader with MediaCapture to get media frames from one or more available sources, including color, depth, and infrared cameras, audio devices, or even custom frame sources such as those that produce skeletal tracking frames. This feature is designed to be used by apps that perform real-time processing of media frames, such as augmented reality and depth-aware camera apps.

If you are interested in simply capturing video or photos, such as a typical photography app, then you probably want to use one of the other capture techniques supported by MediaCapture. For a list of available media capture techniques and articles showing how to use them, see Camera.

> **Note**
>
> The features discussed in this article are only available starting with Windows 10, version 1607.

> **Note**
>
> There is a Universal Windows app sample that demonstrates using **MediaFrameReader** to display frames from different frame sources, including color, depth, and infrared cameras. For more information, see Camera frames sample.

> **Note**
>
> A new set of APIs for using **MediaFrameReader** with audio data were introduced in Windows 10, version 1803. For more information, see Process audio frames with MediaFrameReader.

Select frame sources and frame source groups

Many apps that process media frames need to get frames from multiple sources at once, such as a device's color and depth cameras. The MediaFrameSourceGroup object

represents a set of media frame sources that can be used simultaneously. Call the static method MediaFrameSourceGroup.FindAllAsync to get a list of all of the groups of frame sources supported by the current device.

C#

```
var frameSourceGroups = await MediaFrameSourceGroup.FindAllAsync();
```

You can also create a DeviceWatcher using DeviceInformation.CreateWatcher and the value returned from MediaFrameSourceGroup.GetDeviceSelector to receive notifications when the available frame source groups on the device changes, such as when an external camera is plugged in. For more information see Enumerate devices.

A MediaFrameSourceGroup has a collection of MediaFrameSourceInfo objects that describe the frame sources included in the group. After retrieving the frame source groups available on the device, you can select the group that exposes the frame sources you are interested in.

The following example shows the simplest way to select a frame source group. This code simply loops over all of the available groups and then loops over each item in the SourceInfos collection. Each **MediaFrameSourceInfo** is checked to see if it supports the features we are seeking. In this case, the MediaStreamType property is checked for the value VideoPreview, meaning the device provides a video preview stream, and the SourceKind property is checked for the value Color, indicating that the source provides color frames.

C#

```
var frameSourceGroups = await MediaFrameSourceGroup.FindAllAsync();

MediaFrameSourceGroup selectedGroup = null;
MediaFrameSourceInfo colorSourceInfo = null;

foreach (var sourceGroup in frameSourceGroups)
{
    foreach (var sourceInfo in sourceGroup.SourceInfos)
    {
        if (sourceInfo.MediaStreamType == MediaStreamType.VideoPreview
            && sourceInfo.SourceKind == MediaFrameSourceKind.Color)
        {
            colorSourceInfo = sourceInfo;
            break;
        }
    }
    if (colorSourceInfo != null)
    {
        selectedGroup = sourceGroup;
```

```
            break;
        }
    }
```

This method of identifying the desired frame source group and frame sources works for simple cases, but if you want to select frame sources based on more complex criteria, it can quickly become cumbersome. Another method is to use Linq syntax and anonymous objects to make the selection. The following example uses the **Select** extension method to transform the **MediaFrameSourceGroup** objects in the *frameSourceGroups* list into an anonymous object with two fields: *sourceGroup*, representing the group itself, and *colorSourceInfo*, which represents the color frame source in the group. The *colorSourceInfo* field is set to the result of **FirstOrDefault**, which selects the first object for which the provided predicate resolves to true. In this case, the predicate is true if the stream type is **VideoPreview**, the source kind is **Color**, and if the camera is on the front panel of the device.

From the list of anonymous objects returned from the query described above, the **Where** extension method is used to select only those objects where the *colorSourceInfo* field is not null. Finally, **FirstOrDefault** is called to select the first item in the list.

Now you can use the fields of the selected object to get references to the selected **MediaFrameSourceGroup** and the **MediaFrameSourceInfo** object representing the color camera. These will be used later to initialize the **MediaCapture** object and create a **MediaFrameReader** for the selected source. Finally, you should test to see if the source group is null, meaning the current device doesn't have your requested capture sources.

C#

```
var selectedGroupObjects = frameSourceGroups.Select(group =>
   new
   {
       sourceGroup = group,
       colorSourceInfo = group.SourceInfos.FirstOrDefault((sourceInfo) =>
       {
           // On Xbox/Kinect, omit the MediaStreamType and EnclosureLocation
tests
           return sourceInfo.MediaStreamType == MediaStreamType.VideoPreview
           && sourceInfo.SourceKind == MediaFrameSourceKind.Color
           && sourceInfo.DeviceInformation?.EnclosureLocation.Panel ==
Windows.Devices.Enumeration.Panel.Front;
       })

   }).Where(t => t.colorSourceInfo != null)
   .FirstOrDefault();

MediaFrameSourceGroup selectedGroup = selectedGroupObjects?.sourceGroup;
MediaFrameSourceInfo colorSourceInfo =
```

```
selectedGroupObjects?.colorSourceInfo;

if (selectedGroup == null)
{
    return;
}
```

The following example uses a similar technique as described above to select a source group that contains color, depth, and infrared cameras.

C#

```
var allGroups = await MediaFrameSourceGroup.FindAllAsync();
var eligibleGroups = allGroups.Select(g => new
{
    Group = g,

    // For each source kind, find the source which offers that kind of media
frame,
    // or null if there is no such source.
    SourceInfos = new MediaFrameSourceInfo[]
    {
        g.SourceInfos.FirstOrDefault(info => info.SourceKind ==
MediaFrameSourceKind.Color),
        g.SourceInfos.FirstOrDefault(info => info.SourceKind ==
MediaFrameSourceKind.Depth),
        g.SourceInfos.FirstOrDefault(info => info.SourceKind ==
MediaFrameSourceKind.Infrared),
    }
}).Where(g => g.SourceInfos.Any(info => info != null)).ToList();

if (eligibleGroups.Count == 0)
{
    System.Diagnostics.Debug.WriteLine("No source group with color, depth or
infrared found.");
    return;
}

var selectedGroupIndex = 0; // Select the first eligible group
MediaFrameSourceGroup selectedGroup =
eligibleGroups[selectedGroupIndex].Group;
MediaFrameSourceInfo colorSourceInfo =
eligibleGroups[selectedGroupIndex].SourceInfos[0];
MediaFrameSourceInfo infraredSourceInfo =
eligibleGroups[selectedGroupIndex].SourceInfos[1];
MediaFrameSourceInfo depthSourceInfo =
eligibleGroups[selectedGroupIndex].SourceInfos[2];
```

> ⓘ Note

Starting with Windows 10, version 1803, you can use the MediaCaptureVideoProfile class to select a media frame source with a set of desired capabilities. For more information, see the section **Use video profiles to select a frame source** later in this article.

Initialize the MediaCapture object to use the selected frame source group

The next step is to initialize the **MediaCapture** object to use the frame source group you selected in the previous step. Create an instance of the **MediaCapture** object by calling the constructor. Next, create a MediaCaptureInitializationSettings object that will be used to initialize the **MediaCapture** object. In this example, the following settings are used:

- SourceGroup - This tells the system which source group you will be using to get frames. Remember that the source group defines a set of media frame sources that can be used simultaneously.
- SharingMode - This tells the system whether you need exclusive control over the capture source devices. If you set this to ExclusiveControl, it means that you can change the settings of the capture device, such as the format of the frames it produces, but this means that if another app already has exclusive control, your app will fail when it tries to initialize the media capture device. If you set this to SharedReadOnly, you can receive frames from the frame sources even if they are in use by another app, but you can't change the settings for the devices.
- MemoryPreference - If you specify CPU, the system will use CPU memory which guarantees that when frames arrive, they will be available as SoftwareBitmap objects. If you specify Auto, the system will dynamically choose the optimal memory location to store frames. If the system chooses to use GPU memory, the media frames will arrive as an IDirect3DSurface object and not as a **SoftwareBitmap**.
- StreamingCaptureMode - Set this to Video to indicate that audio doesn't need to be streamed.

Call InitializeAsync to initialize the **MediaCapture** with your desired settings. Be sure to call this within a *try* block in case initialization fails.

C#

```
m_mediaCapture = new MediaCapture();

var settings = new MediaCaptureInitializationSettings()
```

```
{
    SourceGroup = selectedGroup,
    SharingMode = MediaCaptureSharingMode.ExclusiveControl,
    MemoryPreference = MediaCaptureMemoryPreference.Cpu,
    StreamingCaptureMode = StreamingCaptureMode.Video
};
try
{
    await m_mediaCapture.InitializeAsync(settings);
}
catch (Exception ex)
{
    System.Diagnostics.Debug.WriteLine("MediaCapture initialization failed:
" + ex.Message);
    return;
}
```

Set the preferred format for the frame source

To set the preferred format for a frame source, you need to get a **MediaFrameSource** object representing the source. You get this object by accessing the **Frames** dictionary of the initialized **MediaCapture** object, specifying the identifier of the frame source you want to use. This is why we saved the **MediaFrameSourceInfo** object when we were selecting a frame source group.

The **MediaFrameSource.SupportedFormats** property contains a list of **MediaFrameFormat** objects describing the supported formats for the frame source. In this example, a format is selected that has a width of 1080 pixels and can supply frames in 32-bit RGB format. The **FirstOrDefault** extension method selects the first entry in the list. If the selected format is null, then the requested format is not supported by the frame source. If the format is supported, you can request that the source use this format by calling **SetFormatAsync**.

C#

```
var colorFrameSource = m_mediaCapture.FrameSources[colorSourceInfo.Id];
var preferredFormat = colorFrameSource.SupportedFormats.Where(format =>
{
    return format.VideoFormat.Width >= 1080
    && format.Subtype == MediaEncodingSubtypes.Argb32;

}).FirstOrDefault();

if (preferredFormat == null)
{
    // Our desired format is not supported
    return;
}
```

```
await colorFrameSource.SetFormatAsync(preferredFormat);
```

Create a frame reader for the frame source

To receive frames for a media frame source, use a MediaFrameReader.

C#

```
MediaFrameReader m_mediaFrameReader;
```

Instantiate the frame reader by calling CreateFrameReaderAsync on your initialized **MediaCapture** object. The first argument to this method is the frame source from which you want to receive frames. You can create a separate frame reader for each frame source you want to use. The second argument tells the system the output format in which you want frames to arrive. This can save you from having to do your own conversions to frames as they arrive. Note that if you specify a format that is not supported by the frame source, an exception will be thrown, so be sure that this value is in the SupportedFormats collection.

After creating the frame reader, register a handler for the FrameArrived event which is raised whenever a new frame is available from the source.

Tell the system to start reading frames from the source by calling StartAsync.

C#

```
m_mediaFrameReader = await
m_mediaCapture.CreateFrameReaderAsync(colorFrameSource,
MediaEncodingSubtypes.Argb32);
m_mediaFrameReader.FrameArrived += ColorFrameReader_FrameArrived;
await m_mediaFrameReader.StartAsync();
```

Handle the frame arrived event

The MediaFrameReader.FrameArrived event is raised whenever a new frame is available. You can choose to process every frame that arrives or only use frames when you need them. Because the frame reader raises the event on its own thread, you may need to implement some synchronization logic to make sure that you aren't attempting to access the same data from multiple threads. This section shows you how to synchronize drawing color frames to an image control in a XAML page. This scenario

addresses the additional synchronization constraint that requires all updates to XAML controls be performed on the UI thread.

The first step in displaying frames in XAML is to create an Image control.

```
XML

<Image x:Name="iFrameReaderImageControl" MaxWidth="300" MaxHeight="200"/>
```

In your code behind page, declare a class member variable of type **SoftwareBitmap** which will be used as a back buffer that all incoming images will be copied to. Note that the image data itself isn't copied, just the object references. Also, declare a boolean to track whether our UI operation is currently running.

```
C#

private SoftwareBitmap backBuffer;
private bool taskRunning = false;
```

Because the frames will arrive as **SoftwareBitmap** objects, you need to create a SoftwareBitmapSource object which allows you to use a **SoftwareBitmap** as the source for a XAML **Control**. You should set the image source somewhere in your code before you start the frame reader.

```
C#

iFrameReaderImageControl.Source = new SoftwareBitmapSource();
```

Now it's time to implement the **FrameArrived** event handler. When the handler is called, the *sender* parameter contains a reference to the **MediaFrameReader** object which raised the event. Call TryAcquireLatestFrame on this object to attempt to get the latest frame. As the name implies, **TryAcquireLatestFrame** may not succeed in returning a frame. So, when you access the VideoMediaFrame and then SoftwareBitmap properties, be sure to test for null. In this example the null conditional operator ? is used to access the **SoftwareBitmap** and then the retrieved object is checked for null.

The **Image** control can only display images in BRGA8 format with either pre-multiplied or no alpha. If the arriving frame is not in that format, the static method Convert is used to convert the software bitmap to the correct format.

Next, the Interlocked.Exchange method is used to swap the reference of to arriving bitmap with the backbuffer bitmap. This method swaps these references in an atomic

operation that is thread-safe. After swapping, the old backbuffer image, now in the *softwareBitmap* variable is disposed of to clean up its resources.

Next, the CoreDispatcher associated with the **Image** element is used to create a task that will run on the UI thread by calling RunAsync. Because the asynchronous tasks will be performed within the task, the lambda expression passed to **RunAsync** is declared with the *async* keyword.

Within the task, the *_taskRunning* variable is checked to make sure that only one instance of the task is running at a time. If the task isn't already running, *_taskRunning* is set to true to prevent the task from running again. In a *while* loop, **Interlocked.Exchange** is called to copy from the backbuffer into a temporary **SoftwareBitmap** until the backbuffer image is null. For each time the temporary bitmap is populated, the **Source** property of the **Image** is cast to a **SoftwareBitmapSource**, and then SetBitmapAsync is called to set the source of the image.

Finally, the *_taskRunning* variable is set back to false so that the task can be run again the next time the handler is called.

> **Note**
>
> If you access the SoftwareBitmap or Direct3DSurface objects provided by the VideoMediaFrame property of a MediaFrameReference, the system creates a strong reference to these objects, which means that they will not be disposed when you call Dispose on the containing **MediaFrameReference**. You must explicitly call the **Dispose** method of the **SoftwareBitmap** or **Direct3DSurface** directly for the objects to be immediately disposed. Otherwise, the garbage collector will eventually free the memory for these objects, but you can't know when this will occur, and if the number of allocated bitmaps or surfaces exceeds the maximum amount allowed by the system, the flow of new frames will stop. You can copy retrieved frames, using the SoftwareBitmap.Copy method for example, and then release the original frames to overcome this limitation. Also, if you create the **MediaFrameReader** using the overload CreateFrameReaderAsync(Windows.Media.Capture.Frames.MediaFrameSource inputSource, System.String outputSubtype, Windows.Graphics.Imaging.BitmapSize outputSize) or CreateFrameReaderAsync(Windows.Media.Capture.Frames.MediaFrameSource inputSource, System.String outputSubtype), the frames returned are copies of the original frame data and so they do not cause frame acquisition to halt when they are retained.

C#

```
private void ColorFrameReader_FrameArrived(MediaFrameReader sender,
MediaFrameArrivedEventArgs args)
{
    var mediaFrameReference = sender.TryAcquireLatestFrame();
    var videoMediaFrame = mediaFrameReference?.VideoMediaFrame;
    var softwareBitmap = videoMediaFrame?.SoftwareBitmap;

    if (softwareBitmap != null)
    {
        if (softwareBitmap.BitmapPixelFormat !=
Windows.Graphics.Imaging.BitmapPixelFormat.Bgra8 ||
            softwareBitmap.BitmapAlphaMode !=
Windows.Graphics.Imaging.BitmapAlphaMode.Premultiplied)
        {
            softwareBitmap = SoftwareBitmap.Convert(softwareBitmap,
BitmapPixelFormat.Bgra8, BitmapAlphaMode.Premultiplied);
        }

        // Swap the processed frame to _backBuffer and dispose of the unused
image.
        softwareBitmap = Interlocked.Exchange(ref backBuffer,
softwareBitmap);
        softwareBitmap?.Dispose();

        // Changes to XAML ImageElement must happen on UI thread through
Dispatcher
        var task = iFrameReaderImageControl.DispatcherQueue.TryEnqueue(
            async () =>
            {
                // Don't let two copies of this task run at the same time.
                if (taskRunning)
                {
                    return;
                }
                taskRunning = true;

                // Keep draining frames from the backbuffer until the
backbuffer is empty.
                SoftwareBitmap latestBitmap;
                while ((latestBitmap = Interlocked.Exchange(ref backBuffer,
null)) != null)
                {
                    var imageSource =
(SoftwareBitmapSource)iFrameReaderImageControl.Source;
                    await imageSource.SetBitmapAsync(latestBitmap);
                    latestBitmap.Dispose();
                }

                taskRunning = false;
            });
    }
```

```
    if (mediaFrameReference != null)
    {
        mediaFrameReference.Dispose();
    }
}
```

Cleanup resources

When you are done reading frames, be sure to stop the media frame reader by calling StopAsync, unregistering the **FrameArrived** handler, and disposing of the **MediaCapture** object.

C#

```
await m_mediaFrameReader.StopAsync();
m_mediaFrameReader.FrameArrived -= ColorFrameReader_FrameArrived;
m_mediaCapture.Dispose();
m_mediaCapture = null;
```

For more information about cleaning up media capture objects when your application is suspended, see **Show the camera preview in a WinUI 3 app**.

The FrameRenderer helper class

This section provides the full code listing for a provides a helper class that makes it easy to display the frames from color, infrared, and depth sources in your app. Typically, you will want to do something more with depth and infrared data than just display it to the screen, but this helper class is a helpful tool for demonstrating the frame reader feature and for debugging your own frame reader implementation. The code in this section is adapted from the Camera frames sample.

The **FrameRenderer** helper class implements the following methods.

- **FrameRenderer** constructor - The constructor initializes the helper class to use the XAML **Image** element you pass in for displaying media frames.
- **ProcessFrame** - This method displays a media frame, represented by a **MediaFrameReference**, in the **Image** element you passed into the constructor. You should typically call this method from your **FrameArrived** event handler, passing in the frame returned by **TryAcquireLatestFrame**.
- **ConvertToDisplayableImage** - This methods checks the format of the media frame and, if necessary, converts it to a displayable format. For color images, this means making sure that the color format is BGRA8 and that the bitmap alpha mode is

premultiplied. For depth or infrared frames, each scanline is processed to convert the depth or infrared values to a psuedocolor gradient, using the **PsuedoColorHelper** class that is also included in the sample and listed below.

> **Note**
>
> In order to do pixel manipulation on **SoftwareBitmap** images, you must access a native memory buffer. To do this, you must use the IMemoryBufferByteAccess COM interface included in the code listing below and you must update your project properties to allow compilation of unsafe code. For more information, see Create, edit, and save bitmap images.

C#

```
[GeneratedComInterface]
[Guid("5B0D3235-4DBA-4D44-865E-8F1D0E4FD04D")]
[InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
unsafe partial interface IMemoryBufferByteAccess
{
    void GetBuffer(out byte* buffer, out uint capacity);
}
```

C#

```
class FrameRenderer
{
    private Image m_imageElement;
    private SoftwareBitmap m_backBuffer;
    private bool _taskRunning = false;

    public FrameRenderer(Image imageElement)
    {
        m_imageElement = imageElement;
        m_imageElement.Source = new SoftwareBitmapSource();
    }

    // Processes a MediaFrameReference and displays it in a XAML image
control
    public void ProcessFrame(MediaFrameReference frame)
    {
        var softwareBitmap =
FrameRenderer.ConvertToDisplayableImage(frame?.VideoMediaFrame);
        if (softwareBitmap != null)
        {
            // Swap the processed frame to m_backBuffer and trigger UI
thread to render it
            softwareBitmap = Interlocked.Exchange(ref m_backBuffer,
softwareBitmap);
```

```
            // UI thread always reset m_backBuffer before using it.  Unused
bitmap should be disposed.
            softwareBitmap?.Dispose();

            // Changes to xaml ImageElement must happen in UI thread through
Dispatcher
            var task = m_imageElement.DispatcherQueue.TryEnqueue(
                async () =>
                {
                    // Don't let two copies of this task run at the same
time.
                    if (_taskRunning)
                    {
                        return;
                    }
                    _taskRunning = true;

                    // Keep draining frames from the backbuffer until the
backbuffer is empty.
                    SoftwareBitmap latestBitmap;
                    while ((latestBitmap = Interlocked.Exchange(ref
m_backBuffer, null)) != null)
                    {
                        var imageSource =
(SoftwareBitmapSource)m_imageElement.Source;
                        await imageSource.SetBitmapAsync(latestBitmap);
                        latestBitmap.Dispose();
                    }

                    _taskRunning = false;
                });
        }
    }



    // Function delegate that transforms a scanline from an input image to
an output image.
    private unsafe delegate void TransformScanline(int pixelWidth, byte*
inputRowBytes, byte* outputRowBytes);
    /// <summary>
    /// Determines the subtype to request from the MediaFrameReader that
will result in
    /// a frame that can be rendered by ConvertToDisplayableImage.
    /// </summary>
    /// <returns>Subtype string to request, or null if subtype is not
renderable.</returns>

    public static string GetSubtypeForFrameReader(MediaFrameSourceKind kind,
MediaFrameFormat format)
    {
        // Note that media encoding subtypes may differ in case.
        // https://docs.microsoft.com/en-
us/uwp/api/Windows.Media.MediaProperties.MediaEncodingSubtypes
```

```
        string subtype = format.Subtype;
        switch (kind)
        {
            // For color sources, we accept anything and request that it be
converted to Bgra8.
            case MediaFrameSourceKind.Color:
                return
Windows.Media.MediaProperties.MediaEncodingSubtypes.Bgra8;

            // The only depth format we can render is D16.
            case MediaFrameSourceKind.Depth:
                return String.Equals(subtype,
Windows.Media.MediaProperties.MediaEncodingSubtypes.D16,
StringComparison.OrdinalIgnoreCase) ? subtype : null;

            // The only infrared formats we can render are L8 and L16.
            case MediaFrameSourceKind.Infrared:
                return (String.Equals(subtype,
Windows.Media.MediaProperties.MediaEncodingSubtypes.L8,
StringComparison.OrdinalIgnoreCase) ||
                    String.Equals(subtype,
Windows.Media.MediaProperties.MediaEncodingSubtypes.L16,
StringComparison.OrdinalIgnoreCase)) ? subtype : null;

            // No other source kinds are supported by this class.
            default:
                return null;
        }
    }

    /// <summary>
    /// Converts a frame to a SoftwareBitmap of a valid format to display in
an Image control.
    /// </summary>
    /// <param name="inputFrame">Frame to convert.</param>

    public static unsafe SoftwareBitmap
ConvertToDisplayableImage(VideoMediaFrame inputFrame)
    {
        SoftwareBitmap result = null;
        using (var inputBitmap = inputFrame?.SoftwareBitmap)
        {
            if (inputBitmap != null)
            {
                switch (inputFrame.FrameReference.SourceKind)
                {
                    case MediaFrameSourceKind.Color:
                        // XAML requires Bgra8 with premultiplied alpha.
                        // We requested Bgra8 from the MediaFrameReader, so
all that's
                        // left is fixing the alpha channel if necessary.
                        if (inputBitmap.BitmapPixelFormat !=
BitmapPixelFormat.Bgra8)
                        {
```

```
                            System.Diagnostics.Debug.WriteLine("Color frame
in unexpected format.");
                        }
                        else if (inputBitmap.BitmapAlphaMode ==
BitmapAlphaMode.Premultiplied)
                        {
                            // Already in the correct format.
                            result = SoftwareBitmap.Copy(inputBitmap);
                        }
                        else
                        {
                            // Convert to premultiplied alpha.
                            result = SoftwareBitmap.Convert(inputBitmap,
BitmapPixelFormat.Bgra8, BitmapAlphaMode.Premultiplied);
                        }
                        break;

                    case MediaFrameSourceKind.Depth:
                        // We requested D16 from the MediaFrameReader, so
the frame should
                        // be in Gray16 format.
                        if (inputBitmap.BitmapPixelFormat ==
BitmapPixelFormat.Gray16)
                        {
                            // Use a special pseudo color to render 16 bits
depth frame.
                            var depthScale =
(float)inputFrame.DepthMediaFrame.DepthFormat.DepthScaleInMeters;
                            var minReliableDepth =
inputFrame.DepthMediaFrame.MinReliableDepth;
                            var maxReliableDepth =
inputFrame.DepthMediaFrame.MaxReliableDepth;
                            result = TransformBitmap(inputBitmap, (w, i, o)
=> PseudoColorHelper.PseudoColorForDepth(w, i, o, depthScale,
minReliableDepth, maxReliableDepth));
                        }
                        else
                        {
                            System.Diagnostics.Debug.WriteLine("Depth frame
in unexpected format.");
                        }
                        break;

                    case MediaFrameSourceKind.Infrared:
                        // We requested L8 or L16 from the MediaFrameReader,
so the frame should
                        // be in Gray8 or Gray16 format.
                        switch (inputBitmap.BitmapPixelFormat)
                        {
                            case BitmapPixelFormat.Gray16:
                                // Use pseudo color to render 16 bits
frames.
                                result = TransformBitmap(inputBitmap,
PseudoColorHelper.PseudoColorFor16BitInfrared);
                                break;
```

```
                            case BitmapPixelFormat.Gray8:
                                // Use pseudo color to render 8 bits frames.
                                result = TransformBitmap(inputBitmap,
PseudoColorHelper.PseudoColorFor8BitInfrared);
                                break;
                            default:
                                System.Diagnostics.Debug.WriteLine("Infrared
frame in unexpected format.");
                                break;
                        }
                        break;
                }
            }
        }

        return result;
    }



    /// <summary>
    /// Transform image into Bgra8 image using given transform method.
    /// </summary>
    /// <param name="softwareBitmap">Input image to transform.</param>
    /// <param name="transformScanline">Method to map pixels in a scanline.
</param>

    private static unsafe SoftwareBitmap TransformBitmap(SoftwareBitmap
softwareBitmap, TransformScanline transformScanline)
    {
        // XAML Image control only supports premultiplied Bgra8 format.
        var outputBitmap = new SoftwareBitmap(BitmapPixelFormat.Bgra8,
            softwareBitmap.PixelWidth, softwareBitmap.PixelHeight,
BitmapAlphaMode.Premultiplied);

        using (var input =
softwareBitmap.LockBuffer(BitmapBufferAccessMode.Read))
        using (var output =
outputBitmap.LockBuffer(BitmapBufferAccessMode.Write))
        {
            // Get stride values to calculate buffer position for a given
pixel x and y position.
            int inputStride = input.GetPlaneDescription(0).Stride;
            int outputStride = output.GetPlaneDescription(0).Stride;
            int pixelWidth = softwareBitmap.PixelWidth;
            int pixelHeight = softwareBitmap.PixelHeight;

            using (var outputReference = output.CreateReference())
            using (var inputReference = input.CreateReference())
            {
                // Get input and output byte access buffers.
                byte* inputBytes;
                uint inputCapacity;
                ((IMemoryBufferByteAccess)inputReference).GetBuffer(out
```

```
inputBytes, out inputCapacity);
                byte* outputBytes;
                uint outputCapacity;
                ((IMemoryBufferByteAccess)outputReference).GetBuffer(out
outputBytes, out outputCapacity);

                // Iterate over all pixels and store converted value.
                for (int y = 0; y < pixelHeight; y++)
                {
                    byte* inputRowBytes = inputBytes + y * inputStride;
                    byte* outputRowBytes = outputBytes + y * outputStride;

                    transformScanline(pixelWidth, inputRowBytes,
outputRowBytes);
                }
            }
        }

        return outputBitmap;
    }



    /// <summary>
    /// A helper class to manage look-up-table for pseudo-colors.
    /// </summary>

    private static class PseudoColorHelper
    {
        #region Constructor, private members and methods

        private const int TableSize = 1024;   // Look up table size
        private static readonly uint[] PseudoColorTable;
        private static readonly uint[] InfraredRampTable;

        // Color palette mapping value from 0 to 1 to blue to red colors.
        private static readonly Color[] ColorRamp =
        {
            Color.FromArgb(a:0xFF, r:0x7F, g:0x00, b:0x00),
            Color.FromArgb(a:0xFF, r:0xFF, g:0x00, b:0x00),
            Color.FromArgb(a:0xFF, r:0xFF, g:0x7F, b:0x00),
            Color.FromArgb(a:0xFF, r:0xFF, g:0xFF, b:0x00),
            Color.FromArgb(a:0xFF, r:0x7F, g:0xFF, b:0x7F),
            Color.FromArgb(a:0xFF, r:0x00, g:0xFF, b:0xFF),
            Color.FromArgb(a:0xFF, r:0x00, g:0x7F, b:0xFF),
            Color.FromArgb(a:0xFF, r:0x00, g:0x00, b:0xFF),
            Color.FromArgb(a:0xFF, r:0x00, g:0x00, b:0x7F),
        };

        static PseudoColorHelper()
        {
            PseudoColorTable = InitializePseudoColorLut();
            InfraredRampTable = InitializeInfraredRampLut();
        }
```

```
        /// <summary>
        /// Maps an input infrared value between [0, 1] to corrected value
between [0, 1].
        /// </summary>
        /// <param name="value">Input value between [0, 1].</param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]  // Tell the
compiler to inline this method to improve performance

        private static uint InfraredColor(float value)
        {
            int index = (int)(value * TableSize);
            index = index < 0 ? 0 : index > TableSize - 1 ? TableSize - 1 :
index;
            return InfraredRampTable[index];
        }

        /// <summary>
        /// Initializes the pseudo-color look up table for infrared pixels
        /// </summary>

        private static uint[] InitializeInfraredRampLut()
        {
            uint[] lut = new uint[TableSize];
            for (int i = 0; i < TableSize; i++)
            {
                var value = (float)i / TableSize;
                // Adjust to increase color change between lower values in
infrared images

                var alpha = (float)Math.Pow(1 - value, 12);
                lut[i] = ColorRampInterpolation(alpha);
            }

            return lut;
        }



        /// <summary>
        /// Initializes pseudo-color look up table for depth pixels
        /// </summary>
        private static uint[] InitializePseudoColorLut()
        {
            uint[] lut = new uint[TableSize];
            for (int i = 0; i < TableSize; i++)
            {
                lut[i] = ColorRampInterpolation((float)i / TableSize);
            }

            return lut;
        }



        /// <summary>
```

```
        /// Maps a float value to a pseudo-color pixel
        /// </summary>
        private static uint ColorRampInterpolation(float value)
        {
            // Map value to surrounding indexes on the color ramp
            int rampSteps = ColorRamp.Length - 1;
            float scaled = value * rampSteps;
            int integer = (int)scaled;
            int index =
                integer < 0 ? 0 :
                integer >= rampSteps - 1 ? rampSteps - 1 :
                integer;

            Color prev = ColorRamp[index];
            Color next = ColorRamp[index + 1];

            // Set color based on ratio of closeness between the surrounding
colors
            uint alpha = (uint)((scaled - integer) * 255);
            uint beta = 255 - alpha;
            return
                ((prev.A * beta + next.A * alpha) / 255) << 24 | // Alpha
                ((prev.R * beta + next.R * alpha) / 255) << 16 | // Red
                ((prev.G * beta + next.G * alpha) / 255) << 8 |  // Green
                ((prev.B * beta + next.B * alpha) / 255);        // Blue
        }


        /// <summary>
        /// Maps a value in [0, 1] to a pseudo RGBA color.
        /// </summary>
        /// <param name="value">Input value between [0, 1].</param>
        [MethodImpl(MethodImplOptions.AggressiveInlining)]

        private static uint PseudoColor(float value)
        {
            int index = (int)(value * TableSize);
            index = index < 0 ? 0 : index > TableSize - 1 ? TableSize - 1 :
index;
            return PseudoColorTable[index];
        }

        #endregion

        /// <summary>
        /// Maps each pixel in a scanline from a 16 bit depth value to a
pseudo-color pixel.
        /// </summary>
        /// <param name="pixelWidth">Width of the input scanline, in pixels.
</param>
        /// <param name="inputRowBytes">Pointer to the start of the input
scanline.</param>
        /// <param name="outputRowBytes">Pointer to the start of the output
scanline.</param>
        /// <param name="depthScale">Physical distance that corresponds to
```

```
one unit in the input scanline.</param>
        /// <param name="minReliableDepth">Shortest distance at which the
sensor can provide reliable measurements.</param>
        /// <param name="maxReliableDepth">Furthest distance at which the
sensor can provide reliable measurements.</param>

        public static unsafe void PseudoColorForDepth(int pixelWidth, byte*
inputRowBytes, byte* outputRowBytes, float depthScale, float
minReliableDepth, float maxReliableDepth)
        {
            // Visualize space in front of your desktop.
            float minInMeters = minReliableDepth * depthScale;
            float maxInMeters = maxReliableDepth * depthScale;
            float one_min = 1.0f / minInMeters;
            float range = 1.0f / maxInMeters - one_min;

            ushort* inputRow = (ushort*)inputRowBytes;
            uint* outputRow = (uint*)outputRowBytes;

            for (int x = 0; x < pixelWidth; x++)
            {
                var depth = inputRow[x] * depthScale;

                if (depth == 0)
                {
                    // Map invalid depth values to transparent pixels.
                    // This happens when depth information cannot be
calculated, e.g. when objects are too close.
                    outputRow[x] = 0;
                }
                else
                {
                    var alpha = (1.0f / depth - one_min) / range;
                    outputRow[x] = PseudoColor(alpha * alpha);
                }
            }
        }



        /// <summary>
        /// Maps each pixel in a scanline from a 8 bit infrared value to a
pseudo-color pixel.
        /// </summary>
        /// /// <param name="pixelWidth">Width of the input scanline, in
pixels.</param>
        /// <param name="inputRowBytes">Pointer to the start of the input
scanline.</param>
        /// <param name="outputRowBytes">Pointer to the start of the output
scanline.</param>

        public static unsafe void PseudoColorFor8BitInfrared(
            int pixelWidth, byte* inputRowBytes, byte* outputRowBytes)
        {
            byte* inputRow = inputRowBytes;
```

```
            uint* outputRow = (uint*)outputRowBytes;

            for (int x = 0; x < pixelWidth; x++)
            {
                outputRow[x] = InfraredColor(inputRow[x] /
(float)Byte.MaxValue);
            }
        }

        /// <summary>
        /// Maps each pixel in a scanline from a 16 bit infrared value to a
pseudo-color pixel.
        /// </summary>
        /// <param name="pixelWidth">Width of the input scanline.</param>
        /// <param name="inputRowBytes">Pointer to the start of the input
scanline.</param>
        /// <param name="outputRowBytes">Pointer to the start of the output
scanline.</param>

        public static unsafe void PseudoColorFor16BitInfrared(int
pixelWidth, byte* inputRowBytes, byte* outputRowBytes)
        {
            ushort* inputRow = (ushort*)inputRowBytes;
            uint* outputRow = (uint*)outputRowBytes;

            for (int x = 0; x < pixelWidth; x++)
            {
                outputRow[x] = InfraredColor(inputRow[x] /
(float)UInt16.MaxValue);
            }
        }
    }


    // Displays the provided softwareBitmap in a XAML image control.
    public void PresentSoftwareBitmap(SoftwareBitmap softwareBitmap)
    {
        if (softwareBitmap != null)
        {
            // Swap the processed frame to m_backBuffer and trigger UI
thread to render it
            softwareBitmap = Interlocked.Exchange(ref m_backBuffer,
softwareBitmap);

            // UI thread always reset m_backBuffer before using it.  Unused
bitmap should be disposed.
            softwareBitmap?.Dispose();

            // Changes to xaml ImageElement must happen in UI thread through
Dispatcher
            var task =
m_imageElement.Dispatcher.RunAsync(CoreDispatcherPriority.Normal,
                async () =>
                {
                    // Don't let two copies of this task run at the same
```

```
time.
                    if (_taskRunning)
                    {
                        return;
                    }
                    _taskRunning = true;

                    // Keep draining frames from the backbuffer until the
backbuffer is empty.
                    SoftwareBitmap latestBitmap;
                    while ((latestBitmap = Interlocked.Exchange(ref
m_backBuffer, null)) != null)
                    {
                        var imageSource =
(SoftwareBitmapSource)m_imageElement.Source;
                        await imageSource.SetBitmapAsync(latestBitmap);
                        latestBitmap.Dispose();
                    }

                    _taskRunning = false;
                });
        }
    }
}
```

Use MultiSourceMediaFrameReader to get time-correlated frames from multiple sources

Starting with Windows 10, version 1607, you can use **MultiSourceMediaFrameReader** to receive time-corellated frames from multiple sources. This API makes it easier to do processing that requires frames from multiple sources that were taken in close temporal proximity, such as using the **DepthCorrelatedCoordinateMapper** class. One limitation of using this new method is that frame-arrived events are only raised at the rate of the slowest capture source. Extra frames from faster sources will be dropped. Also, because the system expects frames to arrive from different sources at different rates, it does not automatically recognize if a source has stopped generating frames altogether. The example code in this section shows how to use an event to create your own timeout logic that gets invoked if correlated frames don't arrive within an app-defined time limit.

The steps for using **MultiSourceMediaFrameReader** are similar to the steps for using **MediaFrameReader** described previously in this article. This example will use a color source and a depth source. Declare some string variables to store the media frame source IDs that will be used to select frames from each source. Next, declare a **ManualResetEventSlim**, a **CancellationTokenSource**, and an **EventHandler** that will be used to implement timeout logic for the example.

C#

```
private MultiSourceMediaFrameReader m_multiFrameReader = null;
private string m_colorSourceId = null;
private string m_depthSourceId = null;


private readonly ManualResetEventSlim m_frameReceived = new
ManualResetEventSlim(false);
private readonly CancellationTokenSource m_tokenSource = new
CancellationTokenSource();
public event EventHandler CorrelationFailed;
```

Using the techniques described previously in this article, query for a MediaFrameSourceGroup that includes the color and depth sources required for this example scenario. After selecting the desired frame source group, get the MediaFrameSourceInfo for each frame source.

C#

```
var allGroups = await MediaFrameSourceGroup.FindAllAsync();
var eligibleGroups = allGroups.Select(g => new
{
    Group = g,

    // For each source kind, find the source which offers that kind of media
frame,
    // or null if there is no such source.
    SourceInfos = new MediaFrameSourceInfo[]
    {
        g.SourceInfos.FirstOrDefault(info => info.SourceKind ==
MediaFrameSourceKind.Color),
        g.SourceInfos.FirstOrDefault(info => info.SourceKind ==
MediaFrameSourceKind.Depth)
    }
}).Where(g => g.SourceInfos.Any(info => info != null)).ToList();

if (eligibleGroups.Count == 0)
{
    System.Diagnostics.Debug.WriteLine("No source group with color, depth or
infrared found.");
    return;
}

var selectedGroupIndex = 0; // Select the first eligible group
MediaFrameSourceGroup selectedGroup =
eligibleGroups[selectedGroupIndex].Group;
MediaFrameSourceInfo colorSourceInfo =
eligibleGroups[selectedGroupIndex].SourceInfos[0];
MediaFrameSourceInfo depthSourceInfo =
eligibleGroups[selectedGroupIndex].SourceInfos[1];
```

Create and initialize a **MediaCapture** object, passing the selected frame source group in the initialization settings.

```
C#

m_mediaCapture = new MediaCapture();

var settings = new MediaCaptureInitializationSettings()
{
    SourceGroup = selectedGroup,
    SharingMode = MediaCaptureSharingMode.ExclusiveControl,
    MemoryPreference = MediaCaptureMemoryPreference.Cpu,
    StreamingCaptureMode = StreamingCaptureMode.Video
};

await m_mediaCapture.InitializeAsync(settings);
```

After initializing the **MediaCapture** object, retrieve MediaFrameSource objects for the color and depth cameras. Store the ID for each source so that you can select the arriving frame for the corresponding source.

```
C#

MediaFrameSource colorSource =
    m_mediaCapture.FrameSources.Values.FirstOrDefault(
        s => s.Info.SourceKind == MediaFrameSourceKind.Color);

MediaFrameSource depthSource =
    m_mediaCapture.FrameSources.Values.FirstOrDefault(
        s => s.Info.SourceKind == MediaFrameSourceKind.Depth);

if (colorSource == null || depthSource == null)
{
    System.Diagnostics.Debug.WriteLine("MediaCapture doesn't have the Color
and Depth streams");
    return;
}

m_colorSourceId = colorSource.Info.Id;
m_depthSourceId = depthSource.Info.Id;
```

Create and initialize the **MultiSourceMediaFrameReader** by calling CreateMultiSourceFrameReaderAsync and passing an array of frame sources that the reader will use. Register an event handler for the FrameArrived event. This example creates an instance the **FrameRenderer** helper class, described previously in this article, to render frames to an **Image** control. Start the frame reader by calling StartAsync.

Register an event handler for the **CorellationFailed** event declared earlier in the example. We will signal this event if one of the media frame sources being used stops producing frames. Finally, call Task.Run to call the timeout helper method, **NotifyAboutCorrelationFailure**, on a separate thread. The implementation of this method is shown later in this article.

```
C#
```

```
m_multiFrameReader = await m_mediaCapture.CreateMultiSourceFrameReaderAsync(
    new[] { colorSource, depthSource });

m_multiFrameReader.FrameArrived += MultiFrameReader_FrameArrived;

m_frameRenderer = new FrameRenderer(iFrameReaderImageControl);

MultiSourceMediaFrameReaderStartStatus startStatus =
    await m_multiFrameReader.StartAsync();

if (startStatus != MultiSourceMediaFrameReaderStartStatus.Success)
{
    throw new InvalidOperationException(
        "Unable to start reader: " + startStatus);
}

this.CorrelationFailed += MainWindow_CorrelationFailed;
Task.Run(() => NotifyAboutCorrelationFailure(m_tokenSource.Token));
```

The **FrameArrived** event is raised whenever a new frame is available from all of the media frame sources that are managed by the **MultiSourceMediaFrameReader**. This means that the event will be raised on the cadence of the slowest media source. If one source produces multiple frames in the time that a slower source produces one frame, the extra frames from the fast source will be dropped.

Get the MultiSourceMediaFrameReference associated with the event by calling TryAcquireLatestFrame. Get the **MediaFrameReference** associated with each media frame source by calling TryGetFrameReferenceBySourceId, passing in the ID strings stored when the frame reader was initialized.

Call the Set method of the **ManualResetEventSlim** object to signal that frames have arrived. We will check this event in the **NotifyCorrelationFailure** method that is running in a separate thread.

Finally, perform any processing on the time-correlated media frames. This example simply displays the frame from the depth source.

```
C#
```

```
private void MultiFrameReader_FrameArrived(MultiSourceMediaFrameReader
sender, MultiSourceMediaFrameArrivedEventArgs args)
{
    using (MultiSourceMediaFrameReference muxedFrame =
        sender.TryAcquireLatestFrame())
    using (MediaFrameReference colorFrame =
        muxedFrame.TryGetFrameReferenceBySourceId(m_colorSourceId))
    using (MediaFrameReference depthFrame =
        muxedFrame.TryGetFrameReferenceBySourceId(m_depthSourceId))
    {
        // Notify the listener thread that the frame has been received.
        m_frameReceived.Set();
        m_frameRenderer.ProcessFrame(depthFrame);
    }
}
```

The **NotifyCorrelationFailure** helper method was run on a separate thread after the frame reader was started. In this method, check to see if the frame received event has been signaled. Remember, in the **FrameArrived** handler, we set this event whenever a set of correlated frames arrive. If the event hasn't been signaled for some app-defined period of time - 5 seconds is a reasonable value - and the task wasn't cancelled using the **CancellationToken**, then it's likely that one of the media frame sources has stopped reading frames. In this case you typically want to shut down the frame reader, so raise the app-defined **CorrelationFailed** event. In the handler for this event you can stop the frame reader and clean up it's associated resources as shown previously in this article.

C#

```
private void NotifyAboutCorrelationFailure(CancellationToken token)
{
    // If in 5 seconds the token is not cancelled and frame event is not
signaled,
    // correlation is most likely failed.
    if (WaitHandle.WaitAny(new[] { token.WaitHandle,
m_frameReceived.WaitHandle }, 5000)
            == WaitHandle.WaitTimeout)
    {
        CorrelationFailed?.Invoke(this, EventArgs.Empty);
    }
}
```

C#

```
private async void MainWindow_CorrelationFailed(object sender, EventArgs e)
{
    await m_multiFrameReader.StopAsync();
    m_multiFrameReader.FrameArrived -= MultiFrameReader_FrameArrived;
    m_mediaCapture.Dispose();
```

```
    m_mediaCapture = null;
}
```

Use buffered frame acquisition mode to preserve the sequence of acquired frames

Starting with Windows 10, version 1709, you can set the **AcquisitionMode** property of a **MediaFrameReader** or **MultiSourceMediaFrameReader** to **Buffered** to preserve the sequence of frames passed into your app from the frame source.

```
C#

m_mediaFrameReader.AcquisitionMode =
MediaFrameReaderAcquisitionMode.Buffered;
```

In the default acquisition mode, **Realtime**, if multiple frames are acquired from the source while your app is still handling the **FrameArrived** event for a previous frame, the system will send your app the most recently acquired frame and drop additional frames waiting in the buffer. This provides your app with the most recent available frame at all times. This is typically the most useful mode for realtime computer vision applications.

In **Buffered** acquisition mode, the system will keep all frames in the buffer and provide them to your app through the **FrameArrived** event in the order received. Note that in this mode, when system's buffer for frames is filled, the system will stop acquiring new frames until your app completes the **FrameArrived** event for previous frames, freeing up more space in the buffer.

Use MediaSource to display frames in a MediaPlayerElement

Starting with Windows, version 1709, you can display frames acquired from a **MediaFrameReader** directly in a **MediaPlayerElement** control in your XAML page. This is achieved by using the **MediaSource.CreateFromMediaFrameSource** to create **MediaSource** object that can be used directly by a **MediaPlayer** associated with a **MediaPlayerElement**. For detailed information on working with **MediaPlayer** and **MediaPlayerElement**, see Play audio and video with MediaPlayer.

The following code examples show you a simple implementation that displays the frames from a front-facing and back-facing camera simultaneously in a XAML page.

First, add two **MediaPlayerElement** controls to your XAML page.

XML

```
<MediaPlayerElement x:Name="mediaPlayerElement1" Width="320" Height="240"/>
```

XML

```
<MediaPlayerElement x:Name="mediaPlayerElement2" Width="320" Height="240"/>
```

Next, using the techniques shown in previous sections in this article, select a **MediaFrameSourceGroup** that contains **MediaFrameSourceInfo** objects for color cameras on the front panel and back panel. Note that the **MediaPlayer** does not automatically convert frames from non-color formats, such as a depth or infrared data, into color data. Using other sensor types may produce unexpected results.

C#

```
var allGroups = await MediaFrameSourceGroup.FindAllAsync();
var eligibleGroups = allGroups.Select(g => new
{
    Group = g,

    // For each source kind, find the source which offers that kind of media
frame,
    // or null if there is no such source.
    SourceInfos = new MediaFrameSourceInfo[]
    {
        g.SourceInfos.FirstOrDefault(info =>
info.DeviceInformation?.EnclosureLocation.Panel ==
Windows.Devices.Enumeration.Panel.Front
            && info.SourceKind == MediaFrameSourceKind.Color),
        g.SourceInfos.FirstOrDefault(info =>
info.DeviceInformation?.EnclosureLocation.Panel ==
Windows.Devices.Enumeration.Panel.Back
            && info.SourceKind == MediaFrameSourceKind.Color)
    }
}).Where(g => g.SourceInfos.Any(info => info != null)).ToList();

if (eligibleGroups.Count == 0)
{
    System.Diagnostics.Debug.WriteLine("No source group with front and back-
facing camera found.");
    return;
}

var selectedGroupIndex = 0; // Select the first eligible group
MediaFrameSourceGroup selectedGroup =
eligibleGroups[selectedGroupIndex].Group;
```

```
MediaFrameSourceInfo frontSourceInfo = selectedGroup.SourceInfos[0];
MediaFrameSourceInfo backSourceInfo = selectedGroup.SourceInfos[1];
```

Initialize the **MediaCapture** object to use the selected **MediaFrameSourceGroup**.

C#

```
m_mediaCapture = new MediaCapture();

var settings = new MediaCaptureInitializationSettings()
{
    SourceGroup = selectedGroup,
    SharingMode = MediaCaptureSharingMode.ExclusiveControl,
    MemoryPreference = MediaCaptureMemoryPreference.Cpu,
    StreamingCaptureMode = StreamingCaptureMode.Video
};
try
{
    await m_mediaCapture.InitializeAsync(settings);
}
catch (Exception ex)
{
    System.Diagnostics.Debug.WriteLine("MediaCapture initialization failed:
" + ex.Message);
    return;
}
```

Finally, call MediaSource.CreateFromMediaFrameSource to create a **MediaSource** for each frame source by using the Id property of the associated **MediaFrameSourceInfo** object to select one of the frame sources in the **MediaCapture** object's FrameSources collection. Initialize a new **MediaPlayer** object and assign it to a **MediaPlayerElement** by calling SetMediaPlayer. Then set the Source property to the newly created **MediaSource** object.

C#

```
var frameMediaSource1 =
MediaSource.CreateFromMediaFrameSource(m_mediaCapture.FrameSources[frontSour
ceInfo.Id]);
mediaPlayerElement1.SetMediaPlayer(new
Windows.Media.Playback.MediaPlayer());
mediaPlayerElement1.MediaPlayer.Source = frameMediaSource1;
mediaPlayerElement1.AutoPlay = true;

var frameMediaSource2 =
MediaSource.CreateFromMediaFrameSource(m_mediaCapture.FrameSources[backSourc
eInfo.Id]);
mediaPlayerElement2.SetMediaPlayer(new
Windows.Media.Playback.MediaPlayer());
```

```
mediaPlayerElement2.MediaPlayer.Source = frameMediaSource2;
mediaPlayerElement2.AutoPlay = true;
```

Use video profiles to select a frame source

A camera profile, represented by a MediaCaptureVideoProfile object, represents a set of capabilities that a particular capture device provides, such as frame rates, resolutions, or advanced features like HDR capture. A capture device may support multiple profiles, allowing you to select the one that is optimized for your capture scenario. Starting with Windows 10, version 1803, you can use **MediaCaptureVideoProfile** to select a media frame source with particular capabilities before initializing the **MediaCapture** object. The following example code looks for a video profile that supports HDR with Wide Color Gamut (WCG) and returns a **MediaCaptureInitializationSettings** object that can be used to initialize the **MediaCapture** to use the selected device and profile.

First, call MediaFrameSourceGroup.FindAllAsync to get a list of all media frame source groups available on the current device. Loop through each source group and call MediaCapture.FindKnownVideoProfiles to get a list of all of the video profiles for the current source group that support the specified profile, in this case HDR with WCG photo. If a profile that meets the criteria is found, create a new **MediaCaptureInitializationSettings** object and set the **VideoProfile** to the select profile and the **VideoDeviceId** to the **Id** property of the current media frame source group.

C#

```
IReadOnlyList<MediaFrameSourceGroup> sourceGroups = await
MediaFrameSourceGroup.FindAllAsync();
MediaCaptureInitializationSettings settings = null;

foreach (MediaFrameSourceGroup sourceGroup in sourceGroups)
{
    // Find a device that support AdvancedColorPhoto
    IReadOnlyList<MediaCaptureVideoProfile> profileList =
MediaCapture.FindKnownVideoProfiles(
                                    sourceGroup.Id,
                                    KnownVideoProfile.HdrWithWcgPhoto);

    if (profileList.Count > 0)
    {
        settings = new MediaCaptureInitializationSettings();
        settings.VideoProfile = profileList[0];
        settings.VideoDeviceId = sourceGroup.Id;
        break;
    }
}
```

For more information on using camera profiles, see Camera profiles.

Related topics

- Camera
- Basic photo, video, and audio capture with MediaCapture
- Camera frames sample

Process audio frames with MediaFrameReader

Article • 01/16/2025

This article shows you how to use a MediaFrameReader with MediaCapture to get audio data from a media frame source. To learn about using a **MediaFrameReader** to get image data, such as from a color, infrared, or depth camera, see Process media frames with MediaFrameReader. That article provides a general overview of the frame reader usage pattern and discusses some additional features of the **MediaFrameReader** class, such as using **MediaFrameSourceGroup** to retrieve frames from multiple sources at the same time.

> **Note**
>
> The features discussed in this article are only available starting with Windows 10, version 1803.

Setting up your project

The process for acquiring audio frames is largely the same as acquiring other types of media frames. As with any app that uses **MediaCapture**, you must declare that your app uses the *webcam* capability before attempting to access any camera device. If your app will capture from an audio device, you should also declare the *microphone* device capability.

Select frame sources and frame source groups

The first step in capturing audio frames is to initialize a MediaFrameSource representing the source of the audio data, such as a microphone or other audio capture device. To do this, you must create a new instance of the MediaCapture object. For this example, the only initialization setting for the **MediaCapture** is setting the StreamingCaptureMode to indicate that we want to stream audio from the capture device.

After calling MediaCapture.InitializeAsync, you can get the list of accessible media frame sources with the FrameSources property. This example selects all frame sources where the MediaFrameSourceInfo describing the frame source has a MediaStreamType of **Audio**, indicating that the media source produces audio data.

If the query returns one or more frame sources, you can check the CurrentFormat property to see if the source supports the audio format you desire - in this example, float audio data. Check the AudioEncodingProperties to make sure the audio encoding you desire is supported by the source.

C#

```
m_mediaCapture = new MediaCapture();
MediaCaptureInitializationSettings settings = new
MediaCaptureInitializationSettings()
{
    StreamingCaptureMode = StreamingCaptureMode.Audio,
};
await m_mediaCapture.InitializeAsync(settings);

var audioFrameSources = m_mediaCapture.FrameSources.Where(x =>
x.Value.Info.MediaStreamType == MediaStreamType.Audio);

if (audioFrameSources.Count() == 0)
{
    Debug.WriteLine("No audio frame source was found.");
    return;
}

MediaFrameSource frameSource = audioFrameSources.FirstOrDefault().Value;

MediaFrameFormat format = frameSource.CurrentFormat;
if (format.Subtype != MediaEncodingSubtypes.Float)
{
    return;
}

if (format.AudioEncodingProperties.ChannelCount != 2
    || format.AudioEncodingProperties.SampleRate != 48000)
{
    return;
}
```

Create and Start the MediaFrameReader

Get a new instance of **MediaFrameReader** by calling MediaCapture.CreateFrameReaderAsync, passing the **MediaFrameSource** object you selected in the previous step. By default, audio frames are obtained in buffered mode, making it less likely that frames will be dropped, although this can still occur if you are not processing audio frames fast enough and fill up the system's alloted memory buffer.

Register a handler for the MediaFrameReader.FrameArrived event, which is raised by the system when a new frame of audio data is available. Call StartAsync to begin the

acquisition of audio frames. If the frame reader fails to start, the status value returned from the call will have a value other than Success.

C#

```
m_mediaFrameReader = await
m_mediaCapture.CreateFrameReaderAsync(frameSource);

// Optionally set acquisition mode. Buffered is the default mode for audio.
m_mediaFrameReader.AcquisitionMode =
MediaFrameReaderAcquisitionMode.Buffered;

m_mediaFrameReader.FrameArrived += MediaFrameReader_AudioFrameArrived;

var status = await m_mediaFrameReader.StartAsync();

if (status != MediaFrameReaderStartStatus.Success)
{
    Debug.WriteLine("The MediaFrameReader couldn't start.");
}
```

In the **FrameArrived** event handler, call TryAcquireLatestFrame on the **MediaFrameReader** object passed as the sender to the handler to attempt to retrieve a reference to the latest media frame. Note that this object can be null, so you should always check before using the object. The typs of media frame wrapped in the **MediaFrameReference** returned from **TryAcquireLatestFrame** depends on what type of frame source or sources you configured the frame reader to acquire. Since the frame reader in this example was set up to acquire audio frames, it gets the underlying frame using the AudioMediaFrame property.

This **ProcessAudioFrame** helper method in the example below shows how to get an AudioFrame which provides information such as the timestamp of the frame and whether it is discontinuous from the **AudioMediaFrame** object. To read or process the audio sample data, you will need to get the AudioBuffer object from the **AudioMediaFrame** object, create an IMemoryBufferReference, and then call the COM method **IMemoryBufferByteAccess::GetBuffer** to retrieve the data. See the note below the code listing for more information on accessing native buffers.

The format of the data depends on the frame source. In this example, when selecting a media frame source, we explicitly made certain that the selected frame source used two channels of float data. The rest of the example code shows how to determine the duration and sample count for the audio data in the frame.

C#

```
private void MediaFrameReader_AudioFrameArrived(MediaFrameReader sender,
MediaFrameArrivedEventArgs args)
{
    using (MediaFrameReference reference = sender.TryAcquireLatestFrame())
    {
        if (reference != null)
        {
            ProcessAudioFrame(reference.AudioMediaFrame);
        }
    }
}
unsafe private void ProcessAudioFrame(AudioMediaFrame audioMediaFrame)
{

    using (AudioFrame audioFrame = audioMediaFrame.GetAudioFrame())
    using (AudioBuffer buffer =
audioFrame.LockBuffer(AudioBufferAccessMode.Read))
    using (IMemoryBufferReference reference = buffer.CreateReference())
    {
        byte* dataInBytes;
        uint capacityInBytes;
        float* dataInFloat;

        var memoryBuffer = reference.As<IMemoryBufferByteAccess>();
        memoryBuffer.GetBuffer(out dataInBytes, out capacityInBytes);


        // The requested format was float
        dataInFloat = (float*)dataInBytes;

        // Get the number of samples by multiplying the duration by sampling
rate:
        // duration [s] x sampling rate [samples/s] = # samples

        // Duration can be gotten off the frame reference OR the audioFrame
        TimeSpan duration = audioMediaFrame.FrameReference.Duration;

        // frameDurMs is in milliseconds, while SampleRate is given per
second.
        uint frameDurMs = (uint)duration.TotalMilliseconds;
        uint sampleRate =
audioMediaFrame.AudioEncodingProperties.SampleRate;
        uint sampleCount = (frameDurMs * sampleRate) / 1000;

    }
}
```

> **Note**
>
> In order to do operate on the audio data, you must access a native memory buffer. To do this, you must use the **IMemoryBufferByteAccess** COM interface by

including the code listing below. Operations on the native buffer must be performed in a method that uses the **unsafe** keyword. You also need to check the box to allow unsafe code in the **Build** tab of the **Project -> Properties** dialog.

C#

```
[ComImport]
[Guid("5B0D3235-4DBA-4D44-865E-8F1D0E4FD04D")]
[InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
unsafe interface IMemoryBufferByteAccess
{
    void GetBuffer(out byte* buffer, out uint capacity);
}
```

Additional information on using MediaFrameReader with audio data

You can retrieve the **AudioDeviceController** associated with the audio frame source by accessing the **MediaFrameSource.Controller** property. This object can be used to get or set the stream properties of the capture device or to control the capture level. The following example mutes the audio device so that frames continue to be acquired by the frame reader, but all samples have value of 0.

C#

```
m_audioDeviceController.Muted = true;
```

You can use an **AudioFrame** object to pass audio data captured by a media frame source into an **AudioGraph**. Pass the frame into the **AddFrame** method of an **AudioFrameInputNode**. For more information on using audio graphs to capture, process, and mix audio signals, see Audio graphs.

Related topics

- Process media frames with MediaFrameReader
- Camera
- Basic photo, video, and audio capture with MediaCapture
- Camera frames sample
- Audio graphs

Detect and respond to audio level changes by the system

Article • 01/16/2025

Learn how to detect and respond to audio level changes by the system. Starting with Windows 10, version 1803, your app can detect when the system lowers or mutes the audio level of your app's audio capture and audio render streams. For example, the system may mute your app's streams when it goes into the background.

To learn about capturing audio using the **MediaCapture** class, see Basic photo, video, and audio capture with MediaCapture. For information about audio playback using MediaPlayer, see TBD.

The **AudioStateMonitor** class allows you to register to receive an event when the system modifies the volume of an audio capture or render stream. Get an instance of **AudioStateMonitor** for monitoring audio capture streams by calling **CreateForCaptureMonitoring**. Get an instance for monitoring audio render streams by calling **CreateForRenderMonitoring**. Register a handler for the **SoundLevelChanged** event of each monitor to be notified when the audio for the corresponding stream category is changed by the system.

C#

```
AudioStateMonitor captureAudioStateMonitor;
AudioStateMonitor renderAudioStateMonitor;
```

C#

```
captureAudioStateMonitor = AudioStateMonitor.CreateForCaptureMonitoring();
captureAudioStateMonitor.SoundLevelChanged +=
CaptureAudioStateMonitor_SoundLevelChanged;

renderAudioStateMonitor = AudioStateMonitor.CreateForRenderMonitoring();
renderAudioStateMonitor.SoundLevelChanged +=
RenderAudioStateMonitor_SoundLevelChanged;
```

In the **SoundLevelChanged** handler for the capture stream, you can check the **SoundLevel** property of the **AudioStateMonitor** sender to determine the new sound level. Note that a capture stream should never be lowered, or "ducked", by the system. It should only ever be muted or switched back to full volume. If the audio stream is muted, you can stop a capture in progress. If the audio stream is restored to full volume, you can start capturing again.

C#

```
bool isCapturingAudio = false;
bool capturingStoppedForAudioState = false;

private void CaptureAudioStateMonitor_SoundLevelChanged(AudioStateMonitor
sender, object args)
{
    switch (sender.SoundLevel)
    {
        case SoundLevel.Full:
            if (capturingStoppedForAudioState)
            {
                MyStartAudioCapture();
                capturingStoppedForAudioState = false;
            }
            break;
        case SoundLevel.Muted:
            if (isCapturingAudio)
            {
                MyStopAudioCapture();
                capturingStoppedForAudioState = true;
            }
            break;
        case SoundLevel.Low:
            // This should never happen for capture
            Debug.WriteLine("Unexpected audio state.");
            break;
    }
}
```

The following code example illustrates an implementation of the **SoundLevelChanged** handler for audio rendering. Depending on your app scenario, and the type of content you are playing, you may want to pause audio playback when the sound level is ducked.

C#

```
bool contentIsPodcast;
private void RenderAudioStateMonitor_SoundLevelChanged(AudioStateMonitor
sender, object args)
{
    if ((sender.SoundLevel == SoundLevel.Full) ||
        (sender.SoundLevel == SoundLevel.Low && !contentIsPodcast))
    {
        m_mediaPlayer.Play();
    }
    else if ((sender.SoundLevel == SoundLevel.Muted) ||
         (sender.SoundLevel == SoundLevel.Low && contentIsPodcast))
    {
        // Pause playback if we re muted or if we re playing a podcast and
are ducked
        m_mediaPlayer.Pause();
```

```
    }
}
```

Related topics

Basic photo, video, and audio capture with MediaCapture

Launch the camera settings page

Article • 01/16/2025

Windows defines a set of URIs that allow apps to launch the Windows Settings app and display a particular settings page. This article explains how to launch the Windows Settings app directly to the camera settings page and, optionally, navigate directly to the settings for a particular camera on the device. For more information, see Launch the Windows Settings app.

The camera settings URL

Starting with Windows 11, Build 22000, the URI `ms-settings:camera` launches the Windows Settings app and navigates to the camera settings page. Note that in previous versions of Windows, this same URI would launch the default camera application. In addition to the general camera settings page, you can append the query string parameter `cameraId` set to the symbolic link name, in escaped URI format, to launch directly to the settings page for the associated camera.

In the following example, the DeviceInformation class is used to retrieve the symbolic link name for the first video capture device on the current machine, if one exists. Next, LaunchUriAsync is called to launch the Windows Settings app. The `ms-settings:camera` Uri specifies that the camera settings page should be shown. The optional query string parameter `cameraId` is set to the symbolic link name for the camera, escaped with a call to Url.EscapeDataString, to specify that the settings for the associated camera should be shown.

C#

```
private async void bLaunchCameraSettings_Click(object sender,
RoutedEventArgs e)
{
    var captureDevices = await
DeviceInformation.FindAllAsync(DeviceClass.VideoCapture);
    if (captureDevices.Count() > 0)
    {
        var cameraSymbolicLink = captureDevices.First().Id;
        bool result = await Windows.System.Launcher.LaunchUriAsync(new
Uri("ms-settings:camera?cameraId=" +
Uri.EscapeDataString(cameraSymbolicLink)));
    }
}
```

Related topics

- Launch the Windows Settings app

Handle the Windows camera privacy setting

Article • 01/16/2025

Windows allows users to grant or deny access to the device's camera in the Windows Settings app, under **Privacy & Security -> Camera**. Camera access can be disabled for the entire device, for all unpackaged apps, or for individual packaged apps. This article describes the best practices for checking whether your app has access to the camera and handling the case where access is denied by the user.

Check for access before initializing the camera

For packaged apps, you should check to see if your app has camera access before initializing the camera. Use the AppCapability class to determine if your app has access.

C#

C#

```
bool cameraCapabilityAccess = false;
private void CheckCameraAccessStatus()
{
    var status = AppCapability.Create("Webcam").CheckAccess();

    if (status == AppCapabilityAccessStatus.Allowed)
    {
        cameraCapabilityAccess = true;
        cameraButton.IsEnabled = true;
    }
    else
    {
        cameraCapabilityAccess = false;
        cameraButton.IsEnabled = false;
    }
}
```

Handle the access denied error

The Windows camera capture APIs will return the error **E_ACCESSDENIED** when apps attempt to access the camera capture device if the user has disabled camera in the camera privacy Settings page. Apps should check for this error when initializing the

capture device. If the initialization fails with this error, it is recommended that you direct the user to the camera privacy Settings page and potentially enable access for your app. The camera privacy Settings page can be launched using the URI `ms-settings:privacy-webcam`.

The following example illustrates how to check for E_ACCESSDENIED when calling MediaCapture.InitializeAsync.

C#

```
C#

try
{
    await
mediaCapture.InitializeAsync(mediaCaptureInitializationSettings);
}
catch (System.UnauthorizedAccessException ex)
{
    // E_ACCESSDENIED, 0x80070005 in hexadecimal, -2147024891 in decimal
    if (ex.HResult == -2147024891)
    {
        StatusTextBlock.Text = "Access to the camera has been denied." +
            "Click the Settings button to check the camera privacy
settings";
    }

    return;
}
...
// Launch the camera privacy Settings page
private async void LaunchSettingsButton_Click(object sender,
RoutedEventArgs e)
{
    bool result = await Windows.System.Launcher.LaunchUriAsync(new
Uri("ms-settings:privacy-webcam"));
}
```

The following example illustrates handling the E_ACCESSDENIED error returned from IMFActivate::ActivateObject when initializing an IMFMediaSource for a capture device.

```
C++

IMFMediaSource* pSource = NULL;
IMFAttributes* pAttributes = NULL;
IMFActivate** ppDevices = NULL;

// Create an attribute store to specify the enumeration parameters.
```

```
HRESULT hr = MFCreateAttributes(&pAttributes, 1);
if (FAILED(hr))
{
    goto done;
}

// Source type: video capture devices
hr = pAttributes->SetGUID(
    MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE,
    MF_DEVSOURCE_ATTRIBUTE_SOURCE_TYPE_VIDCAP_GUID
);
if (FAILED(hr))
{
    goto done;
}

// Enumerate devices.
UINT32 count;
hr = MFEnumDeviceSources(pAttributes, &ppDevices, &count);
if (FAILED(hr))
{
    goto done;
}

if (count == 0)
{
    hr = E_FAIL;
    goto done;
}

// Create the media source object.
hr = ppDevices[0]->ActivateObject(IID_PPV_ARGS(&pSource));
if (FAILED(hr))
{
    if (hr == E_ACCESSDENIED)
    {
        int response = MessageBox(hWnd, L"Access to the camera was denied.
Open the camera privacy settings?", L"Error", MB_YESNO);
        if (response == IDYES)
        {
            ShellExecute(NULL, L"open", L"ms-settings:privacy-webcam", L"",
L".", SW_SHOWDEFAULT);
        }
    }
    goto done;
}
```

Implement fallback behavior

Apps should implement the previous steps to alert the user detect and alert the user that camera access is restricted due to privacy settings and to direct the user to the

camera privacy Settings page to allow them to update their settings. After these steps, the app should retry camera initialization to see if access has been granted. If the user declines to update their settings to allow your app to access to the camera, consider providing alternative functionality. For example, you could disable camera features, switch to a different mode, or display a placeholder image in place of the camera preview.

Connect to remote cameras

Article • 01/16/2025

This article shows you how to connect to one or more remote cameras and get a **MediaFrameSourceGroup** object that allows you to read frames from each camera. For more information on reading frames from a media source, see Process media frames with MediaFrameReader. For more information on pairing with devices, see Pair devices.

> **Note**
>
> The features discussed in this article are available starting with Windows 10, version 1903.

Create a DeviceWatcher class to watch for available remote cameras

The **DeviceWatcher** class monitors the devices available to your app and notifies your app when devices are added or removed. Get an instance of **DeviceWatcher** by calling **DeviceInformation.CreateWatcher**, passing in an Advanced Query Syntax (AQS) string that identifies the type of devices you want to monitor. The AQS string specifying network camera devices is the following:

syntax

```
@"System.Devices.InterfaceClassGuid:=""{B8238652-B500-41EB-B4F3-
4234F7F5AE99}"" AND
System.Devices.InterfaceEnabled:=System.StructuredQueryType.Boolean#True"
```

> **Note**
>
> The helper method MediaFrameSourceGroup.GetDeviceSelector returns an AQS string that will monitor locally-connected and remote network cameras. To monitor only network cameras, you should use the AQS string shown above.

When you start the returned **DeviceWatcher** by calling the **Start** method, it will raise the **Added** event for every network camera that is currently available. Until you stop the watcher by calling **Stop**, the **Added** event will be raised when new network camera

devices become available and the Removed event will be raised when a camera device becomes unavailable.

The event args passed into the **Added** and **Removed** event handlers are a DeviceInformation or a DeviceInformationUpdate object, respectively. Each of these objects has an **Id** property that is the identifier for the network camera for which the event was fired. Pass this ID into the MediaFrameSourceGroup.FromIdAsync method to get a MediaFrameSourceGroup object that you can use to retrieve frames from the camera.

Remote camera pairing helper class

The following example shows a helper class that uses a **DeviceWatcher** to create and update an **ObservableCollection** of **MediaFrameSourceGroup** objects to support data binding to the list of cameras. Typical apps would wrap the **MediaFrameSourceGroup** in a custom model class. Note that the helper class maintains a reference to the app's DispatcherQueue and updates the collection of cameras on the UI thread.

Also, this example handles the DeviceWatcher.Updated event in addition to the **Added** and **Removed** events. In the **Updated** handler, the associated remote camera device is removed from and then added back to the collection.

C#

```
class RemoteCameraPairingHelper : IDisposable
{
    private DispatcherQueue _dispatcherQueue;
    private DeviceWatcher _watcher;
    private ObservableCollection<MediaFrameSourceGroup>
_remoteCameraCollection;

    public RemoteCameraPairingHelper(DispatcherQueue uiDispatcherQueue)
    {
        _dispatcherQueue = uiDispatcherQueue;
        _remoteCameraCollection = new
ObservableCollection<MediaFrameSourceGroup>();
        var remoteCameraAqs = @"System.Devices.InterfaceClassGuid:=""
{B8238652-B500-41EB-B4F3-4234F7F5AE99}"" AND
System.Devices.InterfaceEnabled:=System.StructuredQueryType.Boolean#True";
        _watcher = DeviceInformation.CreateWatcher(remoteCameraAqs);
        _watcher.Added += Watcher_Added;
        _watcher.Removed += Watcher_Removed;
        _watcher.Updated += Watcher_Updated;
        _watcher.Start();
    }
    public void Dispose()
    {
        _watcher.Stop();
```

```
        _watcher.Updated -= Watcher_Updated;
        _watcher.Removed -= Watcher_Removed;
        _watcher.Added -= Watcher_Added;
    }
    public IReadOnlyList<MediaFrameSourceGroup> FrameSourceGroups
    {

        get { return _remoteCameraCollection; }
    }
    private async void Watcher_Updated(DeviceWatcher sender,
DeviceInformationUpdate args)
    {
        await RemoveDevice(args.Id);
        await AddDeviceAsync(args.Id);
    }
    private async void Watcher_Removed(DeviceWatcher sender,
DeviceInformationUpdate args)
    {
        await RemoveDevice(args.Id);
    }
    private async void Watcher_Added(DeviceWatcher sender, DeviceInformation
args)
    {
        await AddDeviceAsync(args.Id);
    }
    private async Task AddDeviceAsync(string id)
    {
        var group = await MediaFrameSourceGroup.FromIdAsync(id);
        if (group != null)
        {
            _dispatcherQueue.TryEnqueue(DispatcherQueuePriority.Normal, ()
=> {
                _remoteCameraCollection.Add(group);
            });
        }
    }
    private async Task RemoveDevice(string id)
    {
        _dispatcherQueue.TryEnqueue(DispatcherQueuePriority.Normal, () =>
        {
            var existing = _remoteCameraCollection.FirstOrDefault(item =>
item.Id == id);
            if (existing != null)
            {
                _remoteCameraCollection.Remove(existing);
            }
        });
    }
```

C++/WinRT

```
struct RemoteCameraPairingHelper
{
```

```
    RemoteCameraPairingHelper(DispatcherQueue uiDispatcher) :
        m_dispatcherQueue(uiDispatcher)
    {
        m_remoteCameraCollection =
winrt::single_threaded_observable_vector<MediaFrameSourceGroup>();
        auto remoteCameraAqs =
            LR"(System.Devices.InterfaceClassGuid:="{B8238652-B500-41EB-
B4F3-4234F7F5AE99}")"
            LR"(AND
System.Devices.InterfaceEnabled:=System.StructuredQueryType.Boolean#True)";

        m_watcher = DeviceInformation::CreateWatcher(remoteCameraAqs,
nullptr);

        m_watcherAddedAutoRevoker = m_watcher.Added(winrt::auto_revoke, {
this, &RemoteCameraPairingHelper::Watcher_Added });
        m_watcherRemovedAutoRevoker = m_watcher.Removed(winrt::auto_revoke,
{ this, &RemoteCameraPairingHelper::Watcher_Removed });
        m_watcherUpdatedAutoRevoker = m_watcher.Updated(winrt::auto_revoke,
{ this, &RemoteCameraPairingHelper::Watcher_Updated });
        m_watcher.Start();
    }
    ~RemoteCameraPairingHelper()
    {
        m_watcher.Stop();
    }
    IObservableVector<MediaFrameSourceGroup> FrameSourceGroups()
    {
        return m_remoteCameraCollection;
    }
    winrt::fire_and_forget Watcher_Added(DeviceWatcher /* sender */,
DeviceInformation args)
    {
        co_await AddDeviceAsync(args.Id());
    }
    winrt::fire_and_forget Watcher_Removed(DeviceWatcher /* sender */,
DeviceInformationUpdate args)
    {
        co_await RemoveDevice(args.Id());
    }
    winrt::fire_and_forget Watcher_Updated(DeviceWatcher /* sender */,
DeviceInformationUpdate args)
    {
        co_await RemoveDevice(args.Id());
        co_await AddDeviceAsync(args.Id());
    }
    Windows::Foundation::IAsyncAction AddDeviceAsync(winrt::hstring id)
    {
        auto group = co_await MediaFrameSourceGroup::FromIdAsync(id);
        if (group)
        {
            //co_await m_dispatcherQueue;
            co_await wil::resume_foreground(m_dispatcherQueue);
            m_remoteCameraCollection.Append(group);
        }
```

```
    }
    Windows::Foundation::IAsyncAction RemoveDevice(winrt::hstring id)
    {
        //co_await m_dispatcherQueue;
        co_await wil::resume_foreground(m_dispatcherQueue);

        uint32_t ix{ 0 };
        for (auto const&& item : m_remoteCameraCollection)
        {
            if (item.Id() == id)
            {
                m_remoteCameraCollection.RemoveAt(ix);
                break;
            }
            ++ix;
        }
    }

private:
    DispatcherQueue m_dispatcherQueue{ nullptr };
    DeviceWatcher m_watcher{ nullptr };
    IObservableVector<MediaFrameSourceGroup> m_remoteCameraCollection;
    DeviceWatcher::Added_revoker m_watcherAddedAutoRevoker;
    DeviceWatcher::Removed_revoker m_watcherRemovedAutoRevoker;
    DeviceWatcher::Updated_revoker m_watcherUpdatedAutoRevoker;
};
```

Related topics

- Camera
- Basic photo, video, and audio capture with MediaCapture
- Camera frames sample
- Process media frames with MediaFrameReader

Communication

Article • 12/22/2023

This article provides an index of development features that are related to scenarios involving communication in Windows apps.

> **Note**
>
> The **Windows App SDK** currently does not provide APIs related to communication scenarios.

Windows OS features

Windows 10 and later OS releases provide a wide variety of APIs related to communication scenarios for apps. These features are available via a combination of WinRT and Win32 (C++ and COM) APIs provided by the Windows SDK.

WinRT APIs

The following articles provide information about features available via WinRT APIs provided by the Windows SDK.

Expand table

Article	Description
App-to-app communication	Learn how to share data between apps, including how to use the Share contract, copy and paste, drag and drop, and app services.
Interprocess communication	Learn about ways to perform interprocess communication (IPC) between UWP apps, packaged desktop apps, and unpackaged desktop apps.
Networking and web services	Learn about networking and web services technologies that are available to apps.
Sharing named objects	Learn how to share named objects between UWP apps, packaged desktop apps, and unpackaged desktop apps.

Win32 (C++ and COM) APIs

The following articles provide information about features available via Win32 (C++ and COM) APIs provided by the Windows SDK.

Expand table

Article	Description
Networking and Internet	Learn about APIs, components, and services that support your app's use of networking and the Internet.
Remote Procedure Call	Learn about using Remote Procedure Call (RPC) to create distributed client/server programs.
Windows Sockets 2 (Winsock)	Learn how to use Windows Sockets 2 (Winsock) to create advanced Internet, intranet, and other network-capable apps.

.NET features

The .NET SDK also provides APIs related to communication scenarios for WPF and Windows Forms apps.

Expand table

Article	Description
Network programming in the .NET Framework	Learn about building network-enabled apps using .NET.
Networking in Windows Forms	Learn about additional networking scenarios for Windows Forms apps.

Data and files

Article • 06/29/2023

This article provides an index of development features that are related to scenarios involving data and files in Windows apps.

Windows App SDK features

The Windows App SDK provides the following features related to data and files scenarios for Windows 10 and later OS releases.

Expand table

Feature	Description
Manage resources with MRT Core	Use the *MRT Core* APIs in the Microsoft.Windows.ApplicationModel.Resources namespace to manage app resources such as strings and images in multiple languages, scales, and contrast variants independently of your app's logic. MRT Core is a streamlined version of the older resource management APIs in the Windows.ApplicationModel.Resources of the Windows platform.

Windows OS features

Windows 10 and later OS releases provide a wide variety of APIs related to data and files scenarios for apps. These features are available via a combination of WinRT and Win32 (C++ and COM) APIs provided by the Windows SDK.

WinRT APIs

The following articles provide information about features available via WinRT APIs provided by the Windows SDK.

Expand table

Article	Description
App resources	Learn how to use the APIs in the Windows.ApplicationModel.Resources namespace and other related namespaces to manage app resources such as strings and images in multiple languages, scales, and contrast variants independently of your app's logic.

Article	Description
Data binding	Learn how to bind your app's UI to data and keep the UI in sync with changes to the data.
Files, folders, and libraries	Learn how to read and write text and other data formats in files, and to manage files and folders.

Win32 (C++ and COM) APIs

The following articles provide information about features available via Win32 (C++ and COM) APIs provided by the Windows SDK.

Expand table

Article	Description
Menus and other resources	Learn how to manage app resources including icons, cursors, menus, dialog boxes, bitmaps, and much more.
Data access and storage	Learn about data access and storage scenarios involving file and file system management, database access, and much more.

.NET features

The .NET SDK also provides APIs related to data and files scenarios for WPF and Windows Forms apps.

Expand table

Article	Description
Resources in .NET apps	Learn general strategies for managing app resources in .NET apps.
Application resource, content, and data files (WPF)	Learn how to manage app resources in WPF apps.
Application settings (Windows Forms)	Learn how to create, store, and maintain custom application and user preferences in Windows Forms apps.
LINQ	Learn how to use Language-Integrated Query (LINQ) to perform language-level data queries in .NET apps.
XML documents and data	Learn how to parse and write XML, edit XML data in memory, validate XML data, and perform XSLT transformations in .NET apps.

Article	Description
Entity Framework Core	Learn how to work with databases using Entity Framework Core in .NET apps.
Data binding (WPF)	Learn how to bind your WPF app's UI to data and keep the UI in sync with changes to the data.
Data binding (Windows Forms)	Learn how to bind your Windows Forms app's UI to data and keep the UI in sync with changes to the data.
File and stream I/O	Learn how to read and write on data streams and files in .NET apps, both synchronously and asynchronously.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Manage resources with MRT Core

Article • 03/13/2023

MRT Core is a streamlined version of the modern Windows Resource Management System that is distributed as part of the Windows App SDK.

MRT Core has both build-time and run-time features. At build time, the system creates an index of all the different variants of the resources that are packaged up with your app. This index is the Package Resource Index, or PRI, and it's also included in your app's package.

Prerequisites

To use MRT Core APIs in the Windows App SDK:

1. Download and install the latest release of the Windows App SDK. For more information, see Get started with WinUI.
2. Follow the instructions to Create your first WinUI 3 project or to use the Windows App SDK in an existing project.

To learn more about the availability of MRT Core in the Windows App SDK, see release channels.

Package Resource Index (PRI) file

Every app package should contain a binary index of the resources in the app. This index is created at build time and it is contained in one or more PRI files. Each PRI file contains a named collection of resources, referred to as a resource map.

A PRI file contains actual string resources. Embedded binary and file path resources are indexed directly from the project files. A package typically contains a single PRI file per language, named **resources.pri**. The **resources.pri** file at the root of each package is automatically loaded when the ResourceManager object is instantiated.

PRI files contain only data, so they don't use the portable executable (PE) format. They are specifically designed to be data-only.

> **Note**
>
> For .NET apps, in Windows App SDK version 0.8 and onward, the **Build Action** file property for resource files in Visual Studio is automatically set, reducing the need

for manual project configuration. Version 1.0 introduced issue 1674. This is fixed in 1.1 (from the stable channel), but the fix requires .NET SDK 6.0.300 or later. If you're using a lower version of the .NET SDK, please continue to use the workaround in the 1.0 release notes.

Access app resources with MRT Core

MRT Core provides several different ways to access your app resources.

> **Note**
>
> In Windows App SDK 1.0 Preview 1 and later releases, MRT Core APIs are in the Microsoft.Windows.ApplicationModel.Resources namespace. In earlier releases, they are in the Microsoft.ApplicationModel.Resources namespace.

Basic functionality with ResourceLoader

The simplest way to access your app resources programmatically is by using the ResourceLoader class. **ResourceLoader** provides you basic access to string resources from the set of resource files, referenced libraries, or other packages.

Advanced functionality with ResourceManager

The ResourceManager class provides additional info about resources, such as enumeration and inspection. This goes beyond what the **ResourceLoader** class provides.

A ResourceCandidate object represents a single concrete resource value and its qualifiers, such as the string "Hello World" for English, or "logo.scale-100.jpg" as a qualified image resource that's specific to the scale-100 resolution.

Resources available to an app are stored in hierarchical collections, which you can access with a ResourceMap object. The **ResourceManager** class provides access to the various top-level **ResourceMap** instances used by the app, which correspond to the various packages for the app. The ResourceManager.MainResourceMap value corresponds to the resource map for the current app package, and it excludes any referenced framework packages. Each **ResourceMap** is named for the package name that is specified in the package's manifest. Within a **ResourceMap** are subtrees (see ResourceMap.GetSubtree). The subtrees typically correspond to the resource files that contains the resource.

The **ResourceManager** not only supports access to an app's string resources, it also maintains the ability to enumerate and inspect the various file resources as well. In order to avoid collisions between files and other resources that originate from within a file, indexed file paths all reside within a reserved "Files" **ResourceMap** subtree. For example, the file '\Images\logo.png' corresponds to the resource name 'Files/images/logo.png'.

Qualify resource selection with ResourceContext

Resource candidates are chosen based on a particular ResourceContext, which is a collection of resource qualifier values (language, scale, contrast, and so on). A default context uses the app's current configuration for each qualifier value, unless overridden. For example, resources such as images can be qualified for scale, which varies from one monitor to another and hence from one application view to another. For this reason, each application view has a distinct default context. Whenever you retrieve a resource candidate, you should pass in a **ResourceContext** instance to obtain the most appropriate value for a given view.

Sample

For a sample that demonstrates how to use the MRT Core API, see the MRT Core sample.

See also

- UWP Resource Management System
- Localize strings in your UI and app package manifest
- Load images and assets tailored for scale, theme, high contrast, and others
- Tailor your resources for language, scale, high contrast, and other qualifiers

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tailor your resources for language, scale, high contrast, and other qualifiers

Article • 08/19/2024

This topic explains the general concept of resource qualifiers, how to use them, and the purpose of each of the qualifier names. See **ResourceContext.QualifierValues** for a reference table of all the possible qualifier values.

Your app can load assets and resources that are tailored to runtime contexts such as display language, high contrast, display scale factor, and many others. The way you do this is to name your resources' folders or files to match the qualifier names and qualifier values that correspond to those contexts. For example, you may want your app to load a different set of image assets in high contrast mode.

For more info about the value proposition of localizing your app, see Globalization and localization.

Qualifier name, qualifier value, and qualifier

A qualifier name is a key that maps to a set of qualifier values. Here are the qualifier name and qualifier values for contrast.

Expand table

Context	Qualifier name	Qualifier values
The high contrast setting	contrast	standard, high, black, white

You combine a qualifier name with a qualifier value to form a qualifier. `<qualifier name>-<qualifier value>` is the format of a qualifier. `contrast-standard` is an example of a qualifier.

So, for high contrast, the set of qualifiers is `contrast-standard`, `contrast-high`, `contrast-black`, and `contrast-white`. Qualifier names and qualifier values are not case sensitive. For example, `contrast-standard` and `Contrast-Standard` are the same qualifier.

Use qualifiers in folder names

Here is an example of using qualifiers to name folders that contain asset files. Use qualifiers in folder names if you have several asset files per qualifier. That way, you set

the qualifier once at the folder level, and the qualifier applies to everything inside the folder.

Console

```
\Assets\Images\contrast-standard\<logo.png, and other image files>
\Assets\Images\contrast-high\<logo.png, and other image files>
\Assets\Images\contrast-black\<logo.png, and other image files>
\Assets\Images\contrast-white\<logo.png, and other image files>
```

If you name your folders as in the example above, then your app uses the high contrast setting to load resource files from the folder named for the appropriate qualifier. So, if the setting is High Contrast Black, then the resource files in the `\Assets\Images\contrast-black` folder are loaded. If the setting is None (that is, the computer is not in high contrast mode), then the resource files in the `\Assets\Images\contrast-standard` folder are loaded.

Use qualifiers in file names

Instead of creating and naming folders, you can use a qualifier to name the resource files themselves. You might prefer to do this if you only have one resource file per qualifier. Here's an example.

Console

```
\Assets\Images\logo.contrast-standard.png
\Assets\Images\logo.contrast-high.png
\Assets\Images\logo.contrast-black.png
\Assets\Images\logo.contrast-white.png
```

The file whose name contains the qualifier most appropriate for the setting is the one that is loaded. This matching logic works the same way for file names as for folder names.

Reference a string or image resource by name

See the following topics for more info about how to reference a string or image resource by name:

- Refer to a string resource identifier from XAML markup
- Refer to a string resource identifier from code
- Reference an image or other asset from XAML markup and code

Actual and neutral qualifier matches

You don't need to provide a resource file for *every* qualifier value. For example, if you find that you only need one visual asset for high contrast and one for standard contrast, then you can name those assets like this.

Console

```
\Assets\Images\logo.contrast-high.png
\Assets\Images\logo.png
```

The first file name contains the `contrast-high` qualifier. That qualifier is an *actual* match for any high contrast setting when high contrast is *on*. In other words, it's a close match so it's preferred. An *actual* match can only occur if the qualifier contains an *actual* value, as this one does. In this case, `high` is an *actual* value for `contrast`.

The file named `logo.png` has no contrast qualifier on it at all. The absence of a qualifier is a *neutral* value. If no preferred match can be found, then the neutral value serves as a fallback match. In this example, if high contrast is *off*, then there is no actual match. The *neutral* match is the best match that can be found, and so the asset `logo.png` is loaded.

If you were to change the name of `logo.png` to `logo.contrast-standard.png`, then the file name would contain an actual qualifier value. With high contrast off, there would be an actual match with `logo.contrast-standard.png`, and that's the asset file that would be loaded. So, the same files would be loaded, under the same conditions, but because of different matches.

If you only need one set of assets for high contrast and one set for standard contrast, then you can use folder names instead of file names. In this case, omitting the folder name entirely gives you the neutral match.

Console

```
\Assets\Images\contrast-high\<logo.png, and other images to load when high
contrast theme is not None>
\Assets\Images\<logo.png, and other images to load when high contrast theme
is None>
```

For more details on how qualifier matching works, see Manage resources with MRT Core.

Multiple qualifiers

You can combine qualifiers in folder and file names. For example, you may want your app to load image assets when high contrast mode is on *and* the display scale factor is 400. One way to do this is with nested folders.

```
Console

\Assets\Images\contrast-high\scale-400\<logo.png, and other image files>
```

For `logo.png` and the other files to be loaded, the settings must match *both* qualifiers.

Another option is to combine multiple qualifiers in one folder name.

```
Console

\Assets\Images\contrast-high_scale-400\<logo.png, and other image files>
```

In a folder name, you combine multiple qualifiers separated with an underscore. `<qualifier1>[_<qualifier2>...]` is the format.

You can combine multiple qualifiers in a file name in the same format.

```
Console

\Assets\Images\logo.contrast-high_scale-400.png
```

Depending on the tools and workflow you use for asset-creation, or on what you find easiest to read and/or manage, you can either choose a single naming strategy for all qualifiers, or you can combine them for different qualifiers.

AlternateForm

The `alternateform` qualifier is used to provide an alternate form of a resource for some special purpose. This is typically used only by Japanese app developers to provide a furigana string for which the value `msft-phonetic` is reserved (see the section "Support Furigana for Japanese strings that can be sorted" in How to prepare for localization).

Either your target system or your app must provide a value against which `alternateform` qualifiers are matched. Do not use the `msft-` prefix for your own custom `alternateform` qualifier values.

Configuration

It's unlikely that you'll need the `configuration` qualifier name. It can be used to specify resources that are applicable only to a given authoring-time environment, such as test-only resources.

The `configuration` qualifier is used to load a resource that best matches the value of the `MS_CONFIGURATION_ATTRIBUTE_VALUE` environment variable. So, you can set the variable to the string value that has been assigned to the relevant resources, for example `designer`, or `test`.

Contrast

The `contrast` qualifier is used to provide resources that best match high contrast settings.

DXFeatureLevel

It's unlikely that you'll need the `dxfeaturelevel` qualifier name. It was designed to be used with Direct3D game assets, to cause downlevel resources to be loaded to match a particular downlevel hardware configuration of the time. But the prevalence of that hardware configuration is now so low that we recommend you don't use this qualifier.

HomeRegion

The `homeregion` qualifier corresponds to the user's setting for country or region. It represents the home location of the user. Values include any valid BCP-47 region tag. That is, any **ISO 3166-1 alpha-2** two-letter region code, plus the set of **ISO 3166-1 numeric** three-digit geographic codes for composed regions (see United Nations Statistic Division M49 composition of region codes). Codes for "Selected economic and other groupings" are not valid.

Language

A `language` qualifier corresponds to the display language setting. Values include any valid BCP-47 language tag. For a list of languages, see the IANA language subtag registry.

If you want your app to support different display languages, and you have string literals in your code or in your XAML markup, then move those strings out of the code/markup

and into a Resources File (`.resw`). You can then make a translated copy of that Resources File for each language that your app supports.

You typically use a `language` qualifier to name the folders that contain your Resources Files (`.resw`).

```
Console

\Strings\language-en\Resources.resw
\Strings\language-ja\Resources.resw
```

You can omit the `language-` part of a `language` qualifier (that is, the qualifier name). You can't do this with the other kinds of qualifiers; and you can only do it in a folder name.

```
Console

\Strings\en\Resources.resw
\Strings\ja\Resources.resw
```

Instead of naming folders, you can use `language` qualifiers to name the Resources Files themselves.

```
Console

\Strings\Resources.language-en.resw
\Strings\Resources.language-ja.resw
```

See Localize your UI strings for more information on making your app localizable by using string resources, and how to reference a string resource in your app.

LayoutDirection

A `layoutdirection` qualifier corresponds to the layout direction of the display language setting. For example, an image may need to be mirrored for a right-to-left language such as Arabic or Hebrew. Layout panels and images in your UI will respond to layout direction appropriately if you set their FlowDirection property (see Adjust layout and fonts, and support RTL). However, the `layoutdirection` qualifier is for cases where simple flipping isn't adequate, and it allows you to respond to the directionality of specific reading order and text alignment in more general ways.

Scale

Windows automatically selects a scale factor for each display based on its DPI (dots-per-inch) and the viewing distance of the device. See Effective pixels and scale factor. You should create your images at several recommended sizes (at least 100, 200, and 400) so that Windows can either choose the perfect size or can use the nearest size and scale it. So that Windows can identify which physical file contains the correct size of image for the display scale factor, you use a `scale` qualifier. The scale of a resource matches the value of DisplayInformation.ResolutionScale, or the next-largest-scaled resource.

Here's an example of setting the qualifier at the folder level.

Console

```
\Assets\Images\scale-100\<logo.png, and other image files>
\Assets\Images\scale-200\<logo.png, and other image files>
\Assets\Images\scale-400\<logo.png, and other image files>
```

And this example sets it at the file level.

Console

```
\Assets\Images\logo.scale-100.png
\Assets\Images\logo.scale-200.png
\Assets\Images\logo.scale-400.png
```

For info about qualifying a resource for both `scale` and `targetsize`, see Qualify an image resource for targetsize.

TargetSize

The `targetsize` qualifier is primarily used to specify file type association icons or protocol icons to be shown in File Explorer. The qualifier value represents the side length of a square image in raw (physical) pixels. The resource whose value matches the View setting in File Explorer is loaded; or the resource with the next-largest value in the absence of an exact match.

You can define assets that represent several sizes of `targetsize` qualifier value for the App Icon (`/Assets/Square44x44Logo.png`) in the Visual Assets tab of the app package manifest designer.

For info about qualifying a resource for both `scale` and `targetsize`, see Qualify an image resource for targetsize.

Theme

The `theme` qualifier is used to provide resources that best match the default app mode setting, or your app's override using Application.RequestedTheme.

Shell light theme and unplated resources

The *Windows 10 May 2019 Update* introduced a new "light" theme for the Windows Shell. As a result, some application assets that were previously shown on a dark background will now be shown on a light background. For apps that apps that provided altform-unplated assets for the taskbar and window switchers (Alt+Tab, Task View, etc), you should verify that they have acceptable contrast on a light background.

Providing light theme specific assets

Apps that want to provide a tailored resource for shell light theme can use a new alternate form resource qualifier: `altform-lightunplated`. This qualifier mirrors the existing altform-unplated qualifier.

Downlevel considerations

Apps should not use the `theme-light` qualifier with the `altform-unplated` qualifier. This will cause unpredictable behavior on RS5 and earlier versions of Windows due to the way resources are loaded for the Taskbar. On earlier versions of windows, the theme-light version may be used incorrectly. The `altform-lightunplated` qualifier avoids this issue.

Compatibility behavior

For backwards compatibility, Windows includes logic to detect a monochromatic icons and check whether it contrasts with the intended background. If the icon fails to meet contrast requirements, Windows will look for a contrast-white version of the asset. If that's not available, Windows will fall back to using the plated version of the asset.

Important APIs

- ResourceContext.QualifierValues

Related topics

- Effective pixels and scale factor
- Manage resources with MRT Core
- How to prepare for localization
- Localize your UI strings
- BCP-47
- United Nations Statistic Division M49 composition of region codes
- IANA language subtag registry
- Adjust layout and fonts, and support RTL

Localize strings in your UI and the app package manifest

Article • 08/19/2024

For more info about the value proposition of localizing your Windows App SDK app, see Globalization and localization.

If you want your app to support different display languages, and you have string literals in your code or XAML markup or app package manifest, then move those strings into a Resources File (.resw). You can then make a translated copy of that Resources File for each language that your app supports.

Hardcoded string literals can appear in imperative code or in XAML markup, for example as the **Text** property of a **TextBlock**. They can also appear in your app package manifest source file (the `Package.appxmanifest` file), for example as the value for Display name on the Application tab of the Visual Studio Manifest Designer. Move these strings into a Resources File (.resw), and replace the hardcoded string literals in your app and in your manifest with references to resource identifiers.

Unlike image resources, where only one image resource is contained in an image resource file, *multiple* string resources are contained in a string resource file. A string resource file is a Resources File (.resw), and you typically create this kind of resource file in a \Strings folder in your project. For background on how to use qualifiers in the names of your Resources Files (.resw), see Tailor your resources for language, scale, and other qualifiers.

Store strings in a resources file

1. Set your app's default language.
   a. With your solution open in Visual Studio, open `Package.appxmanifest`.
   b. On the Application tab, confirm that the Default language is set appropriately (for example, "en" or "en-US"). The remaining steps will assume that you have set the default language to "en-US".

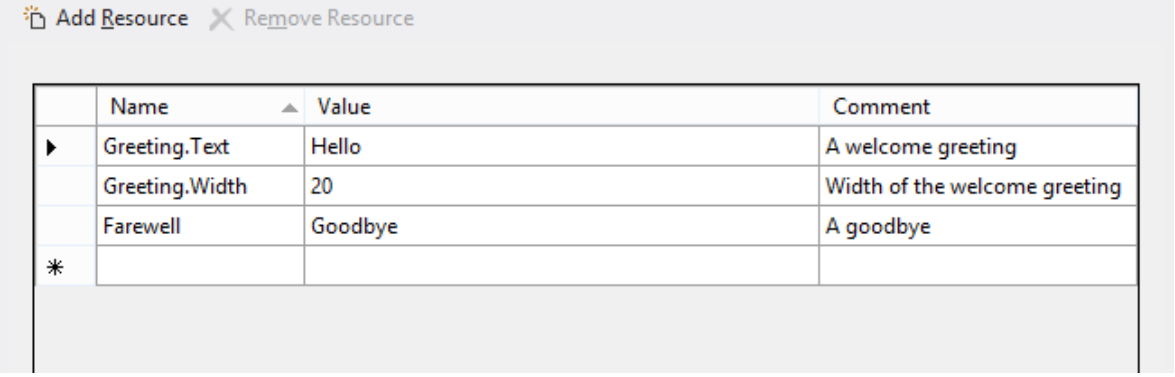
> **Note**
>
> At a minimum, you need to provide string resources localized for this default language. Those are the resources that will be loaded if no better match can be found for the user's preferred language or display language settings.

2. Create a Resources File (.resw) for the default language.
   a. Under your project node, create a new folder and name it `Strings`.
   b. Under `Strings`, create a new sub-folder and name it `en-US`.
   c. Under `en-US`, create a new Resources File (.resw) (under the **WinUI** file types in the **Add New Item** dialog) and confirm that it is named `Resources.resw`.

> **Note**
>
> If you have .NET Resources Files (.resx) that you want to port, see Porting XAML and UI.

3. Open `Resources.resw` and add these string resources.

`Strings/en-US/Resources.resw`



In this example, "Greeting" is a string resource identifier that you can refer to from your markup, as we'll show. For the identifier "Greeting", a string is provided for a Text property, and a string is provided for a Width property. "Greeting.Text" is an example of a property identifier because it corresponds to a property of a UI element. You could also, for example, add "Greeting.Foreground" in the Name column, and set its Value to "Red". The "Farewell" identifier is a simple string resource identifier; it has no sub-properties and it can be loaded from imperative code, as we'll show. The Comment column is a good place to provide any special instructions to translators.

In this example, since we have a simple string resource identifier entry named "Farewell", we cannot *also* have property identifiers based on that same identifier. So, adding "Farewell.Text" would cause a Duplicate Entry error when building `Resources.resw`.

Resource identifiers are case insensitive, and must be unique per resource file. Be sure to use meaningful resource identifiers to provide additional context for translators. And don't change the resource identifiers after the string resources are

sent for translation. Localization teams use the resource identifier to track additions, deletions, and updates in the resources. Changes in resource identifiers —which is also known as "resource identifiers shift"—require strings to be retranslated, because it will appear as though strings were deleted and others added.

Refer to a string resource identifier from XAML

You use an x:Uid directive to associate a control or other element in your markup with a string resource identifier.

XAML

```
<TextBlock x:Uid="Greeting"/>
```

At run-time, `\Strings\en-US\Resources.resw` is loaded (since right now that's the only Resources File in the project). The **x:Uid** directive on the **TextBlock** causes a lookup to take place, to find property identifiers inside `Resources.resw` that contain the string resource identifier "Greeting". The "Greeting.Text" and "Greeting.Width" property identifiers are found and their values are applied to the **TextBlock**, overriding any values set locally in the markup. The "Greeting.Foreground" value would be applied, too, if you'd added that. But only property identifiers are used to set properties on XAML markup elements, so setting **x:Uid** to "Farewell" on this TextBlock would have no effect. `Resources.resw` *does* contain the string resource identifier "Farewell", but it contains no property identifiers for it.

When assigning a string resource identifier to a XAML element, be certain that *all* the property identifiers for that identifier are appropriate for the XAML element. For example, if you set `x:Uid="Greeting"` on a **TextBlock** then "Greeting.Text" will resolve because the **TextBlock** type has a Text property. But if you set `x:Uid="Greeting"` on a **Button** then "Greeting.Text" will cause a run-time error because the **Button** type does not have a Text property. One solution for that case is to author a property identifier named "ButtonGreeting.Content", and set `x:Uid="ButtonGreeting"` on the **Button**.

Instead of setting **Width** from a Resources File, you'll probably want to allow controls to dynamically size to content.

> ⓘ **Note**
>
> For attached properties, you need a special syntax in the **Name** column of a .resw file. For example, to set a value for the AutomationProperties.Name attached

property for the "Greeting" identifier, this is what you would enter in the **Name** column.

XML

```
Greeting.[using:Microsoft.UI.Xaml.Automation]AutomationProperties.Name
```

Refer to a string resource identifier from code

You can explicitly load a string resource based on a simple string resource identifier.

C#

```
var resourceLoader = new
Microsoft.Windows.ApplicationModel.Resources.ResourceLoader();
this.myXAMLTextBlockElement.Text = resourceLoader.GetString("Farewell");
```

You can use this same code from within a Class Library project. At runtime, the resources of the app that's hosting the library are loaded. We recommend that a library loads resources from the app that hosts it, since the app is likely to have a greater degree of localization. If a library does need to provide resources then it should give its hosting app the option to replace those resources as an input.

If a resource name is segmented (it contains "." characters), then replace dots with forward slash ("/") characters in the resource name. Property identifiers, for example, contain dots; so you'd need to do this substitution in order to load one of those from code.

C#

```
this.myXAMLTextBlockElement.Text = resourceLoader.GetString("Fare/Well"); //
<data name="Fare.Well" ...> ...
```

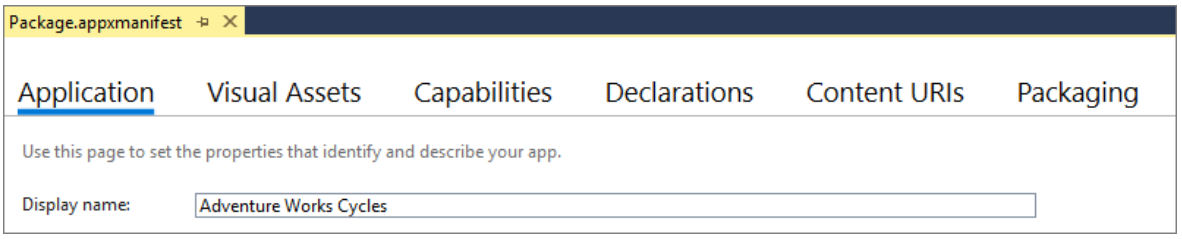
If in doubt, you can use MakePri.exe to dump your app's PRI file. Each resource's `uri` is shown in the dumped file.

XML

```
<ResourceMapSubtree name="Fare"><NamedResource name="Well" uri="ms-
resource://<GUID>/Resources/Fare/Well">...
```

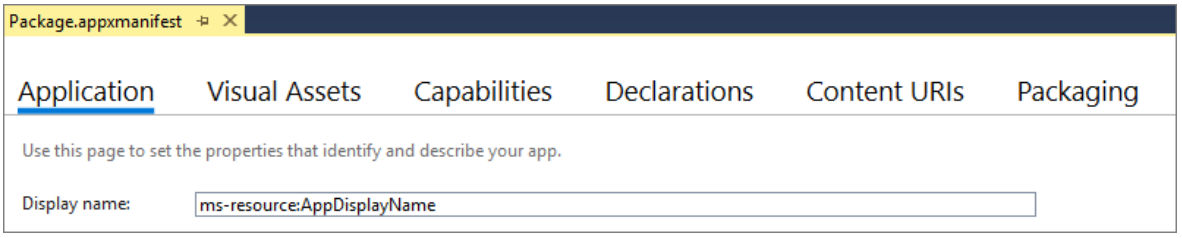
Refer to a string resource identifier from your app package manifest

1. Open your app package manifest source file (the `Package.appxmanifest` file), in which by default your app's `Display name` is expressed as a string literal.



2. To make a localizable version of this string, open `Resources.resw` and add a new string resource with the name "AppDisplayName" and the value "Adventure Works Cycles".

3. Replace the Display name string literal with a reference to the string resource identifier that you just created ("AppDisplayName"). You use the `ms-resource` URI (Uniform Resource Identifier) scheme to do this.



4. Repeat this process for each string in your manifest that you want to localize. For example, your app's Short name (which you can configure to appear on your app's tile on Start). For a list of all items in the app package manifest that you can localize, see Localizable manifest items.

Localize the string resources

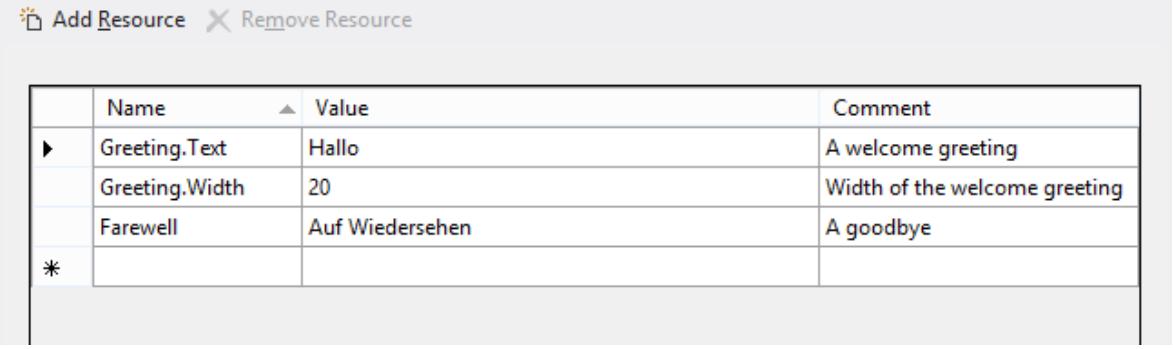
1. Make a copy of your Resources File (.resw) for another language.
   a. Under "Strings", create a new sub-folder and name it "de-DE" for Deutsch (Deutschland).

> **Note**
>
> For the folder name, you can use any BCP-47 language tag. See Tailor your resources for language, scale, and other qualifiers for details on the

language qualifier and a list of common language tags. 2. Make a copy of `Strings/en-US/Resources.resw` in the `Strings/de-DE` folder.

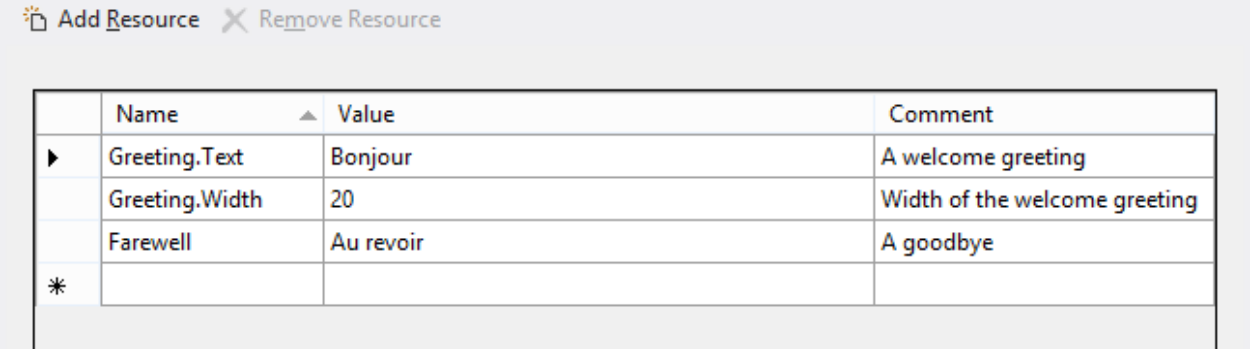
2. Translate the strings.
   a. Open `Strings/de-DE/Resources.resw` and translate the values in the Value column. You don't need to translate the comments.

`Strings/de-DE/Resources.resw`



If you like, you can repeat steps 1 and 2 for a further language.

`Strings/fr-FR/Resources.resw`



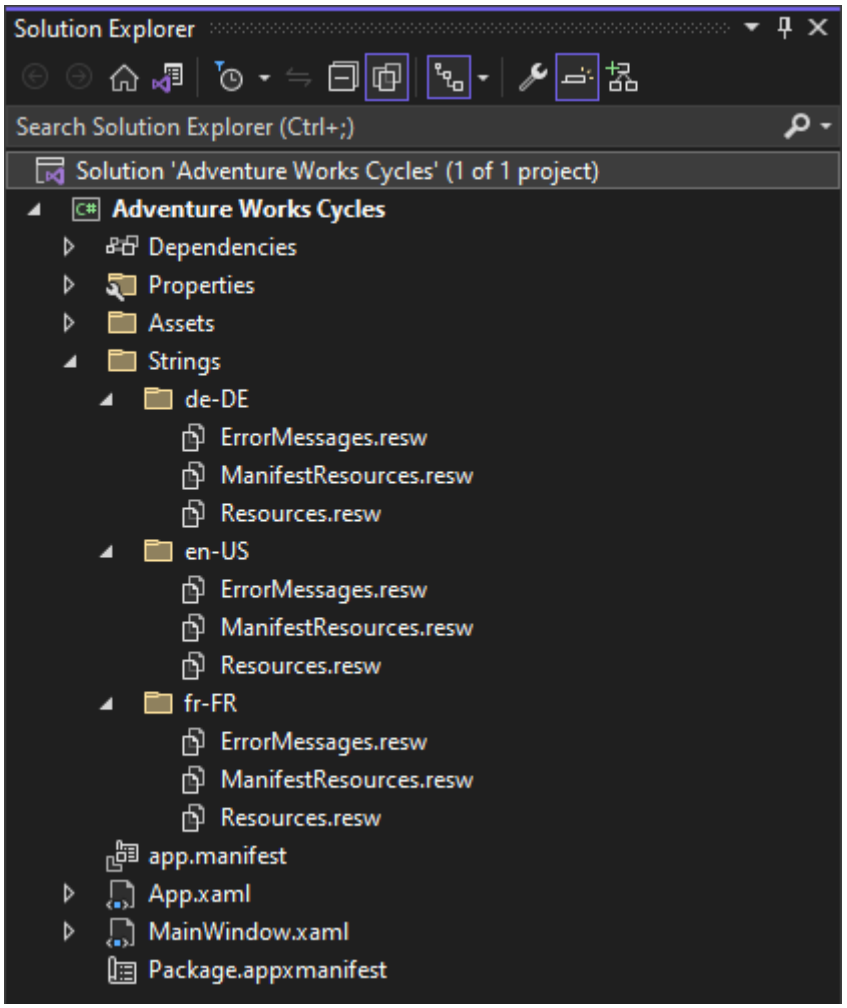
Test your app

Test the app for your default display language. You can then change the display language in **Settings** > **Time & Language** > **Region & language** > **Languages** and re-test your app. Look at strings in your UI and also in the shell (for example, your title bar—which is your Display name—and the Short name on your tiles).

> **Note**
>
> If a folder name can be found that matches the display language setting, then the Resources File inside that folder is loaded. Otherwise, fallback takes place, ending with the resources for your app's default language.

Factoring strings into multiple Resources Files

You can keep all of your strings in a single Resources File (resw), or you can factor them across multiple Resources Files. For example, you might want to keep your error messages in one Resources File, your app package manifest strings in another, and your UI strings in a third. This is what your folder structure would look like in that case.



To scope a string resource identifier reference to a particular file, you just add `/<resources-file-name>/` before the identifier. The markup example below assumes that `ErrorMessages.resw` contains a resource whose name is "PasswordTooWeak.Text" and whose value describes the error.

XAML

```
<TextBlock x:Uid="/ErrorMessages/PasswordTooWeak"/>
```

You only need to add `/<resources-file-name>/` before the string resource identifier for Resources Files *other than* `Resources.resw`. That's because "Resources.resw" is the default file name, so that's what's assumed if you omit a file name (as we did in the earlier examples in this topic).

The code example below assumes that `ErrorMessages.resw` contains a resource whose name is "MismatchedPasswords" and whose value describes the error.

C#

```
var resourceLoader = new
Microsoft.Windows.ApplicationModel.Resources.ResourceLoader("ErrorMessages")
;
this.myXAMLTextBlockElement.Text =
resourceLoader.GetString("MismatchedPasswords");
```

If you were to move your "AppDisplayName" resource out of `Resources.resw` and into `ManifestResources.resw`, then in your app package manifest you would change `ms-resource:AppDisplayName` to `ms-resource:/ManifestResources/AppDisplayName`.

If a resource file name is segmented (it contains "." characters), then leave the dots in the name when you reference it. **Don't** replace dots with forward slash ("/") characters, like you would for a resource name.

C#

```
var resourceLoader = new
Microsoft.Windows.ApplicationModel.Resources.ResourceLoader("Err.Msgs");
```

If in doubt, you can use MakePri.exe to dump your app's PRI file. Each resource's `uri` is shown in the dumped file.

XML

```
<ResourceMapSubtree name="Err.Msgs"><NamedResource
name="MismatchedPasswords" uri="ms-
resource://<GUID>/Err.Msgs/MismatchedPasswords">...
```

Load a string for a specific language or other context

The default ResourceContext (obtained when creating a ResourceLoader) contains a qualifier value for each qualifier name, representing the default runtime context (in other words, the settings for the current user and machine). Resources Files (.resw) are matched—based on the qualifiers in their names—against the qualifier values in that runtime context.

But there might be times when you want your app to override the system settings and be explicit about the language, scale, or other qualifier value to use when looking for a matching Resources File to load. For example, you might want your users to be able to select an alternative language for tooltips or error messages.

You can do that by constructing a new **ResourceContext**, overriding its values, and then using that context object in your string lookups.

```
C#

var resourceManager = new
Microsoft.Windows.ApplicationModel.Resources.ResourceManager();
var resourceContext = resourceManager.CreateResourceContext();
resourceContext.QualifierValues["Language"] = "de-DE";
var resourceMap = resourceManager.MainResourceMap.GetSubtree("Resources");
this.myXAMLTextBlockElement.Text = resourceMap.GetValue("Farewell",
resourceContext).ValueAsString;
```

Using **QualifierValues** as in the code example above works for any qualifier. For the special case of Language, you can alternatively do this instead.

```
C#

resourceContext.Languages = new string[] { "de-DE" };
```

Load strings from a Class Library

The string resources of a referenced Class Library are typically added into a subfolder of the package in which they're included during the build process. The resource identifier of such a string usually takes the form *LibraryName/ResourcesFileName/ResourceIdentifier*.

A library can get a **ResourceLoader** for its own resources. For example, the following code illustrates how either a library or an app that references it can get a **ResourceLoader** for the library's string resources.

```
C#

var resourceLoader = new
Microsoft.Windows.ApplicationModel.Resources.ResourceLoader("ContosoControl/
Resources");
this.myXAMLTextBlockElement.Text =
resourceLoader.GetString("exampleResourceName");
```

If in doubt about the path, you can specify MakePri.exe command line options to dump your component or library's PRI file. Each resource's `uri` is shown in the dumped file.

```xml
<NamedResource name="exampleResourceName" uri="ms-
resource://Contoso.Control/Contoso.Control/ReswFileName/exampleResourceName"
>...
```

Loading strings from other packages

The resources for an app package are managed and accessed through the package's own top-level ResourceMap that's accessible from the ResourceManager. Within each package, various components can have their own **ResourceMap** subtrees, which you can access via ResourceMap.GetSubtree.

A framework package can access its own resources with an absolute resource identifier URI. Also see URI schemes in the UWP documentation for more information.

Loading strings in unpackaged applications

As of Windows Version 1903 (May 2019 Update), unpackaged applications can also leverage the Resource Management System.

Just create your Windows App SDK user controls/libraries and store any strings in a resources file. You can then refer to a string resource identifier from XAML, refer to a string resource identifier from code, or load strings from a Class Library.

To use resources in unpackaged applications, you should do a few things:

1. Use the overloaded constructor of **ResourceManager** to pass file name of your app's .pri file when resolving resources from code as there is no default view in unpackaged scenarios.
2. Use MakePri.exe to manually generate your app's resources.pri file.

    - Run `makepri new /pr <PROJECTROOT> /cf <PRICONFIG> /of resources.pri`
    - The <PRICONFIG> must omit the "<packaging>" section so that all resources are bundled in a single resources.pri file. If using the default MakePri.exe configuration file created by createconfig, you need to delete the "<packaging>" section manually after it is created.
    - The <PRICONFIG> must contain all relevant indexers required to merge all resources in your project into a single resources.pri file. The default

MakePri.exe configuration file created by createconfig includes all indexers.

- If you don't use the default config, make sure the PRI indexer is enabled (review the default config for how to do this) to merge PRIs found from project references, NuGet references, and so on, that are located within the project root.

> **Note**
>
> By omitting `/IndexName`, and by the project not having an app manifest, the IndexName/root namespace of the PRI file is automatically set to *Application*, which the runtime understands for unpackaged apps (this removes the previous hard dependency on package ID). When specifying resource URIs, ms-resource:/// references that omit the root namespace infer *Application* as the root namespace for unpackaged apps (or you can specify *Application* explicitly as in ms-resource://Application/).

3. Copy the PRI file to the build output directory of the .exe
4. Run the .exe

> **Note**
>
> The Resource Management System uses the system display language rather than the user preferred language list when resolving resources based on language in unpackaged apps. The user preferred language list is only used for Windows App SDK packaged apps.

> **Important**
>
> You must manually rebuild PRI files whenever resources are modified. We recommend using a post-build script that handles the MakePri.exe command and copies the resources.pri output to the .exe directory.

Important APIs

- ApplicationModel.Resources.ResourceLoader

See also

- MRT Core sample
- How to load string resources

Load images and assets tailored for scale, theme, high contrast, and others

Article • 10/08/2024

Your app can load image resource files (or other asset files) tailored for display scale factor, theme, high contrast, and other runtime contexts. These images can be referenced from imperative code or from XAML markup, for example as the **Source** property of an Image. They can also appear in your app package manifest source file (the `Package.appxmanifest` file)—for example, as the value for App Icon on the Visual Assets tab of the Visual Studio Manifest Designer—or on your tiles and toasts. By using qualifiers in your images' file names, and optionally dynamically loading them with the help of a ResourceContext, you can cause the most appropriate image file to be loaded that best matches the user's runtime settings for display scale, theme, high contrast, language, and other contexts.

An image resource is contained in an image resource file. You can also think of the image as an asset, and the file that contains it as an asset file; and you can find these kinds of resource files in your project's \Assets folder. For background on how to use qualifiers in the names of your image resource files, see Tailor your resources for language, scale, and other qualifiers.

Some common qualifiers for images are:

- scale
- theme
- contrast
- targetsize

Qualify an image resource for scale, theme, and contrast

The default value for the `scale` qualifier is `scale-100`. So, these two variants are equivalent (they both provide an image at scale 100, or scale factor 1).

Console

```
\Assets\Images\logo.png
\Assets\Images\logo.scale-100.png
```

You can use qualifiers in folder names instead of file names. This is a better strategy when you have several asset files per qualifier. For purposes of illustration, these two variants are equivalent to the two above.

Console

```
\Assets\Images\logo.png
\Assets\Images\scale-100\logo.png
```

The next example shows how you can provide variants of an image resource—named `/Assets/Images/logo.png`—for different settings of display scale, theme, and high contrast. This example uses folder naming.

Console

```
\Assets\Images\contrast-standard\theme-dark
    \scale-100\logo.png
    \scale-200\logo.png
\Assets\Images\contrast-standard\theme-light
    \scale-100\logo.png
    \scale-200\logo.png
\Assets\Images\contrast-high
    \scale-100\logo.png
    \scale-200\logo.png
```

Reference an image or other asset from XAML markup and code

The name—or identifier—of an image resource is its path and file name with any and all qualifiers removed. If you name folders and/or files as in any of the examples in the previous section, then you have a single image resource and its name (as an absolute path) is `/Assets/Images/logo.png`. Here's how you use that name in XAML markup.

XAML

```
<Image x:Name="myXAMLImageElement" Source="ms-
appx:///Assets/Images/logo.png"/>
```

Notice that you use the `ms-appx` URI scheme because you're referring to a file that comes from your app's package. See URI schemes in the UWP documentation. This is how you refer to the same image resource in imperative code.

C#

```
this.myXAMLImageElement.Source = new BitmapImage(new Uri("ms-
appx:///Assets/Images/logo.png"));
```

You can use `ms-appx` to load any arbitrary file from your app package.

C#

```
var uri = new System.Uri("ms-appx:///Assets/anyAsset.ext");
var storagefile = await
Windows.Storage.StorageFile.GetFileFromApplicationUriAsync(uri);
```

The `ms-appx-web` scheme accesses the same files as `ms-appx`, but in the web compartment.

XAML

```
<WebView x:Name="myXAMLWebViewElement" Source="ms-appx-
web:///Pages/default.html"/>
```

C#

```
this.myXAMLWebViewElement.Source = new Uri("ms-appx-
web:///Pages/default.html");
```

For any of the scenarios shown in these examples, use the Uri constructor overload that infers the UriKind. Specify a valid absolute URI including the scheme and authority, or just let the authority default to the app's package as in the example above.

Notice how in these example URIs the scheme ("`ms-appx`" or "`ms-appx-web`") is followed by "`://`" which is followed by an absolute path. In an absolute path, the leading "`/`" causes the path to be interpreted from the root of the package.

> **Note**
>
> The `ms-resource` (for string resources) and `ms-appx(-web)` (for images and other assets) URI schemes perform automatic qualifier matching to find the resource that's most appropriate for the current context. The `ms-appdata` URI scheme (which is used to load app data) does not perform any such automatic matching, but you can respond to the contents of ResourceContext.QualifierValues and explicitly load the appropriate assets from app data using their full physical file name in the URI. For info about app data, see Store and retrieve settings and other app data. Web URI schemes (for example, `http`, `https`, and `ftp`) do not perform automatic

matching, either. For info about what to do in that case, see Hosting and loading images in the cloud.

Absolute paths are a good choice if your image files remain where they are in the project structure. If you want to be able to move an image file, but you're careful that it remains in the same location relative to its referencing XAML markup file, then instead of an absolute path you might want to use a path that's relative to the containing markup file. If you do that, then you needn't use a URI scheme. You will still benefit from automatic qualifier matching in this case, but only because you are using the relative path in XAML markup.

XAML

```
<Image Source="Assets/Images/logo.png"/>
```

Also see Tile and toast support for language, scale, and high contrast.

Reference an image or other asset from a class library

You can load images and other resources from a referenced **Class library (WinUI 3 in Desktop)** project by referencing the resource in a URI that uses the `ms-appx` scheme. The URI should include the name of the class library project and the path to the resource within the class library project. For example, if you have a class library project named `MyClassLibrary` that contains an image named `logo.png` in a folder named `Assets`, you can reference the image in your app project like this:

XAML

```
<Image Source="ms-appx:///MyClassLibrary/Assets/logo.png"/>
```

You'll use this same URI format to reference resources in a class library from XAML markup or from code. For example, you can use the following code to load the image from your class library and put it into a StorageFile object:

C#

```
private async Task<DateTimeOffset> GetLogoCreatedDateAsync()
{
    Uri uri = new($"ms-appx:///MyClassLibrary/Assets/logo.png");
    Windows.Storage.StorageFile file =
        await
```

```
Windows.Storage.StorageFile.GetFileFromApplicationUriAsync(uri);

    return file.DateCreated;
}
```

Note that you can reference images from the class library from both the app project and the class library project itself.

Qualify an image resource for targetsize

You can use the `scale` and `targetsize` qualifiers on different variants of the same image resource; but you can't use them both on a single variant of a resource. Also, you need to define at least one variant without a `targetsize` qualifier. That variant must either define a value for `scale`, or let it default to `scale-100`. So, these two variants of the `/Assets/Square44x44Logo.png` resource are valid.

Console

```
\Assets\Square44x44Logo.scale-200.png
\Assets\Square44x44Logo.targetsize-24.png
```

And these two variants are valid.

Console

```
\Assets\Square44x44Logo.png // defaults to scale-100
\Assets\Square44x44Logo.targetsize-24.png
```

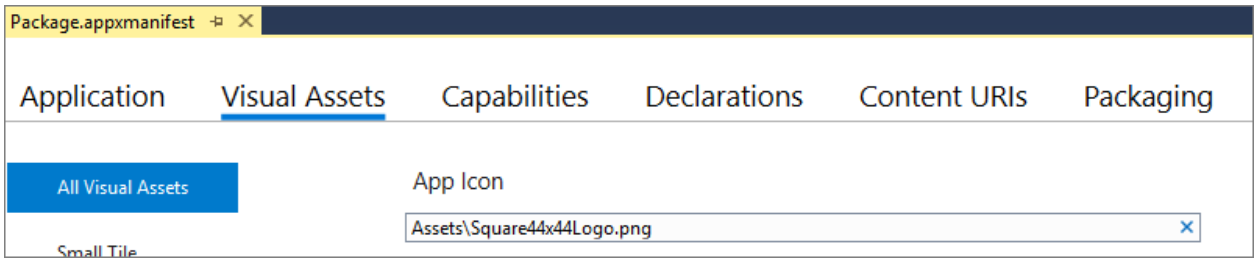
But this variant is not valid.

Console

```
\Assets\Square44x44Logo.scale-200_targetsize-24.png
```

Refer to an image file from your app package manifest

If you name folders and/or files as in either of the two valid examples in the previous section, then you have a single app icon image resource and its name (as a relative path) is `Assets\Square44x44Logo.png`. In your app package manifest, simply refer to the resource by name. There's no need to use any URI scheme.



That's all you need to do, and the OS will perform automatic qualifier matching to find the resource that's most appropriate for the current context. For a list of all items in the app package manifest that you can localize or otherwise qualify in this way, see Localizable manifest items.

Qualify an image resource for layoutdirection

See Mirroring images.

Load an image for a specific language or other context

For more info about the value proposition of localizing your app, see Globalization and localization.

The default ResourceContext contains a qualifier value for each qualifier name, representing the default runtime context (in other words, the settings for the current user and machine). Image files are matched—based on the qualifiers in their names—against the qualifier values in that runtime context.

But there might be times when you want your app to override the system settings and be explicit about the language, scale, or other qualifier value to use when looking for a matching image to load. For example, you might want to control exactly when and which high contrast images are loaded.

You can do that by constructing a new **ResourceContext** (instead of using the default one), overriding its values, and then using that context object in your ResourceMap image lookups with GetValue or TryGetValue.

C#

```
var resourceManager = new
Microsoft.Windows.ApplicationModel.Resources.ResourceManager();
var resourceContext = resourceManager.CreateResourceContext();
resourceContext.QualifierValues["Contrast"] = "high";
var resourceMap = resourceManager.MainResourceMap;
var namedResource = resourceMap.TryGetValue(@"Files/Assets/Logo.png",
```

```
resourceContext);
var imageFileBytes = namedResource.ValueAsBytes;

using (var ms = new InMemoryRandomAccessStream())
{
    using (var writer = new DataWriter(ms.GetOutputStreamAt(0)))
    {
        writer.WriteBytes(imageFileBytes);
        writer.StoreAsync().GetResults();
    }
    var image = new BitmapImage();
    image.SetSource(ms);
    this.myXAMLImageElement.Source = image;
}
```

By default, the ResourceManager class uses the default **ResourceContext**.

Important APIs

The following APIs are imporant to understand when working with image resources:

- ResourceContext
- ResourceMap

Related content

- Store and retrieve settings and other app data
- Tile and toast support for language, scale, and high contrast
- Mirroring images

Data access in Windows apps?

Article • 08/02/2024

You can store data on the user's device by using a SQLite database. You can also connect your Windows app directly to a SQL Server, Cosmos DB, MySQL, or MongoDB database without having to use an external service layer.

Topics in this section

In this section, you'll learn how to store and retrieve data in a Windows app by using different back-end databases. Each topic provides a guide to help you get started.

Expand table

Topic	Description
Use a SQLite database on Windows	Shows you how to use SQLite to store and retrieve data in a light-weight database on the user's device. SQLite is a server-less, embedded database engine.
Use a SQL Server database in a Windows app	Shows you how to connect directly to a SQL Server database and then store and retrieve data by using classes in the System.Data.SqlClient namespace. No service layer required.
Use a Cosmos DB database in a Windows app	Shows you how to work with a Cosmos DB database and test the connection programmatically. Cosmos DB is a cloud-based document database.
Use a MySQL database in a Windows app	Shows you how to connect to a MySQL database and interact with the database in a Windows application. MySQL is an open source, cross-platform relational database.
Use a MongoDB database in a Windows app	Shows you how to work with a MongoDB database and test the connection programmatically. MongoDB is a cross-platform document database.

Related content

- Customer Orders Database sample
- UWP data access
- .NET data documentation

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Use a SQLite database in a Windows app

Article • 08/01/2024

You can use SQLite to store and retrieve data in a lightweight database on the user's device. This guide shows you how to do it in your Windows App SDK apps.

Some benefits of using SQLite for local storage

✔ SQLite is lightweight and self-contained. It's a code library without any other dependencies. There's nothing to configure.

✔ There's no database server. The client and the server run in the same process.

✔ SQLite is in the public domain so you can freely use and distribute it with your app.

✔ SQLite works across platforms and architectures.

You can read more about SQLite here.

Choose an abstraction layer

We recommend that you use either Entity Framework Core or the open-source SQLite library built by Microsoft.

Entity Framework Core

Entity Framework (EF) is an object-relational mapper that you can use to work with relational data by using domain-specific objects. If you've already used this framework to work with data in other .NET apps, you can use the same code in your Windows App SDK app and it will work with appropriate changes to the connection string.

To try it out, see Getting Started with EF Core.

SQLite library

The Microsoft.Data.Sqlite library implements the interfaces in the System.Data.Common namespace. Microsoft actively maintains these implementations, and they provide an intuitive wrapper around the low-level native SQLite API.

The rest of this guide helps you to use this library.

Set up your solution to use the Microsoft.Data.SQlite library

We'll start with a basic Windows App SDK project, and then install the SQLite NuGet package.

All supported versions of Windows support SQLite, so your app does not have to package SQLite libraries. Instead, your app can use the version of SQLite that comes installed with Windows. This helps you in a few ways.

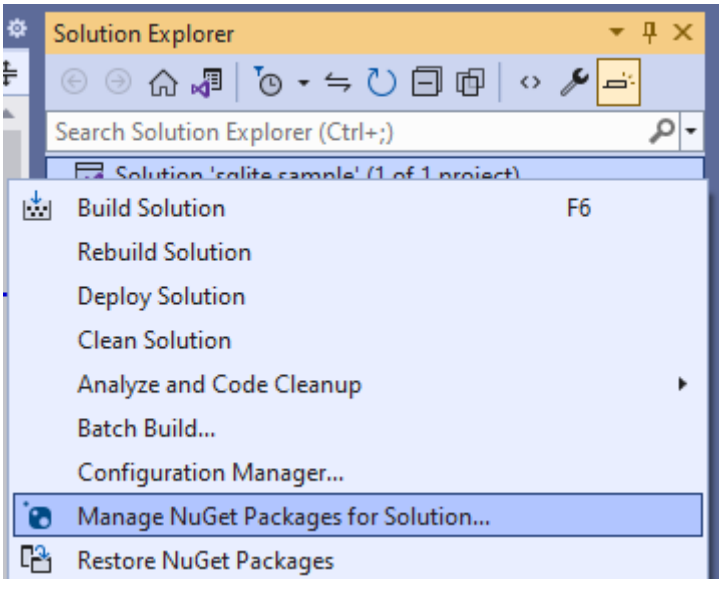
✔ Reduces the size of your application because you don't have to download the SQLite binary, and then package it as part of your application.

✔ Prevents you from having to push a new version of your app to users in the event that SQLite publishes critical fixes to bugs and security vulnerabilities in SQLite. The Windows version of SQLite is maintained by Microsoft in coordination with SQLite.org.

✔ App load time has the potential to be faster because most likely, the SDK version of SQLite will already be loaded into memory.

Let's start by adding a class to your project named **DataAccess**. If you plan to share your data access logic with other client code, you can use a .NET class library project to contain your data access code, but we won't use one in our example.

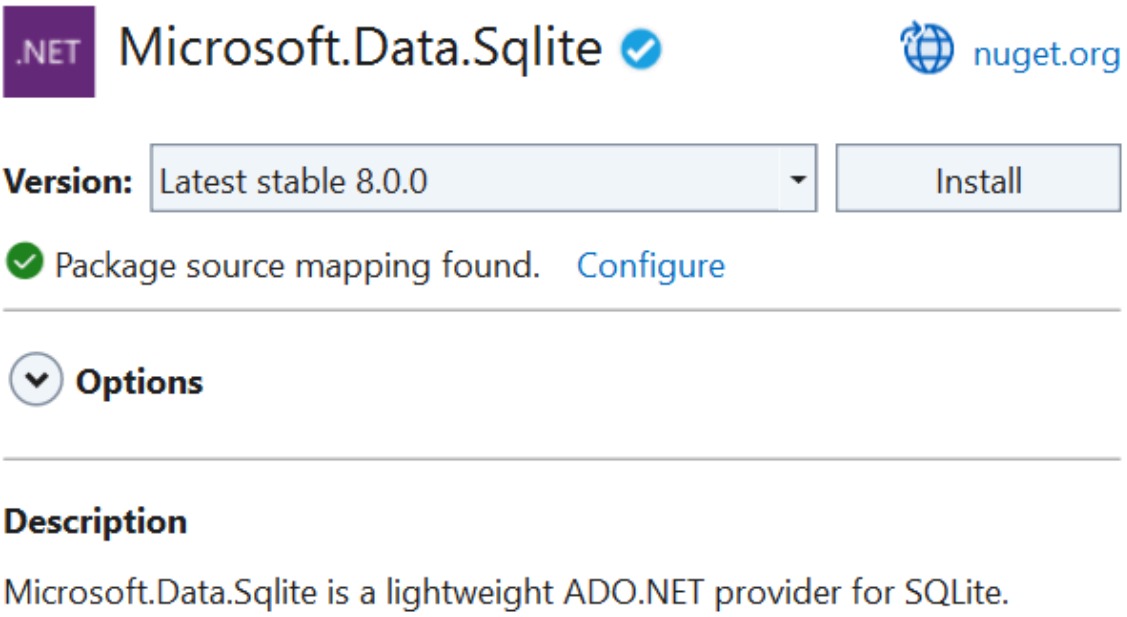
1. Right-click the solution, and then click **Manage NuGet Packages for Solution**.



   At this point, you have a choice. You can use the version of SQLite that is included with Windows or if you have some reason to use a specific version of SQLite, you

can include the SQLite library in your package. We are going to use the version of SQLite that's included with Windows.

2. Choose the **Browse** tab, search for the **Microsoft.Data.SQLite** package, and then install the latest stable version.



Add and retrieve data in a SQLite database

We'll do these things:

[1] Prepare the data access class.

[2] Initialize the SQLite database.

[3] Insert data into the SQLite database.

[4] Retrieve data from the SQLite database.

[5] Add a basic user interface.

Prepare the data access class

Open the `DataAccess` class in your project and make that class static.

> ⓘ **Note**
>
> While our example will place data access code in a static class, this is a design choice and is completely optional.

C#

```
public static class DataAccess
{
}
```

Add the following `using` statements to the top of this file.

C#

```
using Microsoft.Data.Sqlite;
using System.Collections.Generic;
```

Initialize the SQLite database

Add a method to the `DataAccess` class that initializes the SQLite database.

C#

```
public async static void InitializeDatabase()
{
    await ApplicationData.Current.LocalFolder
            .CreateFileAsync("sqliteSample.db",
CreationCollisionOption.OpenIfExists);
    string dbpath = Path.Combine(ApplicationData.Current.LocalFolder.Path,
                                 "sqliteSample.db");
    using (var db = new SqliteConnection($"Filename={dbpath}"))
    {
        db.Open();

        string tableCommand = "CREATE TABLE IF NOT " +
            "EXISTS MyTable (Primary_Key INTEGER PRIMARY KEY, " +
            "Text_Entry NVARCHAR(2048) NULL)";

        var createTable = new SqliteCommand(tableCommand, db);

        createTable.ExecuteReader();
    }
}
```

> **Note**
>
> The code above that uses `ApplicationData` members will only work for packaged apps running in an app container. All other Windows apps should access `ApplicationData` members via the ApplicationDataManager class.

This code creates the SQLite database and stores it in the application's local data store.

In this example, we name the database `sqlliteSample.db` but you can use whatever name you want as long as you use that name in all SqliteConnection objects that you instantiate. In a production application, connection information such as the database filename should be stored in app configuration rather than hard-coded (see **Adding Azure App Configuration by using Visual Studio Connected Services**).

In the constructor of the **App.xaml.cs** file of your project, call the `InitializeDatabase` method of the `DataAccess` class. This will ensure that the database is created or opened each time the app starts.

C#

```
public App()
{
    this.InitializeComponent();

    DataAccess.InitializeDatabase();
}
```

Insert data into the SQLite database

Add a method to the `DataAccess` class that inserts data into the SQLite database. This code uses parameters in the query to prevent SQL injection attacks.

C#

```
public static void AddData(string inputText)
{
    string dbpath = Path.Combine(ApplicationData.Current.LocalFolder.Path,
                                 "sqliteSample.db");
    using (var db = new SqliteConnection($"Filename={dbpath}"))
    {
        db.Open();

        var insertCommand = new SqliteCommand();
        insertCommand.Connection = db;

        // Use parameterized query to prevent SQL injection attacks
        insertCommand.CommandText = "INSERT INTO MyTable VALUES (NULL,
@Entry);";
        insertCommand.Parameters.AddWithValue("@Entry", inputText);

        insertCommand.ExecuteReader();
    }
```

```
}
```

Retrieve data from the SQLite database

Add a method that gets all rows of data from a table in our SQLite database.

C#

```
public static List<string> GetData()
{
    var entries = new List<string>();
    string dbpath = Path.Combine(ApplicationData.Current.LocalFolder.Path,
                                 "sqliteSample.db");
    using (var db = new SqliteConnection($"Filename={dbpath}"))
    {
        db.Open();
        var selectCommand = new SqliteCommand
            ("SELECT Text_Entry from MyTable", db);

        SqliteDataReader query = selectCommand.ExecuteReader();

        while (query.Read())
        {
            entries.Add(query.GetString(0));
        }
    }

    return entries;
}
```

The Read method advances through the rows of returned data. It returns `true` if there are rows left, otherwise it returns `false`.

The GetString method returns the value of the specified column as a string. It accepts an integer value that represents the zero-based column ordinal of the data that you want. You can use similar methods such as GetDataTime and GetBoolean. Choose a method based on what type of data the column contains.

The ordinal parameter isn't as important in this example because we are selecting all of the entries in a single column. However, if multiple columns are part of your query, use the ordinal value to obtain the column you want to pull data from.

Add a basic user interface

In the **MainWindow.xaml** file of the project, add the following XAML.

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <StackPanel>
        <TextBox x:Name="Input_Box"/>
        <Button Click="AddData">Add</Button>
        <ListView x:Name="Output">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding}"/>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackPanel>
</Grid>
```

This basic user interface gives the user a `TextBox` that they can use to type a string that we'll add to the SQLite database. We'll connect the `Button` in this UI to an event handler that will retrieve data from the SQLite database and then show that data in the `ListView`.

In the **MainWindow.xaml.cs** file, add the following handler. This is the method that we associated with the `Click` event of the `Button` in the UI.

C#

```
private void AddData(object sender, RoutedEventArgs e)
{
    DataAccess.AddData(Input_Box.Text);

    Output.ItemsSource = DataAccess.GetData();
}
```

We also want to be sure that any existing data is loaded when the application starts. Add a line of code to the `MainWindow` constructor to call `GetData()`.

C#

```
public MainWindow()
{
    this.InitializeComponent();

    Output.ItemsSource = DataAccess.GetData();
}
```

That's it. Explore the Microsoft.Data.Sqlite to see what other things you can do with your SQLite database. Check out the links below to learn about other ways to use data in

your Windows apps.

Related content

Connect your app directly to a SQL Server database

See Use a SQL Server database in a Windows app.

Share code between different apps across different platforms

See Share code between desktop and UWP.

Add master detail pages with Azure SQL back ends

See Customer Orders Database sample.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Use a SQL Server database in a Windows app

Article • 10/15/2024

Your app can connect directly to a SQL Server database and then store and retrieve data by using classes in the System.Data.SqlClient namespace.

Getting started

In this guide, we'll show you one way to do that in your Windows App SDK apps. If you install the Northwind sample database onto your SQL Server instance, and then use these snippets, you'll end up with a basic UI that shows products from the Northwind sample database.

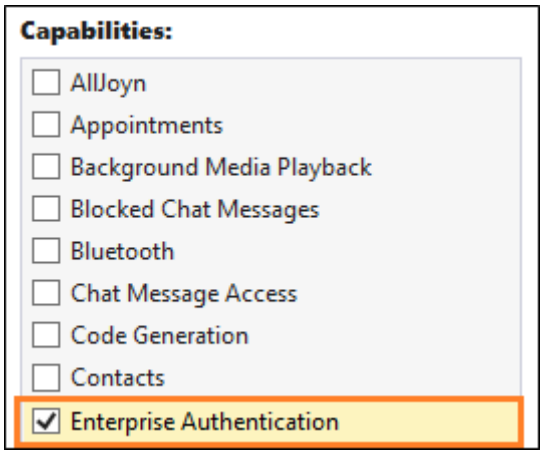
> **Tip**
>
> You can also get scripts to create the **Northwind** and **pubs** sample databases from the SQL Server Samples GitHub repository.

The snippets that appear in this guide are based on this UWP sample app.

First, set up your solution

To connect your app directly to a SQL Server database, your app can target any minimum version of Windows supported by Windows App SDK. You can find that information in the properties page of your project.

1. Open the **Package.appxmanifest** file of your Windows App SDK project in the manifest designer.
2. In the **Capabilities** tab, select the **Enterprise Authentication** checkbox if you are using Windows Authentication for authenticating your SQL Server.



> **ⓘ Important**
>
> You will also need to select **Internet (Client & Server)**, **Internet (Client)**, and **Private Networks (Client & Server)**, regardless of whether or not you're using Windows Authentication.

Add and retrieve data in a SQL Server database

In this section, we'll do these things:

[1] Add a connection string.

[2] Create a class to hold product data.

[3] Retrieve products from the SQL Server database.

[4] Add a basic user interface.

[5] Populate the UI with Products.

> **ⓘ Note**
>
> This section illustrates one way to organize your data access code. It's meant only to provide an example of how you can use System.Data.SqlClient to store and retrieve data from a SQL Server database. You can organize your code in any way that makes the most sense to your application's design.

Add a connection string

In the **App.xaml.cs** file, add a property to the `App` class, that gives other classes in your solution access to the connection string.

Our connection string points to the Northwind database in a SQL Server Express instance. The connection string in this snippet assumes you kept the default instance name `SQLEXPRESS` when installing SQL Server Express. You can make changes to the connection string to match your SQL Server instance, database, and authentication method.

C#

```
sealed partial class App : Application
{
    // Connection string for using Windows Authentication.
    private string connectionString =
        @"Data Source=.\SQLEXPRESS;Initial Catalog=NORTHWIND;Integrated
Security=SSPI";

    public string ConnectionString { get => connectionString; set =>
connectionString = value; }

    ...
}
```

> **ⓘ Important**
>
> In production applications, connection information should be stored securely in app configuration (see Adding Azure App Configuration by using Visual Studio Connected Services). Connection strings and other secrets should never be hard-coded.

Create a class to hold product data

We'll create a class that implements the INotifyPropertyChanged event so that we can bind attributes in our XAML UI to the properties in this class.

C#

```
public class Product : INotifyPropertyChanged
{
    public int ProductID { get; set; }
    public string ProductCode { get { return ProductID.ToString(); } }
    public string ProductName { get; set; }
```

```
    public string QuantityPerUnit { get; set; }
    public decimal UnitPrice { get; set; }
    public string UnitPriceString { get { return
UnitPrice.ToString("######.00"); } }
    public int UnitsInStock { get; set; }
    public string UnitsInStockString { get { return
UnitsInStock.ToString("#####0"); } }
    public int CategoryId { get; set; }

    public event PropertyChangedEventHandler PropertyChanged;
    private void NotifyPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
    }
}
```

Retrieve products from the SQL Server database

In the **MainWindow.xaml.cs** file of the Windows App SDK project, create a method that gets products from the Northwind sample database, and then returns them as an ObservableCollection collection of `Product` instances.

C#

```
public ObservableCollection<Product> GetProducts(string connectionString)
{
    const string GetProductsQuery = "select ProductID, ProductName,
QuantityPerUnit," +
       " UnitPrice, UnitsInStock, Products.CategoryID " +
       " from Products inner join Categories on Products.CategoryID =
Categories.CategoryID " +
       " where Discontinued = 0";

    var products = new ObservableCollection<Product>();
    try
    {
        using (var conn = new SqlConnection(connectionString))
        {
            conn.Open();
            if (conn.State == System.Data.ConnectionState.Open)
            {
                using (SqlCommand cmd = conn.CreateCommand())
                {
                    cmd.CommandText = GetProductsQuery;
                    using (SqlDataReader reader = cmd.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            var product = new Product();
                            product.ProductID = reader.GetInt32(0);
```

```
                            product.ProductName = reader.GetString(1);
                            product.QuantityPerUnit = reader.GetString(2);
                            product.UnitPrice = reader.GetDecimal(3);
                            product.UnitsInStock = reader.GetInt16(4);
                            product.CategoryId = reader.GetInt32(5);
                            products.Add(product);
                        }
                    }
                }
            }
        }
        return products;
    }
    catch (Exception eSql)
    {
        Debug.WriteLine($"Exception: {eSql.Message}");
    }
    return null;
}
```

Add a basic user interface

Add the following XAML to the **MainWindow.xaml** file of the Windows App SDK project.

This XAML creates a ListView to show each product that you returned in the previous snippet, and binds the attributes of each row in the `ListView` to the properties that we defined in the `Product` class.

XML

```
<Grid Background="{ThemeResource SystemControlAcrylicWindowBrush}">
    <RelativePanel>
        <ListView Name="InventoryList"
                  SelectionMode="Single"
                  ScrollViewer.VerticalScrollBarVisibility="Auto"
                  ScrollViewer.IsVerticalRailEnabled="True"
                  ScrollViewer.VerticalScrollMode="Enabled"
                  ScrollViewer.HorizontalScrollMode="Enabled"
                  ScrollViewer.HorizontalScrollBarVisibility="Auto"
                  ScrollViewer.IsHorizontalRailEnabled="True"
                  Margin="20">
            <ListView.HeaderTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal"  >
                        <TextBlock Text="ID" Margin="8,0" Width="50"
Foreground="DarkRed" />
                        <TextBlock Text="Product description" Width="300"
Foreground="DarkRed" />
                        <TextBlock Text="Packaging" Width="200"
Foreground="DarkRed" />
                        <TextBlock Text="Price" Width="80"
```

```
Foreground="DarkRed" />
                        <TextBlock Text="In stock" Width="80"
Foreground="DarkRed" />
                    </StackPanel>
                </DataTemplate>
            </ListView.HeaderTemplate>
            <ListView.ItemTemplate>
                <DataTemplate x:DataType="local:Product">
                    <StackPanel Orientation="Horizontal" >
                        <TextBlock Name="ItemId"
                                   Text="{x:Bind ProductCode}"
                                   Width="50" />
                        <TextBlock Name="ItemName"
                                   Text="{x:Bind ProductName}"
                                   Width="300" />
                        <TextBlock Text="{x:Bind QuantityPerUnit}"
                                   Width="200" />
                        <TextBlock Text="{x:Bind UnitPriceString}"
                                   Width="80" />
                        <TextBlock Text="{x:Bind UnitsInStockString}"
                                   Width="80" />
                    </StackPanel>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </RelativePanel>
</Grid>
```

Show products in the ListView

Open the **MainWindow.xaml.cs** file, and add code to the constructor of the `MainWindow` class that sets the **ItemSource** property of the ListView to the ObservableCollection of `Product` instances.

C#

```
public MainWindow()
{
    this.InitializeComponent();
    InventoryList.ItemsSource = GetProducts((App.Current as
App).ConnectionString);
}
```

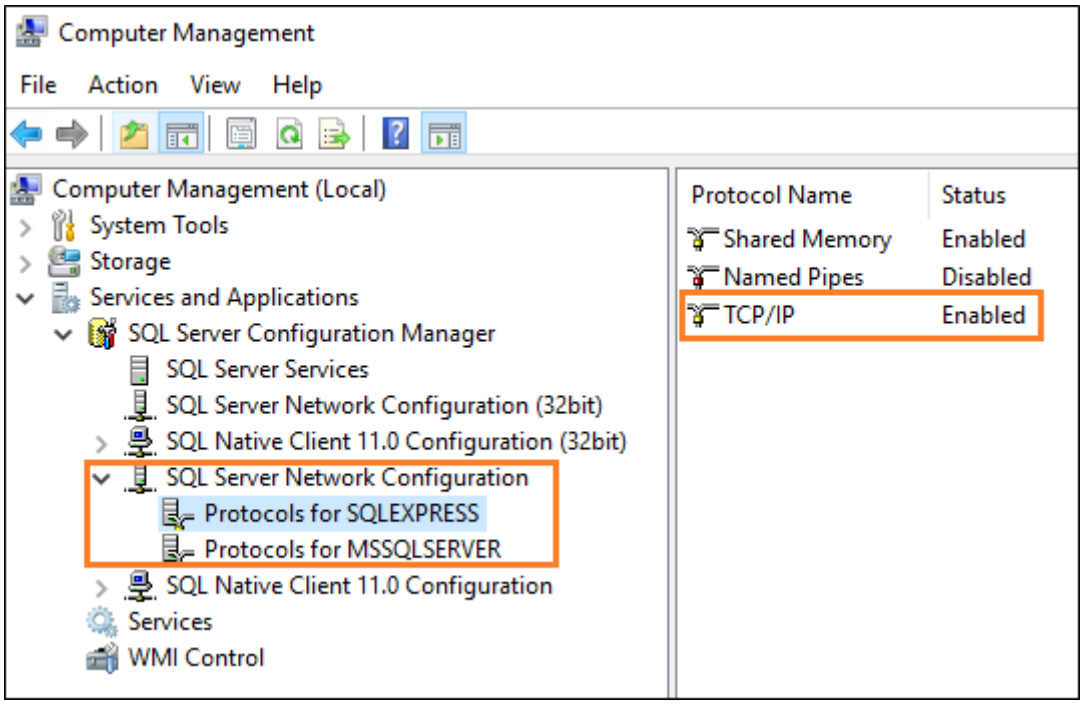
Start the project and see products from the Northwind sample database appear in the UI.

Explore the System.Data.SqlClient namespace to see what other things you can do with data in your SQL Server database.

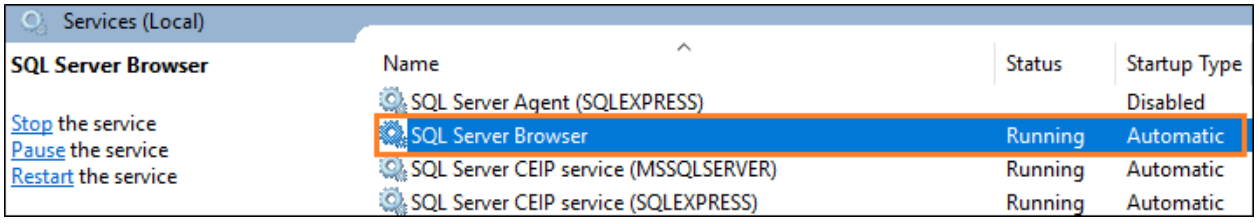
> **Tip**
>
> Try asking **Microsoft Copilot** for help with your SQL queries. Copilot can help you write SQL queries, and suggest ways to improve your code.

Trouble connecting to your database?

In most cases, some aspect of the SQL Server configuration needs to be changed. If you're able to connect to your database from another type of desktop application such as a Windows Forms or WPF application, ensure that you've enabled TCP/IP for SQL Server. You can do that in the **Computer Management** console. (See Windows Tools/Administrative Tools for more information.)



Then, make sure that your SQL Server Browser service is running.



Next step

Use a SQLite database in a Windows app

Use a Cosmos DB database from a Windows app

Article • 08/01/2024

This article contains the steps required to enable working with a Cosmos DB database from a Windows app. It also contains a small code snippet showing how you can interact with the database in code.

Set up your solution

This example can be used with any WPF, Windows Forms, WinUI 3, and UWP project to connect your Windows app to a Cosmos DB database. Follow these steps to install the package and try out example code for some basic tasks.

1. Open the **Package Manager Console** (View -> Other Windows -> Package Manager Console). Use the command `Install-Package Microsoft.Azure.Cosmos` to install the NuGet package for the **Azure Cosmos DB for NoSQL client library for .NET**. This will allow you to programmatically access Cosmos DB databases.
2. Build your project and make sure that the build was successful with no errors.

Next, you'll need to create a Cosmos DB instance in Azure. You can do this by following the steps in Create a NoSQL database account in Azure Cosmos DB.

Work with Cosmos DB using sample code

The following sample code gets a container from a Cosmos DB instance in Azure, then adds a new item to that container. Then it uses Cosmos DB's NoSQL query API to retrieve the item from the container and output the response status. Note that the endpoint, auth key, and database name will need to be customized based on the Cosmos DB instance you created in the previous section.

> **Note**
>
> For a complete example, including information about required Cosmos DB setup and configuration, see **Develop a .NET console application with Azure Cosmos DB for NoSQL**.

C#

```
using Microsoft.Azure.Cosmos;

...

public async Task CosmosSample(string endpoint, string authKey)
{
    // CONNECT
    var client = new CosmosClient(
        accountEndpoint: endpoint,
        authKeyOrResourceToken: authKey
    );
    Database database = client.GetDatabase("sample_customers");
    ContainerProperties properties = new(
        id: "customers",
        partitionKeyPath: "/location"
    );
    Container container = await
database.CreateContainerIfNotExistsAsync(properties);

    // WRITE DATA
    string customerId = "1234";
    string state = "WA";
    var customer = new
    {
        id = customerId,
        name = "John Doe",
        location = state
    };
    var createResponse = await container.CreateItemAsync(customer);
    Console.WriteLine($"[Status code:
{createResponse.StatusCode}]\t{customerId}");

    // READ DATA
    string sql = "SELECT * FROM customers c WHERE c.id = @id";
    var query = new QueryDefinition(
        query: sql
    ).WithParameter("@id", customerId);
    using var feed = container.GetItemQueryIterator<dynamic>
(queryDefinition: query);
    var queryResponse = await feed.ReadNextAsync();
    Console.WriteLine($"[Status code:
{queryResponse.StatusCode}]\t{customerId}");
}
```

Related content

- Azure Cosmos DB for NoSQL client library for .NET
- Use a MySQL database in a Windows app

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Use a MySQL database in a Windows app

Article • 02/04/2025

This article contains the steps required to enable working with a MySQL database from a Windows app. It also contains a small code snippet showing how you can interact with the database in code.

> **Tip**
>
> You can use AI assistance to create a MySQL connection string with GitHub Copilot.

Set up your solution

This example can be used with any WPF, Windows Forms, WinUI 3, and UWP project to connect your Windows app to a MySQL database. Follow these steps to install the package and try out the example code to read data from an existing MySQL database.

1. Open the **Package Manager Console** (View -> Other Windows -> Package Manager Console).
2. Use the command `Install-Package MySql.Data` to install the NuGet package for the MySQL core class library.

This will allow you to programmatically access MySQL databases.

> **Note**
>
> MySQL Connector/NET version 6.4.4 or later is required to use the `MySql.Data` package with Windows authentication.

Test your connection using sample code

The following is an example of connecting to and reading from a remote MySQL database. Note that the server address and database name will need to be customized.

```
C#
```

```
const string M_str_sqlcon =
"Server=myServerAddress;Database=myDataBase;IntegratedSecurity=yes;Uid=auth_
windows;";
using (var mySqlCn = new MySqlConnection(M_str_sqlcon))
{
    using (var mySqlCmd = new MySqlCommand("select * from table1", mySqlCn))
    {
        mySqlCn.Open();
        using (MySqlDataReader mySqlReader =
mySqlCmd.ExecuteReader(CommandBehavior.CloseConnection))
        {
            while (mySqlReader.Read())
            {
                Debug.WriteLine($"{mySqlReader.GetString(0)}:
{mySqlReader.GetString(1)}");
            }
        }
    }
}
```

> **ⓘ Important**
>
> In production applications, connection information should be stored securely in app configuration (see Adding Azure App Configuration by using Visual Studio Connected Services). Connection strings and other secrets should not be hard-coded.

Building a connection string with GitHub Copilot

You can use GitHub Copilot to build the connection string for your MySQL database. You can customize the prompt to create a connection string per your requirements.

The following text shows an example prompt for Copilot Chat that generates a connection string similar to the one shown in the previous code snippet:

Copilot prompt

```
Show me how to create a MySQL connection string to a server named
myServerAddress and a database called myDatabase. Use Windows
authentication.
```

GitHub Copilot is powered by AI, so surprises and mistakes are possible. For more information, see Copilot FAQs.

Learn more about GitHub Copilot in Visual Studio and GitHub Copilot in VS Code.

Related content

- Use a SQL Server database in a Windows app
- Data access in Windows apps

Use a MongoDB database in a Windows app

Article • 08/02/2024

This article contains the steps required to enable working with a MongoDB database from a Windows app. It also contains a small code snippet showing how you can interact with the database in code.

Set up your solution

This example can be used with any WPF, Windows Forms, WinUI 3, and UWP project to connect your Windows app to MongoDB. Follow these steps to install the package and try out the example code to read data from an existing MongoDB database.

1. Open the **Package Manager Console** (View -> Other Windows -> Package Manager Console).
2. Use the command `Install-Package MongoDB.Driver` to install the NuGet package for the official driver for MongoDB.

This will allow you to programmatically access MongoDB databases.

Test your connection using sample code

The following sample code gets a collection from a remote MongoDB client, then adds a new document to that collection. Then it uses MongoDB APIs to retrieve the new size of the collection as well as the inserted document, and prints them out.

C#

```
var client = new MongoClient("mongodb://10.xxx.xx.xxx:27017");
IMongoDatabase database = client.GetDatabase("foo");
IMongoCollection<BsonDocument> collection =
database.GetCollection<BsonDocument>("bar");
var document = new BsonDocument
{
     { "name","MongoDB"},
     { "type","Database"},
     { "count",1},
     { "info",new BsonDocument { { "x", 203 }, { "y", 102 } }}
};
collection.InsertOne(document);
long count = collection.CountDocuments(document);
Console.WriteLine(count);
```

```
IFindFluent<BsonDocument, BsonDocument> document1 =
collection.Find(document);
Console.WriteLine(document1.ToString());
```

Note that the IP address and database name will need to be customized. The port, 27017, is the default MongoDB port number. In a production application, connection information such as server address and database name should be stored in app configuration rather than hard-coded (see **Adding Azure App Configuration by using Visual Studio Connected Services**).

Related content

- Use a SQL Server database in a Windows app
- Use a MySQL database in a Windows app

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Data binding in Windows apps

Article • 05/17/2024

Data binding is a way for your app's UI to display data, and optionally to stay in sync with that data. Data binding allows you to create a separation of concerns between your data and UI, and that results in a simpler conceptual model as well as better readability, testability, and maintainability of your app. In XAML markup, you can choose to use either the {x:Bind} markup extension or the {Binding} markup extension. And you can even use a mixture of the two in the same app—even on the same UI element. `{x:Bind}` was new for UWP in Windows 10, is also available in Windows App SDK, and it has better performance.

Expand table

Topic	Description
Data binding overview	This topic shows you how to bind a control (or other UI element) to a single item or bind an items control to a collection of items in a Windows App SDK app. In addition, we show how to control the rendering of items, implement a details view based on a selection, and convert data for display. For more detailed info, see Data binding in depth.
Data binding in depth	This topic describes data binding features in detail.
Bind hierarchical data and create a master/details view	You can make a multi-level master/details (also known as list-details) view of hierarchical data by binding items controls to **CollectionViewSource** instances that are bound together in a chain.
Data binding and MVVM	This topic describes the Model-View-ViewModel (MVVM) UI architectural design pattern. Data binding is at the core of MVVM, and enables loose coupling between UI and non-UI code.
Functions in x:Bind	In Windows App SDK apps, `{x:Bind}` supports using a function as the leaf step of the binding path. In this topic, learn how properties are bound to functions to do conversions, date formatting, text formatting, text concatenations, etc.

See also

- Data binding in UWP apps

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Windows data binding overview

Article • 01/10/2025

This topic shows you how to bind a control (or other UI element) to a single item or bind an items control to a collection of items in a Windows App SDK app. In addition, we show how to control the rendering of items, implement a details view based on a selection, and convert data for display. For more detailed info, see Data binding in depth.

Prerequisites

This topic assumes that you know how to create a basic Windows App SDK app. For instructions on creating your first Windows App SDK app, see Create your first WinUI 3 (Windows App SDK) project.

Create the project

Create a new **Blank App, Packaged (WinUI 3 in Desktop)** C# project. Name it "Quickstart".

Binding to a single item

Every binding consists of a binding target and a binding source. Typically, the target is a property of a control or other UI element, and the source is a property of a class instance (a data model, or a view model). This example shows how to bind a control to a single item. The target is the `Text` property of a `TextBlock`. The source is an instance of a simple class named `Recording` that represents an audio recording. Let's look at the class first.

Add a new class to your project, and name the class `Recording`.

C#

```
namespace Quickstart
{
    public class Recording
    {
        public string ArtistName { get; set; }
        public string CompositionName { get; set; }
        public DateTime ReleaseDateTime { get; set; }
        public Recording()
```

```
        {
            ArtistName = "Wolfgang Amadeus Mozart";
            CompositionName = "Andante in C for Piano";
            ReleaseDateTime = new DateTime(1761, 1, 1);
        }
        public string OneLineSummary
        {
            get
            {
                return $"{CompositionName} by {ArtistName}, released: "
                    + ReleaseDateTime.ToString("d");
            }
        }
    }
    public class RecordingViewModel
    {
        private Recording defaultRecording = new Recording();
        public Recording DefaultRecording { get { return defaultRecording; }
}
    }
}
```

Next, expose the binding source class from the class that represents your window of markup. We do that by adding a property of type `RecordingViewModel` to **MainWindow.xaml.cs**.

C#

```
namespace Quickstart
{
    public sealed partial class MainWindow : Window
    {
        public MainWindow()
        {
            this.InitializeComponent();
            ViewModel = new RecordingViewModel();
        }
        public RecordingViewModel ViewModel{ get; set; }
    }
}
```

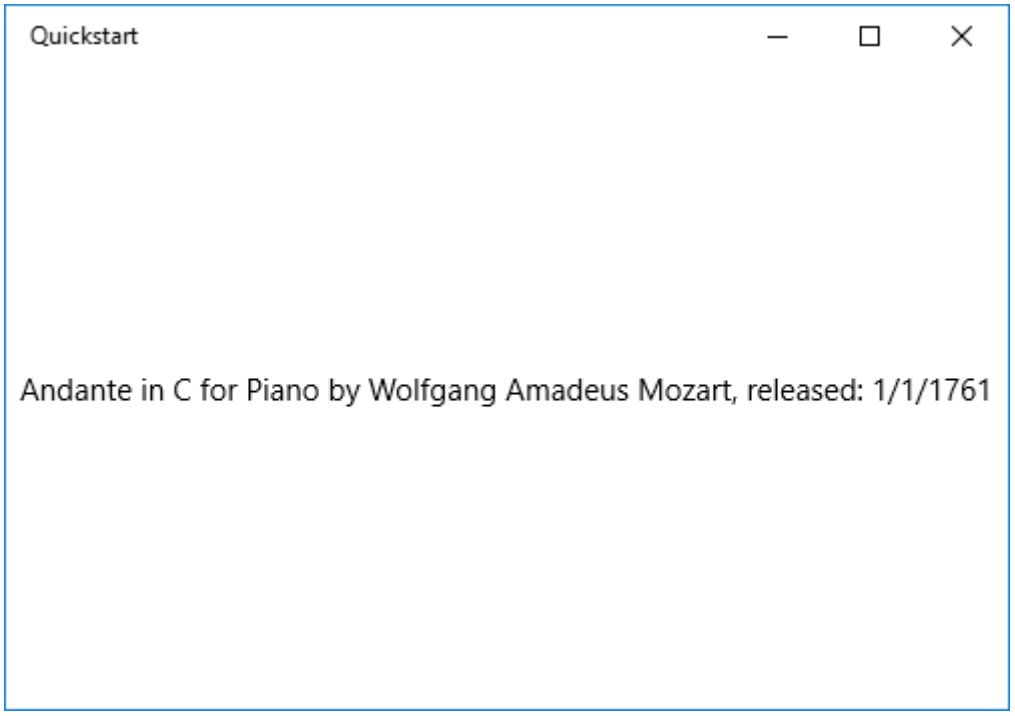
The last piece is to bind a `TextBlock` to the `ViewModel.DefaultRecording.OneLineSummary` property.

XAML

```
<Window x:Class="Quickstart.MainWindow" ... >
    <Grid>
        <TextBlock Text="{x:Bind ViewModel.DefaultRecording.OneLineSummary}"
    HorizontalAlignment="Center"
```

```
    VerticalAlignment="Center"/>
    </Grid>
</Window>
```

Here's the result.



Binding to a collection of items

A common scenario is to bind to a collection of business objects. In C#, the generic ObservableCollection<T> class is a good collection choice for data binding, because it implements the INotifyPropertyChanged and INotifyCollectionChanged interfaces. These interfaces provide change notification to bindings when items are added or removed or a property of the list itself changes. If you want your bound controls to update with changes to properties of objects in the collection, the business object should also implement `INotifyPropertyChanged`. For more info, see Data binding in depth.

This next example binds a ListView to a collection of `Recording` objects. Let's start by adding the collection to our view model. Just add these new members to the `RecordingViewModel` class.

C#

```
public class RecordingViewModel
{
    ...
    private ObservableCollection<Recording> recordings = new
ObservableCollection<Recording>();
    public ObservableCollection<Recording> Recordings{ get{ return
```

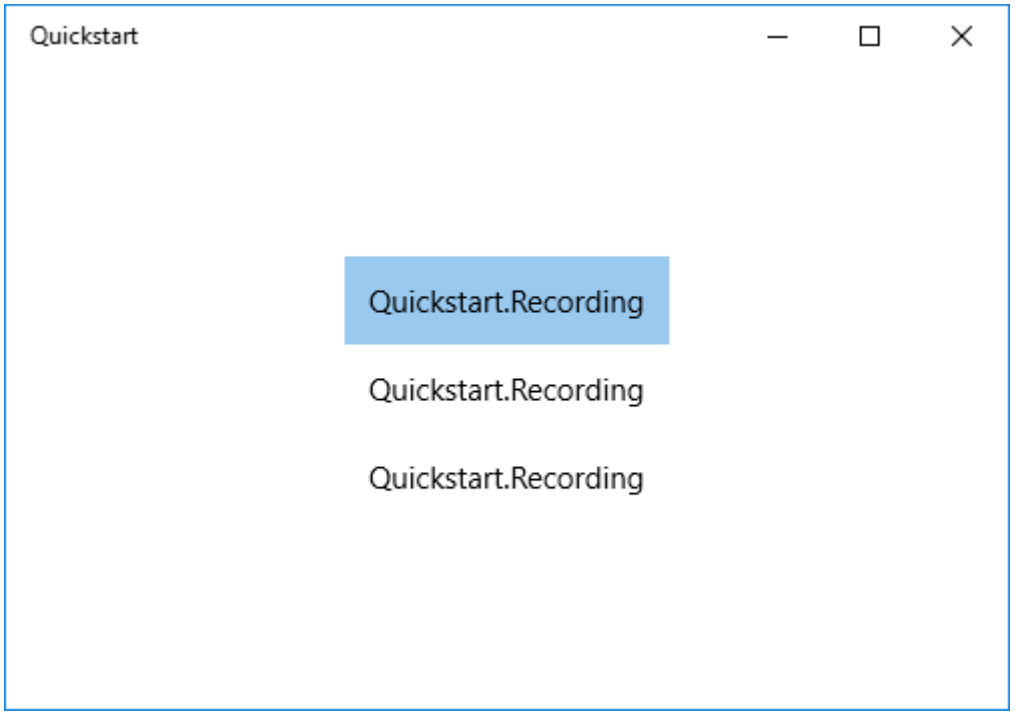
```
recordings; } }
    public RecordingViewModel()
    {
        recordings.Add(new Recording(){ ArtistName = "Johann Sebastian
Bach",
            CompositionName = "Mass in B minor", ReleaseDateTime = new
DateTime(1748, 7, 8) });
        recordings.Add(new Recording(){ ArtistName = "Ludwig van Beethoven",
            CompositionName = "Third Symphony", ReleaseDateTime = new
DateTime(1805, 2, 11) });
        recordings.Add(new Recording(){ ArtistName = "George Frideric
Handel",
            CompositionName = "Serse", ReleaseDateTime = new DateTime(1737,
12, 3) });
    }
}
```

And then bind a ListView to the `ViewModel.Recordings` property.

XAML

```
<Window x:Class="Quickstart.MainWindow" ... >
    <Grid>
        <ListView ItemsSource="{x:Bind ViewModel.Recordings}"
        HorizontalAlignment="Center" VerticalAlignment="Center"/>
    </Grid>
</Window>
```

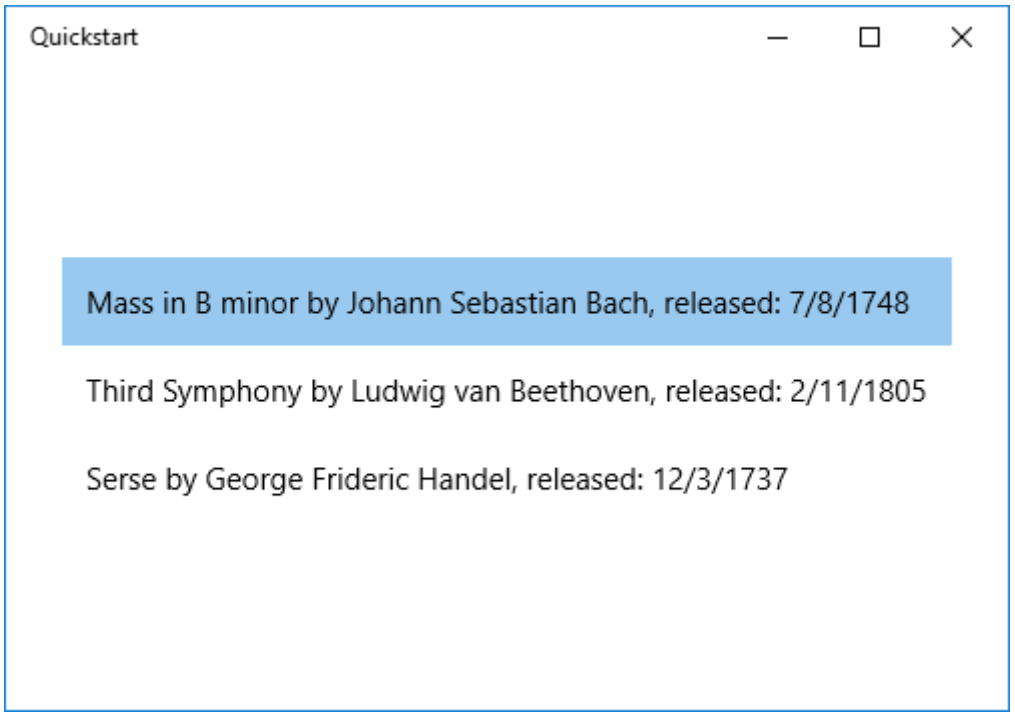
We haven't yet provided a data template for the `Recording` class, so the best the UI framework can do is to call ToString for each item in the ListView. The default implementation of `ToString` is to return the type name.



To remedy this, we can either override ToString to return the value of `OneLineSummary`, or we can provide a data template. The data template option is a more usual solution, and a more flexible one. You specify a data template by using the ContentTemplate property of a content control or the ItemTemplate property of an items control. Here are two ways we could design a data template for `Recording` together with an illustration of the result.

XAML

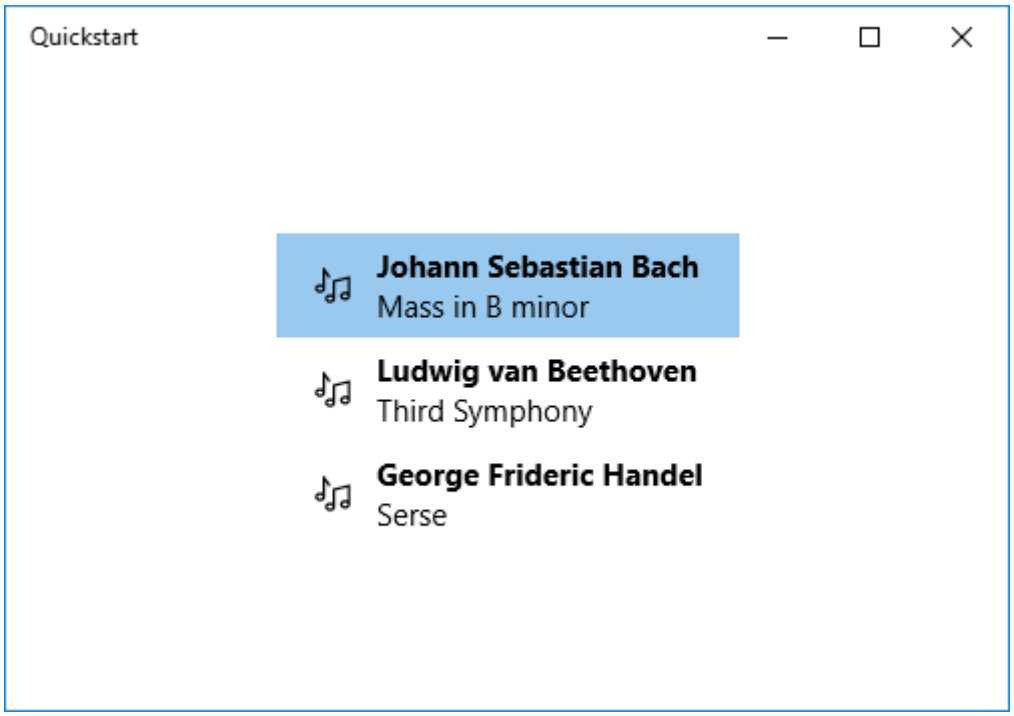
```
<ListView ItemsSource="{x:Bind ViewModel.Recordings}"
HorizontalAlignment="Center" VerticalAlignment="Center">
    <ListView.ItemTemplate>
        <DataTemplate x:DataType="local:Recording">
            <TextBlock Text="{x:Bind OneLineSummary}"/>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```



XAML

```
<ListView ItemsSource="{x:Bind ViewModel.Recordings}"
HorizontalAlignment="Center" VerticalAlignment="Center">
    <ListView.ItemTemplate>
        <DataTemplate x:DataType="local:Recording">
            <StackPanel Orientation="Horizontal" Margin="6">
                <SymbolIcon Symbol="Audio" Margin="0,0,12,0"/>
                <StackPanel>
                    <TextBlock Text="{x:Bind ArtistName}"
FontWeight="Bold"/>
                    <TextBlock Text="{x:Bind CompositionName}"/>
                </StackPanel>
```

```
            </StackPanel>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```



For more information about XAML syntax, see Create a UI with XAML. For more information about control layout, see Define layouts with XAML.

Adding a details view

You can choose to display all the details of `Recording` objects in ListView items. But that takes up a lot of space. Instead, you can show just enough data in the item to identify it and then, when the user makes a selection, you can display all the details of the selected item in a separate piece of UI known as the details view. This arrangement is also known as a master/details view, or a list/details view.

There are two ways to go about this. You can bind the details view to the SelectedItem property of the ListView. Or you can use a CollectionViewSource, in which case you bind both the `ListView` and the details view to the `CollectionViewSource` (doing so takes care of the currently-selected item for you). Both techniques are shown below, and they both give the same results (shown in the illustration).

> ⓘ **Note**

So far in this topic we've only used the {x:Bind} markup extension, but both of the techniques we'll show below require the more flexible (but less performant) {Binding} markup extension.

First, here's the SelectedItem technique. For a C# application, the only change necessary is to the markup.

XAML

```
<Window x:Class="Quickstart.MainWindow" ... >
    <Grid>
        <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
            <ListView x:Name="recordingsListView" ItemsSource="{x:Bind
ViewModel.Recordings}">
                <ListView.ItemTemplate>
                    <DataTemplate x:DataType="local:Recording">
                        <StackPanel Orientation="Horizontal" Margin="6">
                            <SymbolIcon Symbol="Audio" Margin="0,0,12,0"/>
                            <StackPanel>
                                <TextBlock Text="{x:Bind CompositionName}"/>
                            </StackPanel>
                        </StackPanel>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ListView>
            <StackPanel DataContext="{Binding SelectedItem,
ElementName=recordingsListView}"
            Margin="0,24,0,0">
                <TextBlock Text="{Binding ArtistName}"/>
                <TextBlock Text="{Binding CompositionName}"/>
                <TextBlock Text="{Binding ReleaseDateTime}"/>
            </StackPanel>
        </StackPanel>
    </Grid>
</Window>
```

For the CollectionViewSource technique, first add a `CollectionViewSource` as a resource of the top-level `Grid`.

XAML

```
<Grid.Resources>
    <CollectionViewSource x:Name="RecordingsCollection" Source="{x:Bind
ViewModel.Recordings}"/>
</Grid.Resources>
```

ⓘ Note

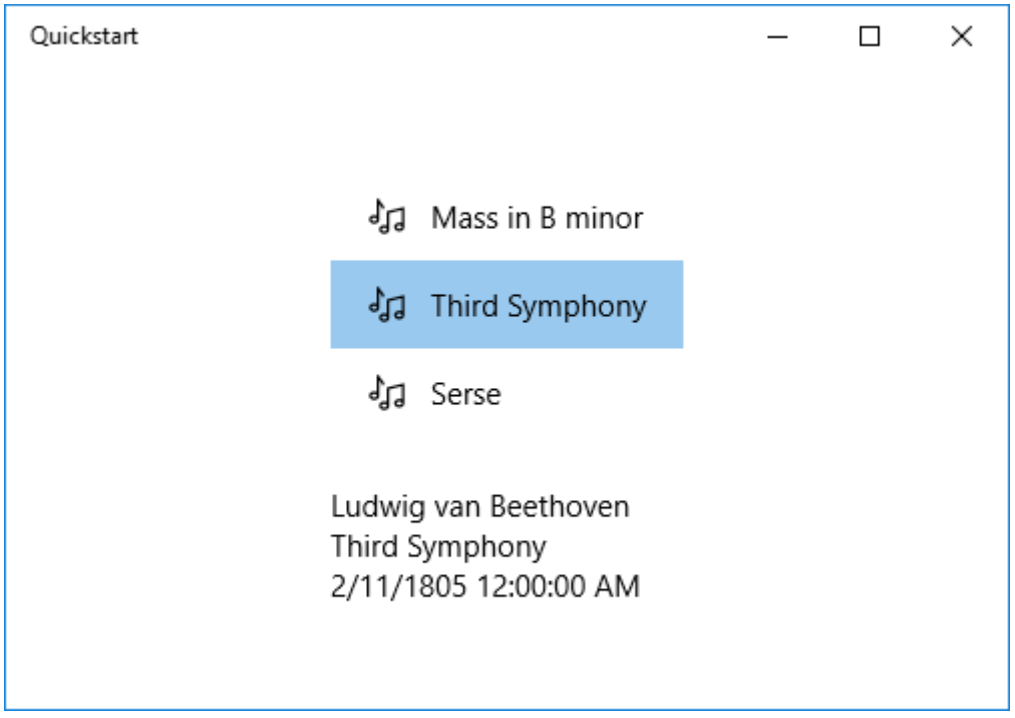
The Window class in WinUI doesn't have a `Resources` property. You can add the `CollectionViewSource` to the top-level `Grid` (or other parent UI element like `StackPanel`) element instead. If you're working within a `Page`, you can add the `CollectionViewSource` to the `Page.Resources`.

And then adjust the bindings on the ListView (which no longer needs to be named) and on the details view to use the CollectionViewSource. Note that by binding the details view directly to the `CollectionViewSource`, you're implying that you want to bind to the current item in bindings where the path cannot be found on the collection itself. There's no need to specify the `CurrentItem` property as the path for the binding, although you can do that if there's any ambiguity.

XAML

```
...
<ListView ItemsSource="{Binding Source={StaticResource
RecordingsCollection}}">
...
<StackPanel DataContext="{Binding Source={StaticResource
RecordingsCollection}}" ...>
...
```

And here's the identical result in each case.



Formatting or converting data values for display

There is an issue with the rendering above. The `ReleaseDateTime` property is not just a date, it's a DateTime. So, it's being displayed with more precision than we need. One solution is to add a string property to the `Recording` class that returns the equivalent of `ReleaseDateTime.ToString("d")`. Naming that property `ReleaseDate` would indicate that it returns a date, and not a date-and-time. Naming it `ReleaseDateAsString` would further indicate that it returns a string.

A more flexible solution is to use something known as a value converter. Here's an example of how to author your own value converter. Add the code below to your **Recording.cs** source code file.

C#

```
public class StringFormatter : Microsoft.UI.Xaml.Data.IValueConverter
{
    // This converts the value object to the string to display.
    // This will work with most simple types.
    public object Convert(object value, Type targetType,
        object parameter, string language)
    {
        // Retrieve the format string and use it to format the value.
        string formatString = parameter as string;
        if (!string.IsNullOrEmpty(formatString))
        {
            return string.Format(formatString, value);
        }

        // If the format string is null or empty, simply
        // call ToString() on the value.
        return value.ToString();
    }

    // No need to implement converting back on a one-way binding
    public object ConvertBack(object value, Type targetType,
        object parameter, string language)
    {
        throw new NotImplementedException();
    }
}
```

Now we can add an instance of `StringFormatter` as a resource and use it in the binding of the `TextBlock` that displays the `ReleaseDateTime` property.

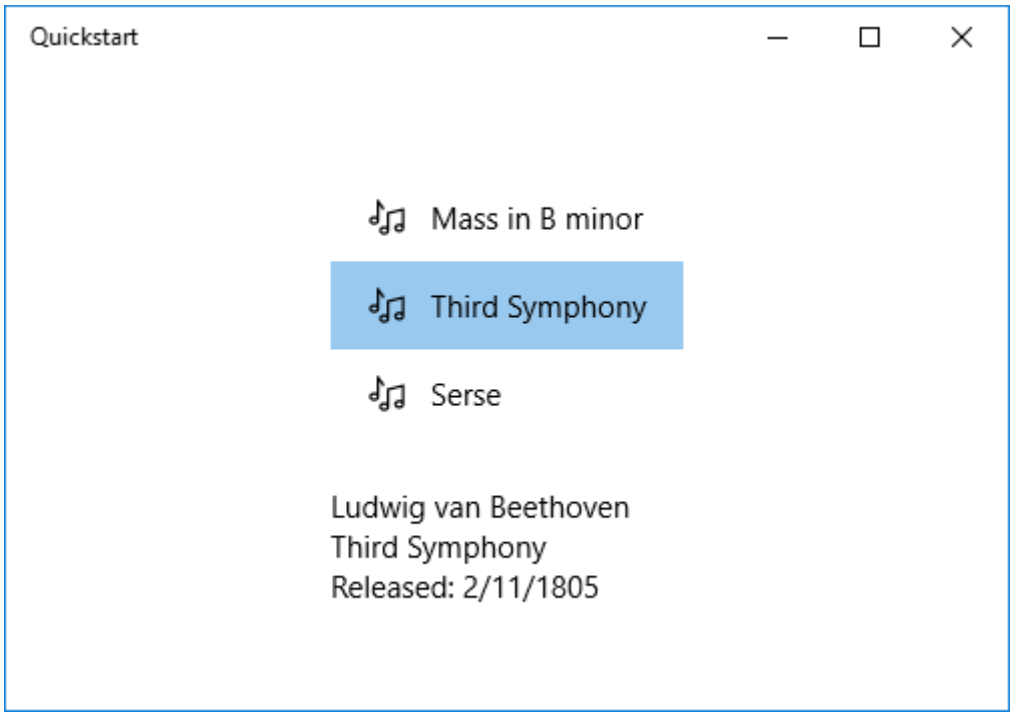
XAML

```
<Grid.Resources>
    ...
    <local:StringFormatter x:Key="StringFormatterValueConverter"/>
</Grid.Resources>
```

```
...
<TextBlock Text="{Binding ReleaseDateTime,
    Converter={StaticResource StringFormatterValueConverter},
    ConverterParameter=Released: \{0:d\}}"/>
...
```

As you can see above, for formatting flexibility we use the markup to pass a format string into the converter by way of the converter parameter. In the code example shown in this topic, the C# value converter makes use of that parameter.

Here's the result.



Related content

- Data binding

Windows data binding in depth

Article • 06/16/2023

In this article, we will describe the Windows App SDK data binding features for the APIs that reside in the **Microsoft.UI.Xaml.Data** namespace.

> **Note**
>
> This topic describes data binding features in detail. For a short, practical introduction, see **Data binding overview**.

Important APIs

- **{x:Bind} markup extension**
- **Binding class**
- **DataContext**
- **INotifyPropertyChanged**

Introduction

Data binding is a way for your app's UI to display data, and optionally to stay in sync with that data. Data binding allows you to separate the concern of data from the concern of UI, and that results in a simpler conceptual model as well as better readability, testability, and maintainability of your app.

You can use data binding to simply display values from a data source when the UI is first shown, but not to respond to changes in those values. This is a mode of binding called *one-time*, and it works well for a value that doesn't change during run-time. Alternatively, you can choose to "observe" the values and to update the UI when they change. This mode is called *one-way*, and it works well for read-only data. Ultimately, you can choose to both observe and update, so that changes that the user makes to values in the UI are automatically pushed back into the data source. This mode is called *two-way*, and it works well for read-write data. Here are some examples.

- You could use the one-time mode to bind an **Image** to the current user's photo.
- You could use the one-way mode to bind a **ListView** to a collection of real-time news articles grouped by newspaper section.
- You could use the two-way mode to bind a **TextBox** to a customer's name in a form.

Independent of mode, there are two kinds of binding, and they're both typically declared in UI markup. You can choose to use either the {x:Bind} markup extension or the {Binding} markup extension. And you can even use a mixture of the two in the same app—even on the same UI element. `{x:Bind}` was new in UWP for Windows 10 and it has better performance. All the details described in this topic apply to both kinds of binding unless we explicitly say otherwise.

UWP Sample apps that demonstrate {x:Bind}

- {x:Bind} sample.
- QuizGame.
- XAML UI Basics sample.

UWP Sample apps that demonstrate {Binding}

- Download the UWP Bookstore1 app.
- Download the Bookstore2 app.

Every binding involves these pieces

- A *binding source*. This is the source of the data for the binding, and it can be an instance of any class that has members whose values you want to display in your UI.
- A *binding target*. This is a **DependencyProperty** of the **FrameworkElement** in your UI that displays the data.
- A *binding object*. This is the piece that transfers data values from the source to the target, and optionally from the target back to the source. The binding object is created at XAML load time from your {x:Bind} or {Binding} markup extension.

In the following sections, we'll take a closer look at the binding source, the binding target, and the binding object. And we'll link the sections together with the example of binding a button's content to a string property named `NextButtonText`, which belongs to a class named `HostViewModel`.

Binding source

Here's a very rudimentary implementation of a class that we could use as a binding source.

```
C#
```

```
public class HostViewModel
{
    public HostViewModel()
    {
        NextButtonText = "Next";
    }

    public string NextButtonText { get; set; }
}
```

That implementation of `HostViewModel`, and its property `NextButtonText`, are only appropriate for one-time binding. But one-way and two-way bindings are extremely common, and in those kinds of binding the UI automatically updates in response to changes in the data values of the binding source. In order for those kinds of binding to work correctly, you need to make your binding source *observable* to the binding object. So in our example, if we want to one-way or two-way bind to the `NextButtonText` property, then any changes that happen at run-time to the value of that property need to be made observable to the binding object.

One way of doing that is to derive the class that represents your binding source from **DependencyObject**, and expose a data value through a **DependencyProperty**. That's how a **FrameworkElement** becomes observable. A `FrameworkElement` is a good binding source right out of the box.

A more lightweight way of making a class observable—and a necessary one for classes that already have a base class—is to implement **System.ComponentModel.INotifyPropertyChanged**. This really just involves implementing a single event named `PropertyChanged`. An example using `HostViewModel` is below.

C#

```
...
using System.ComponentModel;
using System.Runtime.CompilerServices;
...
public class HostViewModel : INotifyPropertyChanged
{
    private string nextButtonText;

    public event PropertyChangedEventHandler PropertyChanged = delegate { };

    public HostViewModel()
    {
        NextButtonText = "Next";
    }
```

```
    public string NextButtonText
    {
        get { return nextButtonText; }
        set
        {
            nextButtonText = value;
            OnPropertyChanged();
        }
    }

    public void OnPropertyChanged([CallerMemberName] string propertyName =
null)
    {
        // Raise the PropertyChanged event, passing the name of the property
whose value has changed.
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
    }
}
```

Now the `NextButtonText` property is observable. When you author a one-way or a two-way binding to that property (we'll show how later), the resulting binding object subscribes to the `PropertyChanged` event. When that event is raised, the binding object's handler receives an argument containing the name of the property that has changed. That's how the binding object knows which property's value to go and read again.

So that you don't have to implement the pattern shown above multiple times, if you're using C# then you can just derive from the `BindableBase` base class that you'll find in the QuizGame sample (in the "Common" folder). Here's an example of how that looks.

C#

```
public class HostViewModel : BindableBase
{
    private string nextButtonText;

    public HostViewModel()
    {
        NextButtonText = "Next";
    }

    public string NextButtonText
    {
        get { return nextButtonText; }
        set { SetProperty(ref nextButtonText, value); }
    }
}
```

Raising the `PropertyChanged` event with an argument of **String.Empty** or `null` indicates that all non-indexer properties on the object should be re-read. You can raise the event to indicate that indexer properties on the object have changed by using an argument of "Item[*indexer*]" for specific indexers (where *indexer* is the index value), or a value of "Item[]" for all indexers.

A binding source can be treated either as a single object whose properties contain data, or as a collection of objects. In C# code, you can one-time bind to an object that implements **List<T>** to display a collection that doesn't change at run-time. For an observable collection (observing when items are added to and removed from the collection), one-way bind to **ObservableCollection<T>** instead. To bind to your own collection classes, use the guidance in the following table.

Scenario	C# (CLR)	C++/WinRT
Bind to an object.	Can be any object.	Can be any object.
Get property change notifications from a bound object.	Object must implement **INotifyPropertyChanged**.	Object must implement **INotifyPropertyChanged**.
Bind to a collection.	**List<T>**	**IVector** of **IInspectable**, or **IBindableObservableVector**. See XAML items controls; bind to a C++/WinRT collection and Collections with C++/WinRT.
Get collection change notifications from a bound collection.	**ObservableCollection<T>**	**IObservableVector** of **IInspectable**. For example, **winrt::single_threaded_observable_vector<T>**.
Implement a collection that supports binding.	Extend **List<T>** or implement **IList**, **IList<Object>**, **IEnumerable**, or **IEnumerable**<Object>. Binding to generic `IList<T>` and `IEnumerable<T>` is not supported.	Implement **IVector** of **IInspectable**. See XAML items controls; bind to a C++/WinRT collection and Collections with C++/WinRT.
Implement a collection	Extend **ObservableCollection<T>** or	Implement **IObservableVector** of **IInspectable**, or **IBindableObservableVector**.

Scenario	C# (CLR)	C++/WinRT
that supports collection change notifications.	implement (non-generic) IList and INotifyCollectionChanged.	
Implement a collection that supports incremental loading.	Extend ObservableCollection<T> or implement (non-generic) IList and INotifyCollectionChanged. Additionally, implement ISupportIncrementalLoading.	Implement IObservableVector of IInspectable, or IBindableObservableVector. Additionally, implement ISupportIncrementalLoading

You can bind list controls to arbitrarily large data sources, and still achieve high performance, by using incremental loading. For example, you can bind list controls to Bing image query results without having to load all the results at once. Instead, you load only some results immediately, and load additional results as needed. To support incremental loading, you must implement **ISupportIncrementalLoading** on a data source that supports collection change notifications. When the data binding engine requests more data, your data source must make the appropriate requests, integrate the results, and then send the appropriate notifications in order to update the UI.

Binding target

In the two examples below, the `Button.Content` property is the binding target, and its value is set to a markup extension that declares the binding object. First {x:Bind} is shown, and then {Binding}. Declaring bindings in markup is the common case (it's convenient, readable, and toolable). But you can avoid markup and imperatively (programmatically) create an instance of the **Binding** class instead if you need to.

XAML

```
<Button Content="{x:Bind ...}" ... />
```

XAML

```
<Button Content="{Binding ...}" ... />
```

Binding object declared using {x:Bind}

There's one step we need to do before we author our {x:Bind} markup. We need to expose our binding source class from the class that represents our page of markup. We do that by adding a property (of type `HostViewModel` in this case) to our `MainWindow` window class.

C#

```
namespace DataBindingInDepth
{
    public sealed partial class MainWindow : Window
    {
        public MainWindow()
        {
            this.InitializeComponent();
            ViewModel = new HostViewModel();
        }

        public HostViewModel ViewModel { get; set; }
    }
}
```

That done, we can now take a closer look at the markup that declares the binding object. The example below uses the same `Button.Content` binding target we used in the "Binding target" section earlier, and shows it being bound to the `HostViewModel.NextButtonText` property.

XAML

```
<!-- MainWindow.xaml -->
<Window x:Class="DataBindingInDepth.MainWindow" ... >
    <Button Content="{x:Bind Path=ViewModel.NextButtonText, Mode=OneWay}"
... />
</Window>
```

Notice the value that we specify for `Path`. This value is interpreted in the context of the window itself, and in this case the path begins by referencing the `ViewModel` property that we just added to the `MainWindow` page. That property returns a `HostViewModel` instance, and so we can *dot* into that object to access the `HostViewModel.NextButtonText` property. And we specify `Mode`, to override the {x:Bind} default of one-time.

The **Path** property supports a variety of syntax options for binding to nested properties, attached properties, and integer and string indexers. For more info, see Property-path syntax. Binding to string indexers gives you the effect of binding to dynamic properties without having to implement **ICustomPropertyProvider**. For other settings, see {x:Bind} markup extension.

To illustrate that the `HostViewModel.NextButtonText` property is indeed observable, add a `Click` event handler to the button, and update the value of `HostViewModel.NextButtonText`. Build, run, and click the button to see the value of the button's `Content` update.

C#

```
// MainWindow.xaml.cs
private void Button_Click(object sender, RoutedEventArgs e)
{
    ViewModel.NextButtonText = "Updated Next button text";
}
```

> ⓘ Note
>
> Changes to **TextBox.Text** are sent to a two-way bound source when the **TextBox** loses focus, and not after every user keystroke.

DataTemplate and x:DataType

Inside a **DataTemplate** (whether used as an item template, a content template, or a header template), the value of `Path` is not interpreted in the context of the window, but in the context of the data object being templated. When using `{x:Bind}` in a data template, so that its bindings can be validated (and efficient code generated for them) at compile-time, the `DataTemplate` needs to declare the type of its data object using `x:DataType`. The example given below could be used as the `ItemTemplate` of an items control bound to a collection of `SampleDataGroup` objects.

XAML

```
<DataTemplate x:Key="SimpleItemTemplate" x:DataType="data:SampleDataGroup">
    <StackPanel Orientation="Vertical" Height="50">
      <TextBlock Text="{x:Bind Title}"/>
      <TextBlock Text="{x:Bind Description}"/>
    </StackPanel>
  </DataTemplate>
```

Weakly-typed objects in your Path

Consider for example that you have a type named `SampleDataGroup`, which implements a string property named `Title`. And you have a property

`MainWindow.SampleDataGroupAsObject`, which is of type `object`, but which actually returns an instance of `SampleDataGroup`. The binding `<TextBlock Text="{x:Bind SampleDataGroupAsObject.Title}"/>` will result in a compile error because the `Title` property is not found on the type `object`. The remedy for this is to add a cast to your `Path` syntax like this: `<TextBlock Text="{x:Bind ((data:SampleDataGroup)SampleDataGroupAsObject).Title}"/>`. Here's another example where `Element` is declared as `object` but is actually a `TextBlock`: `<TextBlock Text="{x:Bind Element.Text}"/>`. And a cast remedies the issue: `<TextBlock Text="{x:Bind ((TextBlock)Element).Text}"/>`.

If your data loads asynchronously

Code to support `{x:Bind}` is generated at compile-time in the partial classes for your windows. These files can be found in your `obj` folder, with names like (for C#) `<view name>.g.cs`. The generated code includes a handler for your window's **Loading** event, and that handler calls the `Initialize` method on a generated class that represent's your window's bindings. `Initialize` in turn calls `Update` to begin moving data between the binding source and the target. `Loading` is raised just before the first measure pass of the window or user control. So if your data is loaded asynchronously it may not be ready by the time `Initialize` is called. So, after you've loaded data, you can force one-time bindings to be initialized by calling `this.Bindings.Update();`. If you only need one-time bindings for asynchronously-loaded data then it's much cheaper to initialize them this way than it is to have one-way bindings and to listen for changes. If your data does not undergo fine-grained changes, and if it's likely to be updated as part of a specific action, then you can make your bindings one-time, and force a manual update at any time with a call to `Update`.

> ⓘ **Note**
>
> `{x:Bind}` is not suited to late-bound scenarios, such as navigating the dictionary structure of a JSON object, nor duck typing. "Duck typing" is a weak form of typing based on lexical matches on property names (as in, "if it walks, swims, and quacks like a duck, then it's a duck"). With duck typing, a binding to the `Age` property would be equally satisfied with a `Person` or a `Wine` object (assuming that those types each had an `Age` property). For these scenarios, use the `{Binding}` markup extension.

Binding object declared using {Binding}

{Binding} assumes, by default, that you're binding to the **DataContext** of your markup window. So we'll set the `DataContext` of our window to be an instance of our binding source class (of type `HostViewModel` in this case). The example below shows the markup that declares the binding object. We use the same `Button.Content` binding target we used in the "Binding target" section earlier, and we bind to the `HostViewModel.NextButtonText` property.

XAML

```
<Window xmlns:viewmodel="using:DataBindingInDepth" ... >
    <Window.DataContext>
        <viewmodel:HostViewModel x:Name="viewModelInDataContext"/>
    </Window.DataContext>
    ...
    <Button Content="{Binding Path=NextButtonText}" ... />
</Window>
```

C#

```
// MainWindow.xaml.cs
private void Button_Click(object sender, RoutedEventArgs e)
{
    viewModelInDataContext.NextButtonText = "Updated Next button text";
}
```

Notice the value that we specify for `Path`. This value is interpreted in the context of the window's **DataContext**, which in this example is set to an instance of `HostViewModel`. The path references the `HostViewModel.NextButtonText` property. We can omit `Mode`, because the {Binding} default of one-way works here.

The default value of **DataContext** for a UI element is the inherited value of its parent. You can of course override that default by setting `DataContext` explicitly, which is in turn inherited by children by default. Setting `DataContext` explicitly on an element is useful when you want to have multiple bindings that use the same source.

A binding object has a `Source` property, which defaults to the **DataContext** of the UI element on which the binding is declared. You can override this default by setting `Source`, `RelativeSource`, or `ElementName` explicitly on the binding (see {Binding} for details).

Inside a **DataTemplate**, the **DataContext** is automatically set to the data object being templated. The example given below could be used as the `ItemTemplate` of an items control bound to a collection of any type that has string properties named `Title` and `Description`.

XAML

```
<DataTemplate x:Key="SimpleItemTemplate">
    <StackPanel Orientation="Vertical" Height="50">
      <TextBlock Text="{Binding Title}"/>
      <TextBlock Text="{Binding Description"/>
    </StackPanel>
  </DataTemplate>
```

> **Note**
>
> By default, changes to **TextBox.Text** are sent to a two-way bound source when the **TextBox** loses focus. To cause changes to be sent after every user keystroke, set `UpdateSourceTrigger` to `PropertyChanged` on the binding in markup. You can also completely take control of when changes are sent to the source by setting `UpdateSourceTrigger` to `Explicit`. You then handle events on the text box (typically **TextBox.TextChanged**), call **GetBindingExpression** on the target to get a **BindingExpression** object, and finally call **BindingExpression.UpdateSource** to programmatically update the data source.

The **Path** property supports a variety of syntax options for binding to nested properties, attached properties, and integer and string indexers. For more info, see Property-path syntax. Binding to string indexers gives you the effect of binding to dynamic properties without having to implement **ICustomPropertyProvider**. The **ElementName** property is useful for element-to-element binding. The **RelativeSource** property has several uses, one of which is as a more powerful alternative to template binding inside a **ControlTemplate**. For other settings, see {Binding} markup extension and the **Binding** class.

What if the source and the target are not the same type?

If you want to control the visibility of a UI element based on the value of a boolean property, or if you want to render a UI element with a color that's a function of a numeric value's range or trend, or if you want to display a date and/or time value in a UI element property that expects a string, then you'll need to convert values from one type to another. There will be cases where the right solution is to expose another property of the right type from your binding source class, and keep the conversion logic encapsulated and testable there. But that isn't flexible nor scalable when you have large

numbers, or large combinations, of source and target properties. In that case you have a couple of options:

- If using `{x:Bind}` then you can bind directly to a function to do that conversion
- Or you can specify a value converter which is an object designed to perform the conversion

Value Converters

Here's a value converter, suitable for a one-time or a one-way binding, that converts a **DateTime** value to a `string` value containing the month. The class implements **IValueConverter**.

C#

```
public class DateToStringConverter : IValueConverter
{
    // Define the Convert method to convert a DateTime value to
    // a month string.
    public object Convert(object value, Type targetType,
        object parameter, string language)
    {
        // value is the data from the source object.
        DateTime thisDate = (DateTime)value;
        int monthNum = thisDate.Month;
        string month;
        switch (monthNum)
        {
            case 1:
                month = "January";
                break;
            case 2:
                month = "February";
                break;
            default:
                month = "Month not found";
                break;
        }
        // Return the value to pass to the target.
        return month;
    }

    // ConvertBack is not implemented for a OneWay binding.
    public object ConvertBack(object value, Type targetType,
        object parameter, string language)
    {
        throw new NotImplementedException();
    }
}
```

And here's how you consume that value converter in your binding object markup.

XAML

```
<UserControl.Resources>
  <local:DateToStringConverter x:Key="Converter1"/>
</UserControl.Resources>
...
<TextBlock Grid.Column="0"
  Text="{x:Bind ViewModel.Month, Converter={StaticResource Converter1}}"/>
<TextBlock Grid.Column="0"
  Text="{Binding Month, Converter={StaticResource Converter1}}"/>
```

The binding engine calls the **Convert** and **ConvertBack** methods if the **Converter** parameter is defined for the binding. When data is passed from the source, the binding engine calls `Convert` and passes the returned data to the target. When data is passed from the target (for a two-way binding), the binding engine calls `ConvertBack` and passes the returned data to the source.

The converter also has optional parameters: **ConverterLanguage**, which allows specifying the language to be used in the conversion, and **ConverterParameter**, which allows passing a parameter for the conversion logic. For an example that uses a converter parameter, see **IValueConverter**.

> ⓘ **Note**
>
> If there is an error in the conversion, do not throw an exception. Instead, return **DependencyProperty.UnsetValue**, which will stop the data transfer.

To display a default value to use whenever the binding source cannot be resolved, set the `FallbackValue` property on the binding object in markup. This is useful to handle conversion and formatting errors. It is also useful to bind to source properties that might not exist on all objects in a bound collection of heterogeneous types.

If you bind a text control to a value that is not a string, the data binding engine will convert the value to a string. If the value is a reference type, the data binding engine will retrieve the string value by calling **ICustomPropertyProvider.GetStringRepresentation** or **IStringable.ToString** if available, and will otherwise call **Object.ToString**. Note, however, that the binding engine will ignore any `ToString` implementation that hides the base-class implementation. Subclass implementations should override the base class `ToString` method instead. Similarly, in native languages, all managed objects appear to implement **ICustomPropertyProvider** and **IStringable**. However, all calls to `GetStringRepresentation` and `IStringable.ToString` are routed to `Object.ToString` or

an override of that method, and never to a new `ToString` implementation that hides the base-class implementation.

> **Note**
>
> The **Windows Community Toolkit** provides a **BoolToVisibilityConverter**. The converter maps `true` to the `Visible` enumeration value and `false` to `Collapsed` so you can bind a `Visibility` property to a boolean without creating a converter. To use the converter, your project must add the **CommunityToolkit.WinUI.Converters** NuGet package.

Function binding in {x:Bind}

`{x:Bind}` enables the final step in a binding path to be a function. This can be used to perform conversions, and to perform bindings that depend on more than one property. See **Functions in x:Bind**

Element-to-element binding

You can bind the property of one XAML element to the property of another XAML element. Here's an example of how that looks in markup.

XAML

```
<TextBox x:Name="myTextBox" />
<TextBlock Text="{x:Bind myTextBox.Text, Mode=OneWay}" />
```

Resource dictionaries with {x:Bind}

The {x:Bind} markup extension depends on code generation, so it needs a code-behind file containing a constructor that calls `InitializeComponent` (to initialize the generated code). You re-use the resource dictionary by instantiating its type (so that `InitializeComponent` is called) instead of referencing its filename. Here's an example of what to do if you have an existing resource dictionary and you want to use `{x:Bind}` in it.

XAML

```
<!-- TemplatesResourceDictionary.xaml -->
<ResourceDictionary
```

```
    x:Class="ExampleNamespace.TemplatesResourceDictionary"
    .....
    xmlns:examplenamespace="using:ExampleNamespace">

    <DataTemplate x:Key="EmployeeTemplate"
x:DataType="examplenamespace:IEmployee">
        <Grid>
            <TextBlock Text="{x:Bind Name}"/>
        </Grid>
    </DataTemplate>
</ResourceDictionary>
```

C#

```
// TemplatesResourceDictionary.xaml.cs
using Microsoft.UI.Xaml.Data;

namespace ExampleNamespace
{
    public partial class TemplatesResourceDictionary
    {
        public TemplatesResourceDictionary()
        {
            InitializeComponent();
        }
    }
}
```

XAML

```
<!-- MainWindow.xaml -->
<Window x:Class="ExampleNamespace.MainWindow"
    ....
    xmlns:examplenamespace="using:ExampleNamespace">

    <Window.Resources>
        <ResourceDictionary>
            ....
            <ResourceDictionary.MergedDictionaries>
                <examplenamespace:TemplatesResourceDictionary/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Window.Resources>
</Window>
```

Event binding and ICommand

{x:Bind} supports a feature called event binding. With this feature, you can specify the handler for an event using a binding, which is an additional option on top of handling

events with a method on the code-behind file. Let's say you have a `ListViewDoubleTapped` event handler on your `MainWindow` class.

C#

```
public sealed partial class MainWindow : Window
{
    ...
    public void ListViewDoubleTapped()
    {
        // Handle double-tapped logic
    }
}
```

You can then bind a ListView's **DoubleTapped** event to a method on the **MainWindow** like this.

XAML

```
<ListView DoubleTapped="{x:Bind ListViewDoubleTapped}" />
```

Overloaded methods cannot be used to handle an event with this technique. Also, if the method that handles the event has parameters then they must all be assignable from the types of all of the event's parameters, respectively. In this case, `ListViewDoubleTapped` is not overloaded and it has no parameters (but it would still be valid even if it took two `object` parameters).

The event binding technique is similar to implementing and consuming commands (a command is a property that returns an object that implements the **ICommand** interface). Both {x:Bind} and {Binding} work with commands. So that you don't have to implement the command pattern multiple times, you can use the `DelegateCommand` helper class that you'll find in the QuizGame UWP sample (in the "Common" folder).

Binding to a collection of folders or files

You can use the APIs in the **Windows.Storage** namespace to retrieve folder and file data in your packaged Windows App SDK apps. However, the various `GetFilesAsync`, `GetFoldersAsync`, and `GetItemsAsync` methods do not return values that are suitable for binding to list controls. Instead, you must bind to the return values of the **GetVirtualizedFilesVector**, **GetVirtualizedFoldersVector**, and **GetVirtualizedItemsVector** methods of the **FileInformationFactory** class. The following code example from the StorageDataSource and GetVirtualizedFilesVector UWP sample shows the typical

usage pattern. Remember to declare the **picturesLibrary** capability in your app package manifest, and confirm that there are pictures in your Pictures library folder.

C#

```
protected override void OnNavigatedTo(NavigationEventArgs e)
{
    var library = Windows.Storage.KnownFolders.PicturesLibrary;
    var queryOptions = new Windows.Storage.Search.QueryOptions();
    queryOptions.FolderDepth = Windows.Storage.Search.FolderDepth.Deep;
    queryOptions.IndexerOption =
Windows.Storage.Search.IndexerOption.UseIndexerWhenAvailable;

    var fileQuery = library.CreateFileQueryWithOptions(queryOptions);

    var fif = new Windows.Storage.BulkAccess.FileInformationFactory(
        fileQuery,
        Windows.Storage.FileProperties.ThumbnailMode.PicturesView,
        190,
        Windows.Storage.FileProperties.ThumbnailOptions.UseCurrentScale,
        false
        );

    var dataSource = fif.GetVirtualizedFilesVector();
    this.PicturesListView.ItemsSource = dataSource;
}
```

You will typically use this approach to create a read-only view of file and folder info. You can create two-way bindings to the file and folder properties, for example to let users rate a song in a music view. However, any changes are not persisted until you call the appropriate `SavePropertiesAsync` method (for example, MusicProperties.SavePropertiesAsync). You should commit changes when the item loses focus because this triggers a selection reset.

Note that two-way binding using this technique works only with indexed locations, such as **Music**. You can determine whether a location is indexed by calling the FolderInformation.GetIndexedStateAsync method.

Note also that a virtualized vector can return `null` for some items before it populates their value. For example, you should check for `null` before you use the SelectedItem value of a list control bound to a virtualized vector, or use SelectedIndex instead.

Binding to data grouped by a key

If you take a flat collection of items (books, for example, represented by a `BookSku` class) and you group the items by using a common property as a key (the `BookSku.AuthorName`

property, for example) then the result is called grouped data. When you group data, it is no longer a flat collection. Grouped data is a collection of group objects, where each group object has:

- a key, and
- a collection of items whose property matches that key.

To take the books example again, the result of grouping the books by author name results in a collection of author name groups where each group has:

- a key, which is an author name, and
- a collection of the `BookSku` objects whose `AuthorName` property matches the group's key.

In general, to display a collection, you bind the **ItemsSource** of an items control (such as **ListView** or **GridView**) directly to a property that returns a collection. If that's a flat collection of items then you don't need to do anything special. But if it's a collection of group objects (as it is when binding to grouped data) then you need the services of an intermediary object called a **CollectionViewSource** which sits between the items control and the binding source. You bind the `CollectionViewSource` to the property that returns grouped data, and you bind the items control to the `CollectionViewSource`. An extra value-add of a `CollectionViewSource` is that it keeps track of the current item, so you can keep more than one items control in sync by binding them all to the same `CollectionViewSource`. You can also access the current item programmatically through the **ICollectionView.CurrentItem** property of the object returned by the **CollectionViewSource.View** property.

To activate the grouping facility of a **CollectionViewSource**, set **IsSourceGrouped** to `true`. Whether you also need to set the **ItemsPath** property depends on exactly how you author your group objects. There are two ways to author a group object: the "is-a-group" pattern, and the "has-a-group" pattern. In the "is-a-group" pattern, the group object derives from a collection type (for example, `List<T>`), so the group object actually is itself the group of items. With this pattern you do not need to set `ItemsPath`. In the "has-a-group" pattern, the group object has one or more properties of a collection type (such as `List<T>`), so the group "has a" group of items in the form of a property (or several groups of items in the form of several properties). With this pattern you need to set `ItemsPath` to the name of the property that contains the group of items.

The example below illustrates the "has-a-group" pattern. The window class has a property named **DataContext**, which returns an instance of our view model. The **CollectionViewSource** binds to the `Authors` property of the view model (`Authors` is the collection of group objects) and also specifies that it's the `Author.BookSkus` property

that contains the grouped items. Finally, the **GridView** is bound to the `CollectionViewSource`, and has its group style defined so that it can render the items in groups.

XAML

```
<Window.Resources>
    <CollectionViewSource
    x:Name="AuthorHasACollectionOfBookSku"
    Source="{x:Bind ViewModel.Authors}"
    IsSourceGrouped="true"
    ItemsPath="BookSkus"/>
</Window.Resources>
...
<GridView
ItemsSource="{x:Bind AuthorHasACollectionOfBookSku}" ...>
    <GridView.GroupStyle>
        <GroupStyle
            HeaderTemplate="{StaticResource AuthorGroupHeaderTemplateWide}"
... />
    </GridView.GroupStyle>
</GridView>
```

You can implement the "is-a-group" pattern in one of two ways. One way is to author your own group class. Derive the class from `List<T>` (where *T* is the type of the items). For example, `public class Author : List<BookSku>`. The second way is to use a LINQ expression to dynamically create group objects (and a group class) from like property values of the **BookSku** items. This approach—maintaining only a flat list of items and grouping them together on the fly—is typical of an app that accesses data from a cloud service. You get the flexibility to group books by author or by genre (for example) without needing special group classes such as **Author** and **Genre**.

The example below illustrates the "is-a-group" pattern using LINQ. This time we group books by genre, displayed with the genre name in the group headers. This is indicated by the "Key" property path in reference to the group **Key** value.

C#

```
using System.Linq;
...
private IOrderedEnumerable<IGrouping<string, BookSku>> genres;

public IOrderedEnumerable<IGrouping<string, BookSku>> Genres
{
    get
    {
        if (genres == null)
        {
```

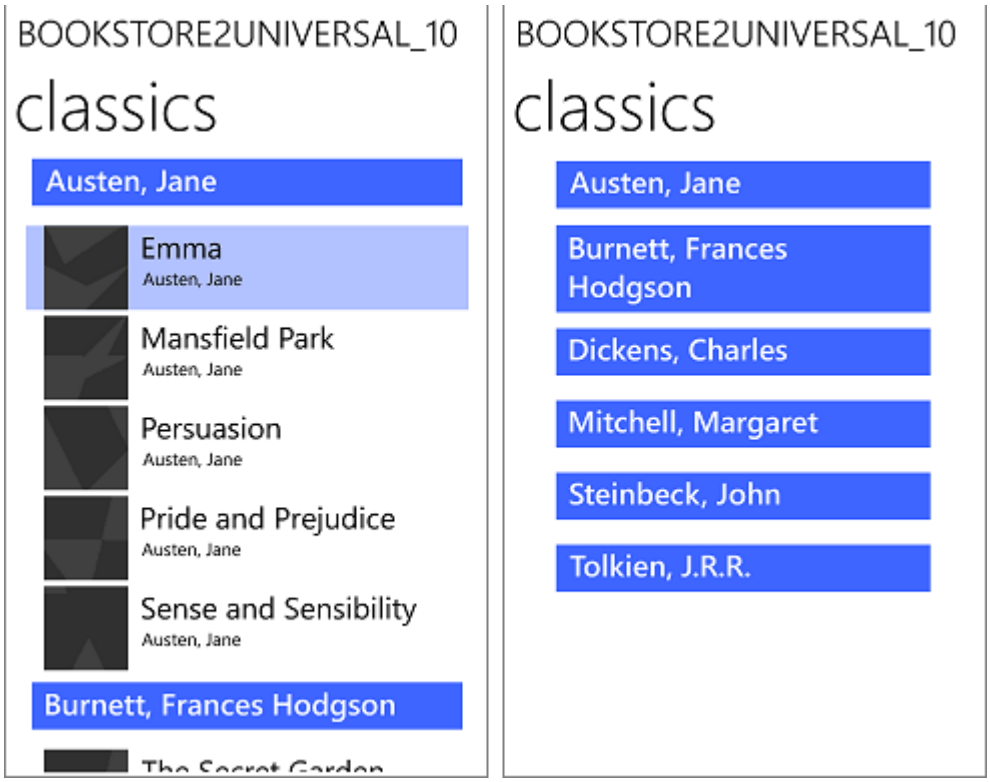
```
            genres = from book in bookSkus
                     group book by book.genre into grp
                     orderby grp.Key
                     select grp;
        }
        return genres;
    }
}
```

Remember that when using {x:Bind} with data templates we need to indicate the type being bound to by setting an `x:DataType` value. If the type is generic then we can't express that in markup so we need to use {Binding} instead in the group style header template.

XAML

```
    <Grid.Resources>
        <CollectionViewSource x:Name="GenreIsACollectionOfBookSku"
        Source="{x:Bind Genres}"
        IsSourceGrouped="true"/>
    </Grid.Resources>
    <GridView ItemsSource="{x:Bind GenreIsACollectionOfBookSku}">
        <GridView.ItemTemplate x:DataType="local:BookTemplate">
            <DataTemplate>
                <TextBlock Text="{x:Bind Title}"/>
            </DataTemplate>
        </GridView.ItemTemplate>
        <GridView.GroupStyle>
            <GroupStyle>
                <GroupStyle.HeaderTemplate>
                    <DataTemplate>
                        <TextBlock Text="{Binding Key}"/>
                    </DataTemplate>
                </GroupStyle.HeaderTemplate>
            </GroupStyle>
        </GridView.GroupStyle>
    </GridView>
```

A **SemanticZoom** control is a great way for your users to view and navigate grouped data. The Bookstore2 UWP sample app illustrates how to use the `SemanticZoom`. In that app, you can view a list of books grouped by author (the zoomed-in view) or you can zoom out to see a jump list of authors (the zoomed-out view). The jump list affords much quicker navigation than scrolling through the list of books. The zoomed-in and zoomed-out views are actually `ListView` or `GridView` controls bound to the same `CollectionViewSource`.



When you bind to hierarchical data—such as subcategories within categories—you can choose to display the hierarchical levels in your UI with a series of items controls. A selection in one items control determines the contents of subsequent items controls. You can keep the lists synchronized by binding each list to its own **CollectionViewSource** and binding the `CollectionViewSource` instances together in a chain. This is called a master/details (or list/details) view. For more info, see How to bind to hierarchical data and create a master/details view.

Diagnosing and debugging data binding problems

Your binding markup contains the names of properties (and, for C#, sometimes fields and methods). So when you rename a property, you'll also need to change any binding that references it. Forgetting to do that leads to a typical example of a data binding bug, and your app either won't compile or won't run correctly.

The binding objects created by {x:Bind} and {Binding} are largely functionally equivalent. But `{x:Bind}` has type information for the binding source, and it generates source code at compile-time. With `{x:Bind}` you get the same kind of problem detection that you get with the rest of your code. That includes compile-time validation of your binding expressions, and debugging by setting breakpoints in the source code generated as the partial class for your page. These classes can be found in the files in your `obj` folder, with names like (for C#) `<view name>.g.cs`). If you have a problem with a binding then turn on **Break On Unhandled Exceptions** in the Microsoft Visual Studio debugger. The

debugger will break execution at that point, and you can then debug what has gone wrong. The code generated by `{x:Bind}` follows the same pattern for each part of the graph of binding source nodes, and you can use the info in the **Call Stack** window to help determine the sequence of calls that led up to the problem.

{Binding} does not have type information for the binding source. But when you run your app with the debugger attached, any binding errors appear in the **Output** and **XAML Binding Failures** windows in Visual Studio. For more information on debugging binding errors in Visual Studio, see XAML data binding diagnostics.

Creating bindings in code

> **Note**
>
> This section only applies to **{Binding}**, because you can't create **{x:Bind}** bindings in code. However, some of the same benefits of `{x:Bind}` can be achieved with **DependencyObject.RegisterPropertyChangedCallback**, which enables you to register for change notifications on any dependency property.

You can also connect UI elements to data using procedural code instead of XAML. To do this, create a new **Binding** object, set the appropriate properties, then call **FrameworkElement.SetBinding** or **BindingOperations.SetBinding**. Creating bindings programmatically is useful when you want to choose the binding property values at run-time or share a single binding among multiple controls. Note, however, that you cannot change the binding property values after you call `SetBinding`.

The following example shows how to implement a binding in code.

XAML

```
<TextBox x:Name="MyTextBox" Text="Text"/>
```

C#

```
// Create an instance of the MyColors class
// that implements INotifyPropertyChanged.
var textcolor = new MyColors();

// Brush1 is set to be a SolidColorBrush with the value Red.
textcolor.Brush1 = new SolidColorBrush(Colors.Red);

// Set the DataContext of the TextBox MyTextBox.
MyTextBox.DataContext = textcolor;
```

```
// Create the binding and associate it with the text box.
var binding = new Binding { Path = new PropertyPath("Brush1") };
MyTextBox.SetBinding(TextBox.ForegroundProperty, binding);
```

{x:Bind} and {Binding} feature comparison

Feature	{x:Bind} vs. {Binding}	Notes
Path is the default property	`{x:Bind a.b.c}` - `{Binding a.b.c}`	
Path property	`{x:Bind Path=a.b.c}` - `{Binding Path=a.b.c}`	In `x:Bind`, `Path` is rooted at the Window by default, not the DataContext.
Indexer	`{x:Bind Groups[2].Title}` - `{Binding Groups[2].Title}`	Binds to the specified item in the collection. Only integer-based indexes are supported.
Attached properties	`{x:Bind Button22.(Grid.Row)}` - `{Binding Button22.(Grid.Row)}`	Attached properties are specified using parentheses. If the property is not declared in a XAML namespace, then prefix it with an xml namespace, which should be mapped to a code namespace at the head of the document.
Casting	`{x:Bind groups[0].(data:SampleDataGroup.Title)}` - Not needed for `{Binding}`.	Casts are specified using parentheses. If the property is not declared in a XAML namespace, then prefix it with an xml namespace, which should be mapped to a code namespace at the head of the document.
Converter	`{x:Bind IsShown, Converter={StaticResource BoolToVisibility}}` - `{Binding IsShown, Converter={StaticResource BoolToVisibility}}`	Converters must be declared at the root of the Window/Control/ResourceDictionary, or in **App.xaml**.
ConverterParameter, ConverterLanguage	`{x:Bind IsShown, Converter={StaticResource BoolToVisibility}, ConverterParameter=One, ConverterLanguage=fr-fr}` -	Converters must be declared at the root of the Window/Control/ResourceDictionary, or in **App.xaml**.

Feature	{x:Bind} vs. {Binding}	Notes
	`{Binding IsShown, Converter={StaticResource BoolToVisibility}, ConverterParameter=One, ConverterLanguage=fr-fr}`	
TargetNullValue	`{x:Bind Name, TargetNullValue=0}` - `{Binding Name, TargetNullValue=0}`	Used when the leaf of the binding expression is null. Use single quotes for a string value.
FallbackValue	`{x:Bind Name, FallbackValue='empty'}` - `{Binding Name, FallbackValue='empty'}`	Used when any part of the path for the binding (except for the leaf) is null.
ElementName	`{x:Bind slider1.Value}` - `{Binding Value, ElementName=slider1}`	With `{x:Bind}` you're binding to a field; `Path` is rooted at the Window by default, so any named element can be accessed via its field.
RelativeSource: Self	`<Rectangle x:Name="rect1" Width="200" Height="{x:Bind rect1.Width}" ... />` - `<Rectangle Width="200" Height="{Binding Width, RelativeSource={RelativeSource Self}}" ... />`	With `{x:Bind}`, name the element and use its name in `Path`.
RelativeSource: TemplatedParent	Not needed for `{x:Bind}` - `{Binding <path>, RelativeSource={RelativeSource TemplatedParent}}`	With `{x:Bind}`, `TargetType` on `ControlTemplate` indicates binding to template parent. For `{Binding}`, regular template binding can be used in control templates for most uses. But use `TemplatedParent` where you need to use a converter, or a two-way binding.
Source	Not needed for `{x:Bind}` - `<ListView ItemsSource="{Binding Orders, Source={StaticResource MyData}}"/>`	For `{x:Bind}` you can directly use the named element, use a property or a static path.
Mode	`{x:Bind Name, Mode=OneWay}` - `{Binding Name, Mode=TwoWay}`	`Mode` can be `OneTime`, `OneWay`, or `TwoWay`. `{x:Bind}` defaults to `OneTime`; `{Binding}` defaults to `OneWay`.

Feature	{x:Bind} vs. {Binding}	Notes
UpdateSourceTrigger	`{x:Bind Name, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}` - `{Binding UpdateSourceTrigger=PropertyChanged}`	`UpdateSourceTrigger` can be `Default`, `LostFocus`, or `PropertyChanged`. `{x:Bind}` does not support `UpdateSourceTrigger=Explicit`. `{x:Bind}` uses `PropertyChanged` behavior for all cases except `TextBox.Text`, where it uses `LostFocus` behavior.

See also

- Data binding overview
- Data binding and MVVM

Functions in x:Bind with Windows App SDK

Article • 06/29/2023

> **Note**
>
> For general info about using data binding in your app with `{x:Bind}` (and for an all-up comparison between `{x:Bind}` and `{Binding}`), see **Data binding in depth** and **{x:Bind} Markup Extension**.

In Windows App SDK apps, **{x:Bind}** supports using a function as the leaf step of the binding path. This enables:

- A simpler way to achieve value conversion
- A way for bindings to depend on more than one parameter

In the following example, the background and foreground of the item are bound to functions to do conversion based on the color parameter

XAML

```
<DataTemplate x:DataType="local:ColorEntry">
    <Grid Background="{x:Bind local:ColorEntry.Brushify(Color),
Mode=OneWay}" Width="240">
        <TextBlock Text="{x:Bind ColorName}" Foreground="{x:Bind
TextColor(Color)}" Margin="10,5" />
    </Grid>
</DataTemplate>
```

C#

```
public class ColorEntry
{
    public string ColorName { get; set; }
    public Color Color { get; set; }

    public static SolidColorBrush Brushify(Color c)
    {
        return new SolidColorBrush(c);
    }

    public SolidColorBrush TextColor(Color c)
    {
        return new SolidColorBrush(((c.R * 0.299 + c.G * 0.587 + c.B *
```

```
0.114) > 150) ? Colors.Black : Colors.White);
    }
}
```

XAML attribute usage

XAML

```
<object property="{x:Bind pathToFunction.FunctionName(functionParameter1,
functionParameter2, ...), bindingProperties}" ... />
```

Path to the function

The path to the function is specified like other property paths and can include dots (.), indexers or casts to locate the function.

Static functions can be specified using `XMLNamespace:ClassName.MethodName` syntax. For example, use the below syntax for binding to static functions in code-behind.

XAML

```
<Window
     xmlns:local="using:MyNamespace">
     ...
    <StackPanel>
        <TextBlock x:Name="BigTextBlock" FontSize="20" Text="Big text" />
        <TextBlock FontSize="{x:Bind
local:MyHelpers.Half(BigTextBlock.FontSize)}"
                   Text="Small text" />
    </StackPanel>
</Window>
```

C#

```
namespace MyNamespace
{
    static public class MyHelpers
    {
        public static double Half(double value) => value / 2.0;
    }
}
```

You can also use system functions directly in markup to accomplish simple scenarios like date formatting, text formatting, text concatenations, etc. For example:

XAML

```
<Window
     xmlns:sys="using:System"
     xmlns:local="using:MyNamespace">
     ...
     <CalendarDatePicker Date="{x:Bind sys:DateTime.Parse(TextBlock1.Text)}"
/>
     <TextBlock Text="{x:Bind sys:String.Format('{0} is now available in
{1}', local:MyPage.personName, local:MyPage.location)}" />
</Window>
```

If the mode is OneWay/TwoWay, then the function path will have change detection performed on it, and the binding will be re-evaluated if there are changes to those objects.

The function being bound to needs to:

- Be accessible to the code and metadata – so internal / private work in C#, but C++ will need methods to be public WinRT methods
- Overloading is based on the number of arguments, not type, and it will try to match to the first overload with that many arguments
- The argument types need to match the data being passed in – we don't do narrowing conversions
- The return type of the function needs to match the type of the property that is using the binding

The binding engine reacts to property change notifications fired with the function name and re-evaluate bindings as necessary. For example:

XAML

```
<DataTemplate x:DataType="local:Person">
   <StackPanel>
      <TextBlock Text="{x:Bind FullName}" />
      <Image Source="{x:Bind IconToBitmap(Icon, CancellationToken),
Mode=OneWay}" />
   </StackPanel>
</DataTemplate>
```

C#

```
public class Person : INotifyPropertyChanged
{
    //Implementation for an Icon property and a CancellationToken property
with PropertyChanged notifications
    ...
```

```
    //IconToBitmap function is essentially a multi binding converter between
several options.
    public Uri IconToBitmap (Uri icon, Uri cancellationToken)
    {
        var foo = new Uri(...);
        if (isCancelled)
        {
            foo = cancellationToken;
        }
        else
        {
            if (fullName.Contains("Sr"))
            {
               //pass a different Uri back
               foo = new Uri(...);
            }
            else
            {
                foo = icon;
            }
        }
        return foo;
    }

    //Ensure FullName property handles change notification on itself as well
as IconToBitmap since the function uses it
    public string FullName
    {
        get { return fullName; }
        set
        {
            fullName = value;
            OnPropertyChanged();
            OnPropertyChanged("IconToBitmap");
            //this ensures Image.Source binding re-evaluates when FullName
changes in addition to Icon and CancellationToken
        }
    }
}
```

> **Tip**
>
> You can use functions in `x:Bind` to achieve the same scenarios as what was supported through Converters and MultiBinding in WPF.

Function arguments

Multiple function arguments can be specified, separated by comma's (,)

- Binding Path – Same syntax as if you were binding directly to that object.
  - If the mode is OneWay/TwoWay then change detection will be performed and the binding re-evaluated upon object changes
- Constant string enclosed in quotes – quotes are needed to designate it as a string. Hat (^) can be used to escape quotes in strings
- Constant Number - for example -123.456
- Boolean – specified as "x:True" or "x:False"

Two way function bindings

In a two-way binding scenario, a second function must be specified for the reverse direction of the binding. This is done using the `BindBack` binding property. In the below example, the function should take one argument which is the value that needs to be pushed back to the model.

XAML

```
<TextBlock Text="{x:Bind a.MyFunc(b), BindBack=a.MyFunc2, Mode=TwoWay}" />
```

See also

- {x:Bind} Markup Extension

Bind hierarchical data and create a master/details view with Windows App SDK

Article • 06/29/2023

> **Note**
>
> Also see the **Master/detail UWP sample**.

You can make a multi-level master/details (also known as list-details) view of hierarchical data by binding items controls to **CollectionViewSource** instances that are bound together in a chain. In this topic we use the {x:Bind} markup extension where possible, and the more flexible (but less performant) {Binding} markup extension where necessary.

One common structure for Windows App SDK apps is to navigate to different details pages when a user makes a selection in a master list. This is useful when you want to provide a rich visual representation of each item at every level in a hierarchy. Another option is to display multiple levels of data on a single page. This is useful when you want to display a few simple lists that let the user quickly drill down to an item of interest. This topic describes how to implement this interaction. The **CollectionViewSource** instances keep track of the current selection at each hierarchical level.

We'll create a view of a sports team hierarchy that is organized into lists for leagues, divisions, and teams, and includes a team details view. When you select an item from any list, the subsequent views update automatically.

Prerequisites

This topic assumes that you know how to create a basic Windows App SDK app. For instructions on creating your first Windows App SDK app, see Create your first WinUI 3 (Windows App SDK) project.

Create the project

Create a new **Blank App, Packaged (WinUI 3 in Desktop)** project. Name it "MasterDetailsBinding".

Create the data model

Add a new class to your project, name it **ViewModel.cs**, and add this code to it. This will be your binding source class.

C#

```
using System.Collections.Generic;
using System.Linq;

namespace MasterDetailsBinding
{
    public class Team
    {
        public string Name { get; set; }
        public int Wins { get; set; }
```

```
        public int Losses { get; set; }
    }

    public class Division
    {
        public string Name { get; set; }
        public IEnumerable<Team> Teams { get; set; }
    }

    public class League
    {
        public string Name { get; set; }
        public IEnumerable<Division> Divisions { get; set; }
    }

    public class LeagueList : List<League>
    {
        public LeagueList()
        {
            AddRange(GetLeague().ToList());
        }

        public IEnumerable<League> GetLeague()
        {
            return from x in Enumerable.Range(1, 2)
                   select new League
                   {
                       Name = "League " + x,
                       Divisions = GetDivisions(x).ToList()
                   };
        }

        public IEnumerable<Division> GetDivisions(int x)
        {
            return from y in Enumerable.Range(1, 3)
                   select new Division
                   {
                       Name = string.Format("Division {0}-{1}", x, y),
                       Teams = GetTeams(x, y).ToList()
                   };
        }

        public IEnumerable<Team> GetTeams(int x, int y)
        {
            return from z in Enumerable.Range(1, 4)
                   select new Team
                   {
                       Name = string.Format("Team {0}-{1}-{2}", x, y, z),
                       Wins = 25 - (x * y * z),
                       Losses = x * y * z
                   };
        }
    }
}
```

Create the view

Next, expose the binding source class from the class that represents your page of markup. We do that by adding a property of type `LeagueList` to **MainWindow**.

C#

```
namespace MasterDetailsBinding
{
    public sealed partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
            ViewModel = new LeagueList();
        }
        public LeagueList ViewModel { get; set; }
    }
}
```

Finally, replace the contents of the **MainWindow.xaml** file with the following markup, which declares three **CollectionViewSource** instances and binds them together in a chain. The subsequent controls can then bind to the appropriate `CollectionViewSource`, depending on its level in the hierarchy.

XAML

```
<Window
    x:Class="MasterDetailsBinding.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MasterDetailsBinding"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid>
        <Grid.Resources>
            <CollectionViewSource x:Name="Leagues"
                Source="{x:Bind ViewModel}"/>
            <CollectionViewSource x:Name="Divisions"
                Source="{Binding Divisions, Source={StaticResource
Leagues}}"/>
            <CollectionViewSource x:Name="Teams"
                Source="{Binding Teams, Source={StaticResource
Divisions}}"/>

            <Style TargetType="TextBlock">
                <Setter Property="FontSize" Value="15"/>
                <Setter Property="FontWeight" Value="Bold"/>
```

```
            </Style>
            <Style TargetType="ListBox">
                <Setter Property="FontSize" Value="15"/>
            </Style>
            <Style TargetType="ContentControl">
                <Setter Property="FontSize" Value="15"/>
            </Style>
        </Grid.Resources>

        <StackPanel Orientation="Horizontal">

            <!-- All Leagues view -->
            <StackPanel Margin="5">
                <TextBlock Text="All Leagues"/>
                <ListBox ItemsSource="{Binding Source={StaticResource
Leagues}}"
                         DisplayMemberPath="Name"/>
            </StackPanel>

            <!-- League/Divisions view -->
            <StackPanel Margin="5">
                <TextBlock Text="{Binding Name, Source={StaticResource
Leagues}}"/>
                <ListBox ItemsSource="{Binding Source={StaticResource
Divisions}}"
                         DisplayMemberPath="Name"/>
            </StackPanel>

            <!-- Division/Teams view -->
            <StackPanel Margin="5">
                <TextBlock Text="{Binding Name, Source={StaticResource
Divisions}}"/>
                <ListBox ItemsSource="{Binding Source={StaticResource
Teams}}"
                         DisplayMemberPath="Name"/>
            </StackPanel>

            <!-- Team view -->
            <ContentControl Content="{Binding Source={StaticResource
Teams}}">
                <ContentControl.ContentTemplate>
                    <DataTemplate>
                        <StackPanel Margin="5">
                            <TextBlock Text="{Binding Name}"
                                       FontSize="15" FontWeight="Bold"/>
                            <StackPanel Orientation="Horizontal"
Margin="10,10">
                                <TextBlock Text="Wins:" Margin="0,0,5,0"/>
                                <TextBlock Text="{Binding Wins}"/>
                            </StackPanel>
                            <StackPanel Orientation="Horizontal"
Margin="10,0">
                                <TextBlock Text="Losses:" Margin="0,0,5,0"/>
                                <TextBlock Text="{Binding Losses}"/>
                            </StackPanel>
```

```
                        </StackPanel>
                    </DataTemplate>
                </ContentControl.ContentTemplate>
            </ContentControl>
        </StackPanel>
    </Grid>
</Window>
```

Note that by binding directly to the **CollectionViewSource**, you're implying that you want to bind to the current item in bindings where the path cannot be found on the collection itself. There's no need to specify the `CurrentItem` property as the path for the binding, although you can do that if there's any ambiguity. For example, the **ContentControl** representing the team view has its **Content** property bound to the `Teams` `CollectionViewSource`. However, the controls in the **DataTemplate** bind to properties of the `Team` class because the `CollectionViewSource` automatically supplies the currently selected team from the teams list when necessary.

See also

- Data binding overview
- Data binding in depth

Windows data binding and MVVM

Article • 06/29/2023

Model-View-ViewModel (MVVM) is a UI architectural design pattern for decoupling UI and non-UI code. With MVVM, you define your UI declaratively in XAML and use data binding markup to link it to other layers containing data and commands. The data binding infrastructure provides a loose coupling that keeps the UI and the linked data synchronized and routes user input to the appropriate commands.

Because it provides loose coupling, the use of data binding reduces hard dependencies between different kinds of code. This makes it easier to change individual code units (methods, classes, controls, etc.) without causing unintended side effects in other units. This decoupling is an example of the *separation of concerns*, which is an important concept in many design patterns.

Benefits of MVVM

Decoupling your code has many benefits, including:

- Enabling an iterative, exploratory coding style. Change that is isolated is less risky and easier to experiment with.
- Simplifying unit testing. Code units that are isolated from one another can be tested individually and outside of production environments.
- Supporting team collaboration. Decoupled code that adheres to well-designed interfaces can be developed by separate individuals or teams, and integrated later.
- Improving maintainability. Fixing bugs in decoupled code is less likely to cause regressions in other code.

In contrast with MVVM, an app with a more conventional "code-behind" structure typically uses data binding for display-only data, and responds to user input by directly handling events exposed by controls. The event handlers are implemented in code-behind files (such as MainWindow.xaml.cs), and are often tightly coupled to the controls, typically containing code that manipulates the UI directly. This makes it difficult or impossible to replace a control without having to update the event handling code. With this architecture, code-behind files often accumulate code that isn't directly related to the UI, such as database-access code, which ends up being duplicated and modified for use with other windows.

App layers

When using the MVVM pattern, an app is divided into the following layers:

- The **model** layer defines the types that represent your business data. This includes everything required to model the core app domain, and often includes core app logic. This layer is completely independent of the view and view-model layers, and often resides partially in the cloud. Given a fully implemented model layer, you can create multiple different client apps if you so choose, such as Windows App SDK and web apps that work with the same underlying data.
- The **view** layer defines the UI using XAML markup. The markup includes data binding expressions (such as x:Bind) that define the connection between specific UI components and various view-model and model members. Code-behind files are sometimes used as part of the view layer to contain additional code needed to customize or manipulate the UI, or to extract data from event handler arguments before calling a view-model method that performs the work.
- The **view-model** layer provides data binding targets for the view. In many cases, the view-model exposes the model directly, or provides members that wrap specific model members. The view-model can also define members for keeping track of data that is relevant to the UI but not to the model, such as the display order of a list of items. The view-model also serves as an integration point with other services such as data access code. For simple projects, you might not need a separate model layer, but only a view-model that encapsulates all the data you need.

Basic and advanced MVVM

As with any design pattern, there is more than one way to implement MVVM, and many different techniques are considered part of MVVM. For this reason, there are several different third-party MVVM frameworks supporting the various XAML-based platforms, including Windows App SDK. However, these frameworks generally include multiple services for implementing decoupled architecture, making the exact definition of MVVM somewhat ambiguous.

Although sophisticated MVVM frameworks can be very useful, especially for enterprise-scale projects, there is typically a cost associated with adopting any particular pattern or technique, and the benefits are not always clear, depending on the scale and size of your project. Fortunately, you can adopt only those techniques that provide a clear and tangible benefit, and ignore others until you need them.

In particular, you can get a lot of benefit simply by understanding and applying the full power of data binding and separating your app logic into the layers described earlier. This can be achieved using only the capabilities provided by the Windows App SDK, and

without using any external frameworks. In particular, the {x:Bind} markup extension makes data binding easier and higher performing than in previous XAML platforms, eliminating the need for a lot of the boilerplate code required earlier.

For additional guidance on using basic, out-of-the-box MVVM, check out the Customers Orders Database UWP sample on GitHub. Many of the other UWP app samples also use a basic MVVM architecture, and the Traffic App UWP sample includes both code-behind and MVVM versions, with notes describing the MVVM conversion.

See also

Topics

Data binding in depth
{x:Bind} markup extension

UWP MVVM Samples

Customers Orders Database sample
VanArsdel Inventory sample
Traffic App sample

Files, folders, and libraries with Windows App SDK

Article • 02/08/2024

Packaged Windows App SDK apps can leverage the powerful APIs provided by the Windows.Storage, Windows.Storage.Streams, and Windows.Storage.Pickers namespaces to efficiently read and write various data formats in files, as well as manage files and folders. This section covers essential topics such as reading and writing app settings, file and folder pickers, and accessing special sand-boxed locations like the Video/Music library. Learn how to optimize your app's file management capabilities with the Windows App SDK.

Windows 10 and later OS releases provide a wide variety of APIs related to files, folders, libraries, and settings for apps. These features are available via a combination of WinRT and .NET APIs provided by the Windows SDK and .NET SDK.

Read and write data with WinRT storage APIs

Packaged apps have access to all the WinRT storage APIs available to UWP apps. Whether you're migrating an existing UWP app or creating a new app, you can use these APIs to read and write data. For examples of using the storage APIs in a WinUI app, see Access files and folders with Windows App SDK and WinRT APIs.

Read and write data with .NET file APIs

In addition to the WinRT APIs, packaged apps can use the .NET APIs in the System.IO namespace to read and write data. When a new WinUI 3 project is created, its `Package.appxmanifest` file contains the following setting:

XML

```
<Capabilities>
  <rescap:Capability Name="runFullTrust" />
</Capabilities>
```

Declaring this restricted capability provides full access to the file system, registry, and other restricted capabilities. For more info, see Restricted capability declarations. In other words, the app has the same access to the file system as any other .NET desktop app when using the .NET file APIs.

For examples of using the .NET APIs in a WinUI app, see Access files and folders with Windows App SDK and .NET APIs.

Additional resources for working with files and folders

If you're developing packaged WinUI apps, the WinRT storage APIs can be a powerful tool for reading and writing data. The following UWP topics provide a wealth of information for developers looking to leverage these APIs in their apps.

Expand table

Topic	Description
Enumerate and query files and folders	Access files and folders in either a folder, library, device, or network location. You can also query the files and folders in a location by constructing file and folder queries.
Create, write, and read a file	Read and write a file using a StorageFile object.
Best practices for writing to files	Learn best practices for using various file writing methods of the FileIO and PathIO classes.
Get file properties	Get properties—top-level, basic, and extended—for a file represented by a StorageFile object.
Open files and folders with a picker	Access files and folders by letting the user interact with a picker. You can use the FolderPicker to gain access to a folder. **NOTE:** In a desktop app (which includes WinUI 3 apps), you can use file and folder pickers from Windows.Storage.Pickers. However, if the desktop app requires elevation to run, you'll need a different approach because these APIs aren't designed to be used in an elevated app. For an example, see FileSavePicker.
Save a file with a picker	Use FileSavePicker to let users specify the name and location where they want your app to save a file.
Accessing HomeGroup content	Access content stored in the user's HomeGroup folder, including pictures, music, and videos.
Determining availability of Microsoft OneDrive files	Determine if a Microsoft OneDrive file is available using the StorageFile.IsAvailable property.

Topic	Description
Files and folders in the Music, Pictures, and Videos libraries	Add existing folders of music, pictures, or videos to the corresponding libraries. You can also remove folders from libraries, get the list of folders in a library, and discover stored photos, music, and videos.
Track recently used files and folders	Track files that your user accesses frequently by adding them to your app's most recently used list (MRU). The platform manages the MRU for you by sorting items based on when they were last accessed, and by removing the oldest item when the list's 25-item limit is reached. All apps have their own MRU.
Track file system changes in the background	Track changes to the file system, even when the app isn't running.
Access the SD card	You can store and access non-essential data on an optional microSD card, especially on low-cost mobile devices that have limited internal storage.
Fast access to file properties	Efficiently gather a list of files and their properties from a library to use via the Windows Runtime APIs.

See also

Access files and folders with Windows App SDK and WinRT APIs

Access files and folders with Windows App SDK and .NET APIs

System.IO

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Access files and folders with Windows App SDK and WinRT APIs

Article • 06/29/2023

Packaged Windows App SDK apps can leverage WinRT APIs for reading and writing app settings, file and folder pickers, and special sand-boxed locations such as the Video/Music library. Additionally, any packaged desktop app can utilize both WinRT and Win32 APIs in the Windows SDK, as well as the APIs provided in the .NET SDK. This article provides guidance on how to use the WinRT storage APIs to query files and folders, retrieve file properties, and work with the **Pictures** library.

Query files and folders

The following example shows how to use the StorageFolder and StorageFile APIs to query the **Documents** library for files and folders. The example uses the `GetFilesInFolderAsync` method to recursively iterate through the folder structure and append the file names to a `StringBuilder` object.

C#

```
using System.Text;
using Windows.Storage;
...
private async Task<string> GetDocumentsContentsAsync()
{
    StorageFolder docsFolder = KnownFolders.DocumentsLibrary;
    StringBuilder outputBuilder = new();
    await GetFilesInFolderAsync(docsFolder, outputBuilder);

    return outputBuilder.ToString();
}

private async Task GetFilesInFolderAsync(StorageFolder folder, StringBuilder
outputBuilder)
{
    IReadOnlyList<IStorageItem> storageItem = await folder.GetItemsAsync();

    foreach (var item in storageItem)
    {
        if (item is StorageFolder)
        {
            await GetFilesInFolderAsync(item as StorageFolder,
outputBuilder);
        }
        else
```

```
        {
            outputBuilder.AppendLine($"Found {item.Name} in folder
{folder.Name}");
        }
    }
}
```

Get basic file properties

The following example takes the `GetFilesInFolderAsync` method from the previous example and adds the ability to retrieve the file size and date modified for each file. The example uses the BasicProperties API to retrieve the file size and date modified for each file, formats the file size, and appends the size and date modified to the `StringBuilder` object after each file and folder name.

C#

```
using System.Text;
using Windows.Storage;
using Windows.Storage.FileProperties;
...
private async Task GetFilesInFolderAsync(StorageFolder folder, StringBuilder
outputBuilder)
{
    IReadOnlyList<IStorageItem> storageItem = await folder.GetItemsAsync();

    foreach (var item in storageItem)
    {
        if (item is StorageFolder)
        {
            await GetFilesInFolderAsync(item as StorageFolder,
outputBuilder);
        }
        else
        {
            outputBuilder.AppendLine($"Found {item.Name} in folder
{folder.Name}");

            // Append each file's size and date modified.
            BasicProperties basicProperties = await
item.GetBasicPropertiesAsync();
            string fileSize = string.Format("{0:n0}", basicProperties.Size);
            outputBuilder.AppendLine($" - File size: {fileSize} bytes");
            outputBuilder.AppendLine($" - Date modified:
{basicProperties.DateModified}");
        }
    }
}
```

Working with the Pictures library

In this example, the app is configured to receive notifications when the **Pictures** library is updated. The example uses the StorageLibrary API to retrieve the **Pictures** library and the DefinitionChanged event to receive notifications when the library is updated. The `DefinitionChanged` event is invoked when the list of folders in the current library changes. The example uses the library's `Folders` property to iterate through the folders in the **Pictures** library and writes the folder name to the console.

C#

```
using Windows.Storage;
...
private async Task Configure()
{
    StorageLibrary picturesFolder = await
StorageLibrary.GetLibraryAsync(KnownLibraryId.Pictures);
    picturesFolder.DefinitionChanged += picturesFolder_DefinitionChanged;
}
private void picturesFolder_DefinitionChanged(StorageLibrary sender, object
args)
{
    foreach (StorageFolder item in sender.Folders)
    {
        Console.WriteLine($"Folder {item.Name} found.");
    }
}
```

See also

Access files and folders with Windows App SDK and .NET APIs

Files, folders, and libraries with Windows App SDK

Access files and folders with Windows App SDK and .NET

Article • 06/29/2023

Packaged Windows App SDK apps can leverage .NET APIs for reading and writing files, working with folders, and reading drive and volume information. Additionally, any packaged desktop app can utilize both WinRT and Win32 APIs in the Windows SDK, as well as the APIs provided in the .NET SDK. This article provides guidance on how to use the .NET System.IO APIs to read and write files, manage drives and folders, and work with memory streams to encode or decode string data.

Read and write files with .NET APIs

In the following example, `ReadWriteFiles` creates a new file, writes a set of integers to the file, and then reads the integers back from the file. The example uses the FileStream class to create a new file and to open the file for reading or writing. The example uses the BinaryWriter class to write the integers to the file and the BinaryReader class to read the integers from the file.

C#

```
using System.IO;
...
ReadWriteFiles("test.bin");
...
private void ReadWriteFiles(string fileName)
{
    if (File.Exists(fileName))
    {
        Console.WriteLine($"{fileName} already exists!");
        return;
    }

    using (FileStream fs = new(fileName, FileMode.CreateNew))
    {
        using BinaryWriter writer = new(fs);
        for (int i = 0; i < 11; i++)
        {
            writer.Write(i);
        }
    }

    using (FileStream fs = new(fileName, FileMode.Open, FileAccess.Read))
    {
        using BinaryReader reader = new(fs);
```

```
        for (int i = 0; i < 11; i++)
        {
            Console.WriteLine(reader.ReadInt32());
        }
    }
}
```

Manage drives and folders in .NET

The following example shows how to use the DirectoryInfo and Directory classes to create, delete, and manage folders. The example uses the `DirectoryInfo` class to create a new directory, create a subdirectory, and delete the directory. The `DirectoryInfo` class provides methods for creating, moving, and enumerating through directories and subdirectories. The `Directory` class provides *static* methods for creating, moving, and enumerating through directories and subdirectories.

C#

```
using System.IO;
...
private void FolderTest()
{
    FolderManagement(@"c:\MyDir", "Projects");
}
private void FolderManagement(string path, string subfolderName)
{
    DirectoryInfo di = new(path);
    try
    {
        // Create directory if it doesn't exist
        if (di.Exists)
        {
            Console.WriteLine("Path already exists.");
        }
        else
        {
            di.Create();
            Console.WriteLine("The directory was created successfully.");
        }

        // Create subdirectory if it doesn't exist
        string subfolderPath = Path.Combine(path, subfolderName);
        if (Directory.Exists(subfolderPath))
        {
            Console.WriteLine("Subfolder path already exists.");
        }
        else
        {
            di.CreateSubdirectory(subfolderName);
```

```
            Console.WriteLine("The subdirectory was created successfully.");
        }

        // Delete directory
        di.Delete(true);
        Console.WriteLine("The directory was deleted successfully.");
    }
    catch (Exception ex)
    {
        Console.WriteLine("The process failed: {0}", ex.ToString());
    }
}
```

This example using the static GetDrives method to retrieve information about all drives on the system. The DriveInfo class provides information about a drive, such as the drive type, label, file system, and available free space.

C#

```
using System.IO;
...
private void DriveManagement()
{
    DriveInfo[] drives = DriveInfo.GetDrives();

    foreach (DriveInfo d in drives)
    {
        Console.WriteLine($"Drive name: {d.Name}");
        Console.WriteLine($"  Drive type: {d.DriveType}");
        if (d.IsReady)
        {
            Console.WriteLine($"  Volume label: {d.VolumeLabel}");
            Console.WriteLine($"  File system type: {d.DriveFormat}");
            Console.WriteLine($"  Space available to user:
{d.AvailableFreeSpace, 15} bytes");
            Console.WriteLine($"  Total available space: {d.TotalFreeSpace,
15} bytes");
            Console.WriteLine($"  Total size of drive: {d.TotalSize, 15}
bytes ");
        }
    }
}
```

Encode and decode strings with MemoryStream

This example shows how to use the MemoryStream class to encode and decode string data. It first creates a `MemoryStream` to asynchronously write a string to a memory stream

and then read the string from the memory stream. The Encoding class is used to convert the string to a byte array and then write the byte array to the memory stream. A StreamReader is then used to asynchronously read the byte array from the memory stream and then convert the byte array back to a string by calling ReadToEndAsync.

C#

```
using System.IO;
using System.Text;
...
private async Task EncodeDecodeStringAsync(string inputData)
{
    using MemoryStream stream = new();
    var inputBytes = Encoding.UTF8.GetBytes(inputData);
    await stream.WriteAsync(inputBytes, 0, inputBytes.Length);
    stream.Seek(0, SeekOrigin.Begin);

    using StreamReader reader = new(stream);
    string text = await reader.ReadToEndAsync();
    Console.WriteLine(text);
}
```

> **Note**
>
> For information about converting between .NET streams and WinRT streams, see **How to: Convert between .NET and Windows Runtime streams**.

See also

Access files and folders with Windows App SDK and WinRT APIs

Files, folders, and libraries with Windows App SDK

Reference for Windows 11 settings

Article • 05/06/2024

The information provided on this page includes details for accessing the status of Windows Backup and Restore settings that are supported in Windows 11. This public documentation ensures effective data portability by providing third-party developers with a streamlined process to access the data. Settings that are supported on both Windows 10 and Windows 11 are documented in Reference for common Windows settings.

Settings status is accessed in one of two ways:

1. Via the Windows registry: For settings below that include registry details, please use that information to access the settings.
2. Via the Cloud Data Store Reader tool. These settings must be extracted from a data store to be readable. If the setting below does not list registry details, then the settings must be extracted using the Cloud Data Store Reader tool. For information on how to use this tool, see Cloud Data Store Settings Reader Tool (readCloudDataSettings.exe).

Cellular

User preferences that customize the Windows behavior when a cellular connection is available.

Registry values under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WcmSvc\CellularFailover

Expand table

Registry value	Type	Data	Description
AllowFailover	REG_DWORD	0 or 1	Use cellular whenever Wi-Fi is poor.

Registry values under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WcmSvc\SubscriptionManager\<ICCID>\<IMSI>

Expand table

Registry value	Type	Data	Description
UserCost	REG_DWORD	0 or 1	Maps the metered/unmetered state to each IMSI. 0: Unmetered. 1: Metered.

Registry values under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WwanSvc\DisallowAutoConnectBy

Expand table

Registry value	Type	Data	Description
<interface GUID>	REG_DWORD	0 or 1	Let Windows keep the device connected to cellular.

Registry values under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WwanSvc\RoamingPolicyForPhone<interface GUID>

Expand table

Registry value	Type	Data	Description
InternetAlwaysOn	REG_DWORD	0 or 1	Roaming or no roaming. When entering a roaming area, your data connection will be turned off if roaming is not allowed.

Gaming: Game Bar, Game Mode, Gaming Shortcuts

This setting controls settings related to gaming and controls such as Game bar and gaming shortcuts.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\GameDVR

Expand table

Registry value	Type	Data	Description
VKMSaveHistoricalVideo	REG_DWORD	0 or 1	Toggles save historical video enabled.
VKMTakeScreenshot	REG_DWORD	0 or 1	Toggles take screenshot enabled.
VKTakeScreenshot	REG_DWORD	ASCII Value for keys	Key binding for save historical screenshot.
VKSaveHistoricalVideo	REG_DWORD	ASCII Value for keys	Key binding for take screenshot.
VKMToggleBroadcast	REG_DWORD	0 or 1	Toggles broadcast enabled.
VKToggleBroadcast	REG_DWORD	ASCII Value for keys	Key binding for take broadcast.
VKMToggleCameraCapture	REG_DWORD	0 or 1	Toggles camera capture enabled.
VKToggleCameraCapture	REG_DWORD	ASCII Value for keys	Key binding for camera capture.
VKMToggleGameBar	REG_DWORD	0 or 1	Toggles game bar enabled.
VKToggleGameBar	REG_DWORD	ASCII Value for keys	Key binding for game bar.
VKMToggleMicrophoneCapture	REG_DWORD	0 or 1	Toggles microphone capture enabled.
VKToggleMicrophoneCapture	REG_DWORD	ASCII Value for keys	Key binding for microphone capture.
VKMToggleRecording	REG_DWORD	0 or 1	Toggles recording enabled.
VKToggleRecording	REG_DWORD	ASCII Value for keys	Key binding for recording.
VKMToggleRecordingIndicator	REG_DWORD	0 or 1	Toggles recording indicator enabled.
VKToggleRecordingIndicator	REG_DWORD	ASCII Value for keys	Key binding for recording indicator.

Registry values under HKCU\Software\Microsoft\GameBar

Expand table

Registry value	Type	Data	Description
UseNexusForGameBarEnabled	REG_DWORD	0 or 1	Key associated with toggle Gaming -> Game Bar -> "Allow your controller to open Game Bar"
AutoGameModeEnabled	REG_DWORD	0 or 1	Key associated with toggle Gaming -> Game Mode -> Game Mode.

Personalization - Start - Layout - Pins and recommendations

Specifies the start layout type.

Registry values under HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Ad

Expand table

Registry value	Type	Data	Description
Start_Layout	REG_DWORD	0 [Default], 1 [More Pins], 2 [More Recommendations]	The start layout.

Personalization - Start - Recommendations

Specifies whether recommendations for tips, shortcuts, new apps and more are shown.

Registry values under HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Adv

Expand table

Registry value	Type	Data	Description
Start_IrisRecommendations	REG_DWORD	0 or 1	Specifies whether recommendations are enabled.

Personalization - Taskbar - Alignment

This setting sets the alignment of the taskbar.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\TaskbarA

Expand table

Registry value	Type	Data	Description
SystemSettings_DesktopTaskbar_Al	REG_SZ	0 or 1	Specifies the taskbar alignment. 0 is left aligned. 1 is center aligned.

Personalization - Taskbar - Autohide

This setting sets the auto-hide behavior of the taskbar.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\StuckRects3\

Expand table

Registry value	Type	Data	Description
SystemSettings_DesktopTaskbar_Autohide	REG_BINARY	A binary blob.	This is an opaque binary blob copied from the following location on the backed up.
SystemSettings_Taskbar_Autohide	REG_BINARY	A binary blob.	This is an opaque binary blob copied from the following location on the backed up.

Personalization - Taskbar - Combine buttons

This setting enables combining buttons and hiding labels on the taskbar.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\TaskbarG

Expand table

Registry value	Type	Data	Description
SystemSettings_DesktopTaskbar_GroupingMode	REG_SZ	0, 1, or 2	0: Always, 1: When taskbar is full, 2: Never.

Personalization - Taskbar - Flashing

This setting enables flashing for taskbar apps.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\TaskbarF

Expand table

Registry value	Type	Data	Description
SystemSettings_DesktopTaskbar_Flashing	REG_SZ	0 or 1	Enables flashing for taskbar apps.

Personalization - Taskbar - Share window

This setting enables sharing any window from taskbar.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\TaskbarS

Expand table

Registry value	Type	Data	Description
SystemSettings_DesktopTaskbar_Sn	REG_SZ	0 or 1	Enables sharing any window from taskbar.

Personalization - Taskbar - Show desktop

This setting enables showing the desktop by clicking the far corner of the taskbar.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\TaskbarS

Expand table

Registry value	Type	Data	Description
SystemSettings_DesktopTaskbar_Sd	REG_SZ	0 or 1	Enables showing the desktop by clicking the far corner of the taskbar.

Personalization - Taskbar - Show recent searches

This setting enables showing recent searches when hovering on the search icon in taskbar.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Feeds\DSB\OpenOnHover

Expand table

Registry value	Type	Data	Description
SystemSettings_DesktopTaskbar_Sh	REG_SZ	0 or 1	Enables showing recent searches when hovering on the search icon in taskbar.

Personalization - Taskbar - Task view

This setting shows or hides the Task View button on the taskbar.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\ShowTas

Expand table

Registry value	Type	Data	Description
SystemSettings_DesktopTaskbar_TaskView	REG_SZ	0 or 1	Specifies whether the task view button is shown on the taskbar.

Personalization - Taskbar - Widgets button

This setting shows or hides the Widgets button on the taskbar.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\TaskbarD

Expand table

Registry value	Type	Data	Description
SystemSettings_DesktopTaskbar_Da	REG_SZ	0 or 1	Specifies whether the Widgets button is shown on the taskbar.

Personalization - Themes

This setting is used to set a personalized theme.

Type: Windows.Data.PersonalizationThemes.CurrentThemeType enumeration

CurrentThemeType Values

Expand table

Name	Value	Description
InboxTheme	0	In-box theme.
ContrastTheme	1	Contrast theme.

Type: Windows.Data.PersonalizationThemes.CurrentThemeType structure

CurrentThemeType Properties

Expand table

Name	Type	Description
type	**CurrentThemeType**	The current theme type.
basePersonalizationThemeName	wstring	The name of the current inbox theme applied in the system. The user may have done customization on top of this inbox theme.

Text Input

This setting helps to chose a theme for touch keyboard, voice typing, emoji and more, and input method editors.

Registry values under HKCU\Software\Microsoft\TabletTip\1.7

Expand table

Registry value	Type	Data	Description
IsKeyBackgroundEnabled	REG_DWORD	0 or 1	Indicates if key background is enabled.
KeyLabelSize	REG_DWORD	Min value: 0, Max value: 2	Key label size.
UserKeyboardScalingFactor	REG_DWORD	Min value: 1, Max value: 1000	User keyboard scaling factor.
SelectedThemeIndex	REG_DWORD	Min value: 0, Max value: 15	Selected theme index.
SelectedThemeName	REG_SZ	One of the following values: "LightTheme", "DarkTheme", "ColorPopTheme", "BlackWhiteTheme", "PoppyRedTheme", "IceBlueTheme", "PlatinumTheme", "TangerineTidesTheme", "LilacRiverTheme", "SilkyDawnTheme", "IndigoBreezeTheme", "PinkBlueTheme", "GreenPurpleTheme", "PinkOrangeTheme", "CustomTheme"	Selected theme name.
ThemeDataVersion	REG_DWORD	2	Theme data version.

Typing

This setting contains toggles and other settings related to touch keyboard, text suggestions and preferences.

Registry values under HKCU\Software\Microsoft\input\Settings

Expand table

Registry value	Type	Data	Description
EnableAutoShiftEngage	REG_DWORD	1 or 0	Enables auto shift engage.
EnableDoubleTapSpace	REG_DWORD	1 or 0	Enables double-tap space.
EnableAutocorrection	REG_DWORD	1 or 0	Enables auto-correction.
IsVoiceTypingKeyEnabled	REG_DWORD	1 or 0	Indicates if voice typing key is enabled.
MultilingualEnabled	REG_DWORD	1 or 0	Indicates in multilingual is enabled.
EnableHwkbTextPrediction	REG_DWORD	An integer.	ASCII Value for keys.

VPN

VPN settings that apply to all VPN connections configured in the Settings app.

Registry values under HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\RasMan\Parameters

Expand table

Registry value	Type	Data	Description
NoCostedNetwork	REG_DWORD	0 or 1	Block VPN over metered networks.
NoRoamingNetwork	REG_DWORD	0 or 1	Block VPN while roaming.

Wi-Fi

Global random hardware addresses preference.

Registry values under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WlanSvc\Interfaces\

<Interface GUID>

Expand table

Registry value	Type	Data	Description
RandomMacState	REG_BINARY	00 00 00 00 or 01 00 00 00	Whether to use random hardware addresses for newly configured Wi-Fi networks.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Reference for Windows 11 and Windows 10 settings

Article • 04/03/2024

The information provided on this page includes details for accessing the status of Windows Backup and Restore settings that are supported for both Windows 10 and Windows 11. This public documentation ensures effective data portability by providing third-party developers with a streamlined process to access the data. Settings that are supported on Windows 11 only are documented in Reference for Windows 11 settings.

Settings status is accessed in one of two ways:

1. Via the Windows registry: For settings below that include registry details, please use that information to access the settings.
2. Via the Cloud Data Store Reader tool. These settings must be extracted from a data store to be readable. If the setting below does not list registry details, then the settings must be extracted using the Cloud Data Store Reader tool. For information on how to use this tool, see Cloud Data Store Settings Reader Tool (readCloudDataSettings.exe).

App compatibility

The app compatibility setting is a JSON file that describes compatibility information for apps installed on the device. The path to this JSON is:

`C:\Windows\appcompat\Backup\[user SID].json`

The format of the backup JSON file. Install, Update, and Uninstall nodes contain arrays of applications, which contain arrays of files. The following code segment describes the format of the file and provides descriptions for each field.

JSON

```
{
  "Install": [
    {
      "path": App's uninstall registry path,
      "programId": The unique identifier of the installed Win32 application,
      "compatFlags": Applicable backup/restore compatibility flags OR'ed together. For the list of supported values, see
the table below,
      "restoreAction": Actions to be taken on app restore,
      "files": [
        {
          "name": File name,
          "path": File path,
          "osComponent": Boolean stating if the file is an OS file,
          "size": The file size as a 32-bit value,
          "magic": The PE header's magic number,
          "peHeaderHash": Hash of the file's PE header,
          "sizeOfImage": PE header's SizeOfImage value,
          "peChecksum": PE header's CheckSum value,
          "linkDate": PE header's TimeDateStamp value,
          "linkerVersion": PE header's MarjorImageVersion and MinorImageVersion,
          "binFileVersion": File version obtained from GetFileVersionInfo,
          "binProductVersion": Product version obtained from GetFileVersionInfo,
          "binaryType": Type of binary (e.g. PE64_AMD64),
          "created": File creation time obtained from file system,
          "modified": File modification time obtained from file system,
          "lastAccessed": File access time obtained from file system,
          "verLanguage": Language obtained from GetFileVersionInfo,
          "id": Unique identifier obtained from hashing file contents,
          "switchBackContext": Value for OS runtime compatibility fixes,
          "sigDisplayName": Display name obtained from the file signature,
          "sigPublisherName": Publisher name obtained from the file signature,
          "sigMoreInfoURL": URL obtained from the file signature,
          "fileVersion": File version obtained from GetFileVersionInfo,
          "companyName": Company name obtained from GetFileVersionInfo,
          "fileDescription": File description obtained from GetFileVersionInfo,
          "internalName": Internal name obtained from GetFileVersionInfo,
          "legalCopyright": Copyright information obtained from GetFileVersionInfo,
          "originalFileName": Original filename obtained from GetFileVersionInfo,
          "productName": Product name obtained from GetFileVersionInfo,
          "productVersion": Product version obtained from GetFileVersionInfo,
          "peImageType": Image type obtained from PE header,
          "peSubsystem": Subsystem obtained from PE header,
          "runLevel": Executable's runlevel obtained from app manifest,
          "uiAccess": UI access obtained from app manifest,
          "crcChecksum": File's CRC checksum,
```

```
          "clrVersion": CLR version obtained from app manifest,
          "boeProgramId": Unique ID describing the application,
          "boeProgramName": Same as "productName", if it exists. Otherwise same as "name",
          "boeProgramPublisher": Same as "companyName", if it exists. Otherwise same as "fileDescription", if it exists,
          "boeProgramVersion": Same as "productVersion", if it exists. Otherwise same as "fileVersion", if it exists.
Otherwise same as "binProductVersion", if it exists. Otherwise same as "binFileVersion", if it exists,
          "boeProgramLanguage": Same as "verLanguage", if it exists,
          "fileSize": File's size as a 64-bit number,
          "peCharacteristics": Image characteristics obtained from PE header,
          "sha256": SHA256 hash of file,
        }
      ]
    },
  ],
  "Update": [
    { }
  ],
  "Uninstalled": [
    { }
  ]
}
```

The value of the *compatFlags* field of the backup JSON file is a combination of the following values, computed with an OR operation.

Expand table

Value	Description
0x00000000	App is compatible.
0x00000001	App is incompatible with this OS.
0x00000002	App is developed for specific OEM.
0x00000004	App is a redistributable package.
0x00000008	Plugin or extension that requires the main app.
0x00000010	Middleware.
0x00000020	App was acquired from third party such as a game store.
0x00000040	App is incompatible with HVCI.
0x00000080	App is incompatible with Kernet CET.
0x00000100	App depends on a kernel mode driver.
0x00000200	App is not compatible with Android.
0x00000400	App will not work if S mode is enabled.
0x00000800	App is only available in a certain market.
0x00001000	App does not support this architecture.
0x00002000	App is targeted for other device families.
0x00004000	Generic hardware incompatibility.
0x00008000	App is incompatible up to certain version.

AppList Backup Data

Type: Windows.Data.Apps.AppMetaData structure

AppMetaData values

Unless otherwise specified, the Values are found in the AppXManifest for MSIX packages, and Uninstall registries for other formats.

Expand table

Name	Type	Description
appId	wstring	PackageFamilyName (packaged) or ProductCode (unpackaged).
installSource	wstring	Description of type of installer for the app. See *installSource* for supported values.
lightIconInfo	**IconInfo**	Information about the light icon.
darkIconInfo	**IconInfo**	Information about the dark icon.
appName	wstring	App display name.
publisher	wstring	Publishers name of the app from Add Remove Programs.
lastLaunchTime	uint64	This is a calculated value that represents the last time the app was launched.
appVersion	wstring	Version of the app from Add Remove Programs.
appLanguage	wstring	Language list in Add Remove Programs.
appArch	wstring	Architecture specified in Add Remove Programs.
reinstallId	wstring	Reinstall ID specified in Add Remove Programs.
productUrl	wstring	Product URL Specified in Add Remove Programs.
isPinned	bool	Boolean indicating if this app was pinned to the start menu.
wingetID	wstring	Identifier to indicate if this app can be installed through winget, and the winget ID.
wingetSource	wstring	Specifies where the app was sourced from through the Winget APIs. See *wingetSource* for supported values.

Supported values for the *installSource* field.

Expand table

Value	Description
"Store MSIX"	An MSIX from the Microsoft Store.
"Sideloaded MSIX"	A sideloaded MSIX.
"Edge PWA MSIX"	A PWA MSIX.
"Unknown MSIX"	Not one of the other MSIX values.
"Store Win32"	A non-UWP app from the Microsoft Store.
"Android"	An Android app.
"External MSI"	An external MSI.

Supported values for the *wingetSource* field.

Expand table

Value	Description
"External"	Installed from the web, but winget has an ID a match in the winget catalog.
"Winget"	Installed from the winget catalog.
"Spark"	Microsoft Store non-UWP app.
"MSStore"	Microsoft Store MSIX app.
"NoReliableInfo"	No data provided for source.

Type: Windows.Data.Apps.IconInfo structure

IconInfo values

The paths to the icons used to generate the IconInfo, can be found in the AppXManifest for MSIX packages, and Uninstall registries for other formats.

Expand table

Name	Type	Description
appIconAssetId	wstring	The ID representing an icon from the cloud. The appIconAssetId represents the icon stored in the cloud which was generated from the app installer either through add remove programs metadata or AppxManifest files.
isPlated	bool	Plated or not.

Type: Windows.Data.Apps.ShortcutInfo structure

ShortcutInfo values

Expand table

Name	Type	Description
targetPath	wstring	Link to the executable that launching the tile will shell execute.
shortcutArgs	wstring	Arguments provided on launch.

Type: Windows.Data.Apps.AppLevelTileInfo structure

AppLevelTileInfo values

Expand table

Name	Type	Description
tileId	wstring	UnifiedTileIdentifier for tile
lightIconInfo	**IconInfo**	UnifiedTileIdentifier for light tile
darkIconInfo	**IconInfo**	UnifiedTileIdentifier for dark tile
displayName	wstring	String displayed on tile
sortName	wstring	String value used for sorting in search
packageId	wstring	PackageFamilyName (packaged), ProductCode (unpackaged), empty for unmapped products (appId is specified in **AppMetaData**)
shortcut	**ShortcutInfo**	Shortcut information.
suiteName	wstring	String name for a collection of apps.

Type: Windows.Data.Apps.FileInfo structure

FileInfo values

FileInfo values are populated from the *App Compatibility* JSON above. All FileInfo values are provided as WSTRINGs, though some of the values in the JSON are different.

Expand table

Name	Type	Description
name	wstring	Read from App compatibility JSON. Optional.
path	wstring	Read from App compatibility JSON. Optional.
osComponent	wstring	Read from App compatibility JSON. Optional.
size	wstring	Read from App compatibility JSON. Optional.
magic	wstring	Read from App compatibility JSON. For information on the PE header, see PE Format.

Name	Type	Description
peHeaderHash	wstring	Read from App compatibility JSON. Optional.
sizeOfImage	wstring	Read from App compatibility JSON. Optional.
peChecksum	wstring	Read from App compatibility JSON. Optional.
linkDate	wstring	Read from App compatibility JSON. Optional.
linkerVersion	wstring	Read from App compatibility JSON. Optional.
binFileVersion	wstring	Read from App compatibility JSON. Optional.
binProductVersion	wstring	Read from App compatibility JSON. Optional.
binaryType	wstring	Read from App compatibility JSON. Optional.
created	wstring	Read from App compatibility JSON. Optional.
modified	wstring	Read from App compatibility JSON. Optional.
lastAccessed	wstring	Read from App compatibility JSON. Optional.
verLanguage	wstring	Read from App compatibility JSON. Optional.
id	wstring	Read from App compatibility JSON. Optional.
switchBackContext	wstring	Read from App compatibility JSON. Optional.
sigDisplayName	wstring	Read from App compatibility JSON. Optional.
sigPublisherName	wstring	Read from App compatibility JSON. Optional.
sigMoreInfoURL	wstring	Read from App compatibility JSON. Optional.
fileVersion	wstring	Read from App compatibility JSON. Optional.
companyName	wstring	Read from App compatibility JSON. Optional.
fileDescription	wstring	Read from App compatibility JSON. Optional.
internalName	wstring	Read from App compatibility JSON. Optional.
legalCopyright	wstring	Read from App compatibility JSON. Optional.
originalFileName	wstring	Read from App compatibility JSON. Optional.
productName	wstring	Read from App compatibility JSON. Optional.
productVersion	wstring	Read from App compatibility JSON. Optional.
peImageType	wstring	Read from App compatibility JSON. Optional.
peSubsystem	wstring	Read from App compatibility JSON. Optional.
runLevel	wstring	Read from App compatibility JSON. Optional.
uiAccess	wstring	Read from App compatibility JSON. Optional.
crcChecksum	wstring	Read from App compatibility JSON. Optional.
clrVersion	wstring	Read from App compatibility JSON. Optional.
boeProgramId	wstring	Read from App compatibility JSON. Optional.
boeProgramName	wstring	Read from App compatibility JSON. Optional.
boeProgramPublisher	wstring	Read from App compatibility JSON. Optional.
boeProgramVersion	wstring	Read from App compatibility JSON. Optional.
boeProgramLanguage	wstring	Read from App compatibility JSON. Optional.
fileSize	wstring	Read from App compatibility JSON. Optional.
peCharacteristics	wstring	Read from App compatibility JSON. Optional.
sha256	wstring	Read from App compatibility JSON. Optional.
aumid	wstring	Application user model ID. Optional. For more information, see Find the Application User Model ID of an installed app

Type: Windows.Data.Apps.AppCompatInfo structure

AppCompatInfo values

Expand table

Name	Type	Description
name	wstring	Read from App compatibility JSON.
path	wstring	Read from App compatibility JSON.
appId	wstring	Read from App compatibility JSON.
programId	wstring	Read from App compatibility JSON.
compatFlags	wstring	Read from App compatibility JSON.
restoreActions	wstring	Read from App compatibility JSON
files	**vector <FileInfo>**	Read from App compatibility JSON.

Type: Windows.Data.Apps.DeviceMetadata structure

DeviceMetadata values

Expand table

Name	Type	Description
userIntent	uint32	This is a bit array of values specifying user intent during Windows setup. See *userIntent* values. This data is read from HKCU\Software\Microsoft\Windows\CurrentVersion\CloudExperienceHost\Intent
predictedUserIntent	uint32	Windows sets flag to indicate that the user had one or more apps that are a signal of a developer.
devModeEnabled	bool	Whether the user has specified Developer Mode from Windows Settings. This data is read from from HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\AppModelUnlock

Supported *userIntent* values.

Expand table

Value	Description
0b00000001	None
0b00000010	Gaming
0b00000100	Family
0b00001000	Creativity
0b00010000	Schoolwork
0b00100000	Entertainment
0b01000000	Business
0b10000000	Development

Autoplay

This setting helps to set defaults for removable drives and memory cards

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\AutoplayHandlers

Expand table

Registry value	Type	Data	Description
DisableAutoplay	REG_DWORD	0/1	Toggle the use of autoplay for all media and devices.
UserChosenExecuteHandlers\CameraAlternate\ShowPicturesOnArrival	REG_SZ	A string.	User selected default autoplay behavior for removable drive. See table below for supported values.
UserChosenExecuteHandlers\ StorageOnArrival	REG_SZ	A string.	User selected default autoplay behavior for memory card. See table below for supported values.
EventHandlersDefaultSelection\CameraAlternate\ShowPicturesOnArrival	REG_SZ	A string.	System default autoplay behavior for removable drive. See table below for supported values.
EventHandlersDefaultSelection\StorageOnArrival	REG_SZ	A string.	System default autoplay behavior for memory card. See table below for supported values.

Supported data values for ShowPicturesOnArrival

Expand table

Data value	Description
dsd9eksajf9re3669zh5z2jykhws2jy42gypaqjh1qe66nyek1hg!content!import	Import Photos and Videos (Photos)
MSPlayMediaOnArrival	Play media on arrival.
MSOpenFolder	Open folder.
MSPromptEachTime	Prompt each time.
OneDriveAutoPlay	Import Photos and Videos (OneDrive).
MSTAKENOACTION	Take no action.

Supported data values for StorageOnArrival

Expand table

Data value	Description
MSTAKENOACTION	Take no action.
MSOpenFolder	Open folder.
MSStorageSense	Configure Storage Settings (Settings).
MSPromptEachTime	Prompt each time.

Background

Setting for managing and personalizing the desktop background.

Type: Windows.Data.Background.WallpaperPosition enumeration

WallpaperPosition values

Expand table

Name	Value	Description
Fill	0	Fill.
Fit	1	Fit.
Stretch	2	Stretch.

Name	Value	Description
Tile	3	Tile.
Center	4	Center.
Span	5	Span

Type: Windows.Data.Background.WallpaperKind enumeration

WallpaperKind values

Expand table

Name	Value	Description
SolidColor	0	Solid color.
Image	1	Image.
Slideshow	2	Slideshow.
Spotlight	3	Spotlight.

Type: Windows.Data.Background.DesktopWallpaper structure

The scope of this type is per device.

DesktopWallpaper Properties

Expand table

Name	Type	Description
kind	**WallpaperKind**	Specifies whether current background is set as Wallpaper or Solid Color or Slideshow or Spotlight.
position	**WallpaperPosition**	Specifies how wallpaper or slideshow images are positioned on background.
color	Windows.Data.Common.Color	Specifies the solid color value if background is selected as Solid Color
itemId	wstring	The unique ID for the wallpaper or slideshow uploaded during backup.
contentUri	wstring	The url for the wallpaper or slideshow uploaded during backup.
intervalInSeconds	uint64	The interval between images of slideshow if background is selected as slideshow.
shuffle	bool	Signifies whether slideshow images are shuffled if background is selected as slideshow
syncRootRelativePath	wstring	Signifies path to slideshow folder if background is selected as slideshow.

Calling

The settings below are for a deprecated Windows calling experience and are no longer read by the operating system, however the settings data may be present on user devices or in the cloud.

Type: Windows.data.calling.Settings structure

Settings properties

Expand table

Name	Type	Description
perKeySettings	map<wstring, KeyPathSettings>	A map of per-key settings

Type: Windows.data.calling.KeyPathSettings structure

KeyPathSettings properties

Expand table

Name	Type	Description
values	map<wstring, wstring>	A map of key/path settings

Type: Windows.data.calling.callhistoryItem structure

callhistoryItem properties

Expand table

Name	Type	Description
uniqueId	String	Unique identifier for the record.
phoneNumber	String	Phone number of the caller.
calltype	Enum CallType	Possible values: Outgoing, IncomingAnswered, IncomingMissed, IncomingRinging, OutgoingMissed, OutgoingRinging.
voicemailCall	Bool	Indicates whether the call was a voice mail.
videocall	Bool	Indicates whether the call was a video call.
Seen	Bool	Indicates whether the call history was seen by the user.
callerIdBlocked	Bool	Indicates whether the call was blocked.
emergencycall	Bool	Indicates whether call was an emergency call.
linenumber	String	The number of the phone line that received the call.
lineName	String	The phone line's name.
callerLocation	String	Caller location.
callerCategory	String	Caller category.
callerCategoryDescription	String	Caller category description.
Calltimestamp	Unit64	unix time stamp.
Callarrivaltime	Unit64	Unix time the call arrived on the device.
callEndTime	Unit64	Unix time stamp when the call was ended.

Type: Windows.Data.calling.callhistory structure

callhistory Properties

Expand table

Name	Type	Description
historyItems	Map<string, CallHistoryItem>	A collection of call history items where the keys are each history item's UniqueId.
highestSequenceNumber	Unit32	Highest sequence number issued, used for internal business logic.

Type: Windows.data.calling.callfavorites structure

callfavorites Properties

Expand table

Name	Type	Description
favoriteItems	vector<CallFavoriteItem>	A collection of calling favorites.

Type: Windows.data.calling.CallFavoriteItem structure

CallFavoriteItem Properties

Expand table

Name	Type	Description
phoneNumber	String	A collection of calling favorites.
displayName	String	Display name of the favorite.
lineNumber	String	Line number the favorite is associated with.
phoneNumberName	Uint32	Phone number name.
remoteIdHash	String	Remote ID hash.
propHash	String	Property hash.
isVideoCall	Bool	Indicates whether the item is a video call.
isPublicSwitchTelephoneNetwork	Bool	Indicates whether the item is a telephone call.
applicationId	String	Application ID.
callbackToken	String	Callback token.
UniqueId	Unit64	A unique identifier for the item.

Contrast themes

Settings related to high-contrast themes.

Type: Windows.Data.ContrastThemes.CurrentThemeType enumeration

CurrentThemeType values

Expand table

Name	Value	Description
Contrast_Inbox	0	In-box high contrast theme.
Contrast_Custom	1	Customized high contrast theme.
Backup_disabled	2	Backup disabled.

Type: Windows.Data.ContrastThemes.SynchedTheme structure

SynchedTheme properties

Expand table

Name	Value	Description
type	**CurrentThemeType**	Specifies whether current theme is personalized theme or inbox high contrast theme or customized high contrast theme.

Name	Value	Description
currentThemePath	wstring	The complete path to the currently applied theme file.
contrastThemePath	wstring	The complete path to the last applied high contrast theme file.This can be same as of *currentThemePath* if current theme is high contrast.
baseContrastThemeName	wstring	Specifies the base in-box high contrast theme is applied. Any customization may have been applied on top of these themes. Supported values are "High Contrast #1", " or "High Contrast #2", "High Contrast Black", or "High Contrast White"
customThemeName	wstring	The user-defined name for a customized contrast theme.
rgbBackground	**Windows.Data.Common.Color**	Theme background color.
rgbText	**Windows.Data.Common.Color**	Theme text color.
rgbHyperlink	**Windows.Data.Common.Color**	Theme hyperlink color.
rgbInactiveText	**Windows.Data.Common.Color**	Theme inactive text color.
rgbSelectedText1	**Windows.Data.Common.Color**	Signifies color value of Text of highlighted text.
rgbSelectedText2	**Windows.Data.Common.Color**	Signifies color value of Text of highlight.
rgbButtonText1	**Windows.Data.Common.Color**	Signifies color value of Text of button text.
rgbButtonText2	**Windows.Data.Common.Color**	Signifies color value of Face of button.

Type: Windows.Data.Common.Color structure

Color values

Expand table

Name	Value	Description
red	uint8	Red channel value of an RGBA color.
green	uint8	Green channel value of an RGBA color.
blue	uint8	Blue channel value of an RGBA color.
alpha	uint8	Alpha channel value of an RGBA color.

Date and Time

Settings related to date and time.

Registry values under HKLM\SYSTEM\CurrentControlSet\Services\tzautoupdate

Expand table

Registry value	Type	Data	Description
start	REG_DWORD	3/4	3: Set timezone toggle value on. 4: Set Timezone toggle value off.

Registry values under HKLM\SYSTEM\CurrentControlSet\Control\TimeZoneInformation

Expand table

Registry value	Type	Data	Description
timeZone	REG_SZ	System timezone as string.	The currently set time zone for the system.

Do not disturb

Set "do not disturb" status manually or automatically, so that notifications will be sent directly to the notification center.

Type: Windows.Data.DoNotDisturb.ChangeReason enumeration

ChangeReason values

Expand table

Name	Value	Description
Default	0	Default setting.
User	1	User setting.

Type: BoolWithMetadata structure

BoolWithMetadata properties

Expand table

Name	Value	Description
value	bool	The boolean value.
changeReason	**ChangeReason**	The reason a user changed their profile.

Type: Windows.Data.DoNotDisturb.QuietHoursProfile structure

This setting is multi-instance. The following is an example command line for retrieving this type:

```
readCloudDataSettings.exe enum -type:windows.data.donotdisturb.QuietHoursProfile
```

For more information on retrieving multi-instance settings, see Cloud Data Store Settings Reader Tool.

QuietHoursProfile Properties

Expand table

Name	Type	Description
isInitialized	bool	CDS data initialize status.
settings	map<uint64, bool>	Map of the boolean state of the settings. Specifies when a given settings checkbox is checked or unchecked.
allowedContacts	set<wstring>	The set of allowed contacts as a list.
allowedApps	set<wstring>	The list of app names to set notifications on.
defaultAllowedAppsRemoved	set<wstring>	The list of names of the default app names removed.

Type: Windows.Data.DoNotDisturb.QuietHoursSettings structure

QuietHoursSettings Properties

Expand table

Name	Type	Description
isInitialized	bool	CDS data initialize status. .

Name	Type	Description
selectedProfile	wstring	The string value of the user profile. "Unrestricted" or "Priority Only"
shouldShowSummaryToast	BoolWithMetadata	Sets whether the summary toast should be shown and the associated reason.

Type: Windows.Data.DoNotDisturb.QuietMoment structure

This setting is multi-instance. The following is an example command line for retrieving this type:

```
readCloudDataSettings.exe enum -type:windows.data.donotdisturb.quietmoment
```

For more information on retrieving multi-instance settings, see Cloud Data Store Settings Reader Tool.

QuietMoment Properties

Expand table

Name	Type	Description
isInitialized	bool	CDS data initialize status. Default value is false.
isEnabled	bool	Settings enabled or disabled.
assignedProfile	wstring	The value of the user profile set by automatic rules. "Unrestricted", "Priority Only", or "Alarms Only"
startTime	TimeSpan	When automatic rules should start.
endTime	TimeSpan	When automatic rules should end.
repeatType	uint32	How frequently the rules should apply. "Daily", "Weekend", or "Weekdays"
shouldShowActiveToast	BoolWithMetadata	Sets whether the active toast should be shown and the associated reason.

File Explorer Classic

Settings related to the classic Windows File Explorer.

Type: Windows.Data.FileExplorerClassic.ShellStateSetting structure

ShellStateSetting Properties

Expand table

Name	Type	Description
bmigratedFromSSF	bool	Indicates whether the shell state is migrated from legacy setting framework to current solution.
bshowAllObjects	bool	Indicates whether Shell state should show all objects.
bshowExtensions	bool	Indicates whether Shell state should show all extensions.
bshowCompColor	bool	Indicates whether Shell state should show all colors.
bdoubleClickInWebView	bool	Indicates whether Shell state should show all objects.
bdontPrettyPath	bool	Indicates whether Shell state should show pretty path
showInfoTip	bool	Indicates whether Shell state should show info tip.
noConfirmRecycle	bool	Indicates whether Shell state should should confirm recycle option.
showSuperHidden	bool	Indicates whether Shell state should show super hidden.
sepProcess	bool	Indicates whether Shell state should show the separation process.
iconsOnly	bool	Indicates whether Shell state should show icons only.
showTypeOverlay	bool	Indicates whether Shell state should show type overlay.

Type: Windows.Data.FileExplorerClassic.CabinetStateSettings structure

CabinetStateSettings Properties

Expand table

Name	Type	Description
migratedFromSSF	bool	Indicates whether the cabinet state is migrated from legacy setting framework to current solution.
fullPathTitle	bool	Indicates whether the cabinet state setting should show full path title.
saveLocalView	bool	Indicates whether the cabinet state setting should save local view.
newWindowMode	bool	Indicates whether the cabinet state setting should open in new window mode.

Type: Windows.Data.FileExplorerClassic.AdvancedSettings structure

Expand table

Name	Type	Description
migratedFromSSF	bool	Indicates whether advanced settings are migrated from legacy setting framework to current solution.
alwaysShowMenus	bool	Indicates whether advanced settings should always show menu.
folderContentsInfoTip	bool	Indicates whether advanced settings should show folder contents info tip.
hideDrivesWithNoMedia	bool	Indicates whether advanced settings should hide the drives with no media present.
navPaneExpandToCurrentFolder	bool	Indicates whether advanced settings should expand the navigation pane to current folder.
navPaneShowAllFolders	bool	Indicates whether advanced settings should expand the navigation panel to show all folders.
navPaneShowFavorites	bool	Indicates whether advanced settings should expand the navigation panel to show all favorites.
persistBrowsers	bool	Indicates advanced settings for file explorer should persist browsers.
sharingWizardOn	bool	Indicates whether sharing wizard is on for advanced settings.
showDriveLetters	bool	Indicates whether drives letters are shown for advanced settings.
showPreviewHandlers	bool	Indicates whether to show preview handlers.
typeAhead	bool	Indicates whether to show type ahead.
showStatusBar	bool	Indicates whether the file explorer should show status bar.
showLibraries	bool	Indicates whether the file explorer advanced settings should show libraries.
showCompColor	bool	Indicates whether to show comp color for advanced settings.

Type: Windows.Data.FileExplorerClassic.ExplorerSettings structure

ExplorerSettings Properties

Expand table

Name	Type	Description
migratedFromSSF	bool	Indicates whether the explorer settings are migrated from legacy setting framework to current solution.
underlineIconsAlways	bool	Indicates whether Explorer settings should always underline icons.
underlineIconsNever	bool	Indicates whether Explorer settings should never underline icons.
underlineIconsAsBrowser	bool	Indicates whether Explorer settings should underline icons as browser.

Type: Windows.Data.FileExplorerClassic.SearchSettings structure

SearchSettings Properties

Expand table

Name	Type	Description
migratedFromSSF	bool	Indicates whether the search settings are migrated from legacy setting framework to current solution.
searchOnly	bool	Indicates whether search setting show search only.
wholeFileSystem	bool	Indicates whether search setting show the whole file system.
systemFolders	bool	Indicates whether search setting show system folders.
archivedFiles	bool	Indicates whether search setting show archived files.

Type: Type: Windows.Data.FileExplorerClassic.RegistrySettings structure

These are blobs that are in the registry. There are three things that use registry stream settings:

Expand table

Name	Type	Description
migratedFromSSF	bool	Indicates whether the registry settings are migrated from legacy setting framework to current solution.
RibbonQat	blob	A single blob.
detailsPreviewPaneSettings	unit32	Details about preview pane settings.
readingPaneSettings	unit32	Indicates reading pane settings.
navigationPaneVisible	bool	Indicates whether navigation pane is visible or not.

RegistrySettingsProperties

Expand table

Name	Type	Description
migratedFromSSF	bool	Indicates whether registry settings are migrated from legacy setting framework to current solution.
ribbonQat	blob	A single blob.
detailsPreviewPaneSettings	unit32	Details about preview pane settings.
readingPaneSettings	unit32	Indicates reading pane settings.
navigationPaneVisible	bool	Indicates whether navigation pane is visible or not.

Type: Windows.Data.FileExplorerClassic.FolderOptionGeneralSettings structure

These are settings found in File Explorer->Folder options(...)->General tab

- File explorer can be opened to either Home or This PC
- File explorer can be opened to One Drive folder as well if user has signed in to One Drive (This option is available only if user has signed in)

FolderOptionGeneralSettings Properties

Expand table

Name	Type	Description
defaultOpenLocation	unit8	Indicates whether file explorer can be opened to either home or this PC. Default is home (0).

Name	Type	Description
browseFoldersInNewWindow	bool	Indicates whether to browse folder in new window.
useDoubleClickToOpen	bool	Indicates whether to use double click to open.
showRecentlyUsedFiles	bool	Indicates whether to show recently used files.
showFrequentlyUsedFolders	bool	Indicates whether to show frequently used folders.
showFilesFromOffice	bool	Indicates whether to show files from MS office.

Type: Windows.Data.FileExplorerClassic.FolderOptionsAdvancedSettings structure

These are settings found in File Explorer->Folder options(...)->View tab->Advanced settings

FolderOptionsAdvancedSettings Properties

Expand table

Name	Type	Description
alwaysShowIconsNeverThumbnails	bool	This is used to show icons only.
compactView	bool	Decrease space between items.
displayFileIconOnThumbnails	bool	Indicates whether to display file icon on thumbnails.
displayFileSizeInfoInFolderTips	bool	Indicates whether to display file size information in folder tips.
displayFullFilePath	bool	Show folder contents info tip.
showSysHiddenFiles	bool	Show/hide hidden files and folders.
hideEmptyDrives	bool	Hide drives with no media.
showExtensions	bool	Show/hide extensions of known file types.
hideFolderMergeConflicts	bool	Indicates whether to show/hide folder merge conflicts.
showSuperHiddenOSProtectedFiles	bool	Show/hide OS protected files.
showDriveLetters	bool	Show/hide drive letters.
colorEncryptedCompressedFiles	bool	Show encrypted and compressed files in color.
showPopupDescription	bool	Indicates pop up descriptions.
showSyncProviderNotification	bool	Indicates notification provider sync notification.
useCheckboxesForSelection	bool	Used to show check boxes for selection.
useSharingWizard	bool	Sharing wizard on/off.
navPaneAlwaysShowAvailablityStatus	bool	Show Navigation pane all cloud states.
navPaneShowLibraries	bool	Show navigation pane libraries.
navPaneShowNetwork	bool	Show pane navigation network.
navPaneShowThisPC	bool	This is used to show navigation pane to show current folder.

Type: Windows.Data.FileExplorerClassic.RecycleBinSettings structure

These are settings found in Recycle bin-> right click Properties

RecycleBinSettings Properties

Expand table

Name	Type	Description
displayDeleteConfirmationDialog	bool	Indicates whether to show delete confirmation dialog

Type: Windows.Data.FileExplorerClassic.DesktopIconSettings structure

These are settings related to desktop icons.

DesktopIconSettings Properties

Expand table

Name	Type	Description
viewAutoArrangeIcons	bool	Indicates values for AutoArrangement of icons.
viewAlignIconsToGrid	bool	Indicates values for snapping to grid.
viewShowDesktopIcons	bool	Indicates values to view desktop icons.
sortColPropertyKeyFmtid	wstring	L"B725F130-47EF-101A-A5F1-02608C9EEBAC".
sortColPropertyKeyPid	Unit32	10.
sortColDirection	bool	1=Ascending -1=Descending Default value is Ascending (true).

Input Method Editors (IME)

History files are used to optimize the Japanese IME user experience across devices.

Type: Windows.Data.Input.HistoryFiles structure

This type is multi-instance and must be retrieved using the following collection collection names:

- "historyfiles"

The following is an example command line for retrieving this type:

```
readCloudDataSettings.exe enum -type:Windows.Data.Input.HistoryFiles -collection:historyfiles
```

For more information on retrieving multi-instance settings, see Cloud Data Store Settings Reader Tool.

HistoryFiles Properties

Expand table

Name	Type	Description
InputDataFiles	Map<string,FilePathInfo>	A map of input history FilePathInfo structs where the key represents a relative folder path.

Type: Windows.Data.Input.FilePathInfo structure

FilePathInfo Properties

Expand table

Name	Type	Description
filePath	Map<string,fileData>	A map of FileData structs where the key is the file name.

Type: Windows.Data.Input.FileData structure

FileData Properties

Expand table

Name	Type	Description
filePath	blob	Raw input data, byte array serialized as a collection of integers.

Internet Explorer

Although Internet explorer has reached end of life, some settings stored by the browser may remain in the cloud or on Windows through settings backup.

The **SettingUnit** structure used with the following settings has the following definition:

C++

```
struct SettingUnit
{
    String settingType
    String settingUnitID
    FILETIME timeStamp
    blob settingData
}
```

Type: Windows.Data.InternetExplorer.Favorites structure

Favorites Properties

Expand table

Name	Type	Description
favoriteSettings	vector<**SettingUnit**>	**SettingUnit** structures related to Internet Explorer favorites.

Type: Windows.Data.InternetExplorer.TypedURLS structure

TypedURLS Properties

Expand table

Name	Type	Description
typedURLSettings	vector<**SettingUnit**>	**SettingUnit** structures related to Internet Explorer TypedURLS.

Type: Windows.Data.InternetExplorer.BrowserHistory structure

BrowserHistory Properties

Expand table

Name	Type	Description
BrowserHistory	vector<**SettingUnit**>	**SettingUnit** structures related to Internet Explorer browser history.

Type: Windows.Data.InternetExplorer.AutoComplete structure

AutoComplete Properties

Expand table

Name	Type	Description
AutoCompleteSetting	vector<**SettingUnit**>	**SettingUnit** structures related to Internet Explorer autocomplete.

Type: Windows.Data.InternetExplorer.TabRoaming structure

TabRoaming Properties

Expand table

Name	Type	Description
TabRoamingSetting	vector<**SettingUnit**>	**SettingUnit** structures related to Internet Explorer tab roaming settings.

Lock screen

Setting for managing and personalizing the lock screen.

Type: Windows.Data.LockScreenSettings.LockScreenKind enumeration

LockScreenKind values

Expand table

Name	Value	Description
Picture	0	Picture.
Slideshow	1	Slideshow
Spotlight	2	Spotlight

Type: Windows.Data.LockScreenSettings structure

The scope of this type is per device.

LockScreenSettings Properties

Expand table

Name	Type	Description
kind	**LockScreenKind**	Specifies whether current Lockscreen is set as Wallpaper or Slideshow or Spotlight
pictureOnSignInScreen	bool	Specifies whether show the lock screen background picture on the sign-in screen is enabled.
funItems	bool	Specifies whether "Get fun facts, tips, tricks, and more on your lock screen" is enabled.
itemId	wstring	The unique ID for the lockscreen wallpaper uploaded to during backup.
contentUri	wstring	The url for the lockscreen wallpaper uploaded to during backup
lockScreenStatus	wstring	Not used.
slideShowAutoLock		Not used.
slideShowEnabledOnBattery		Not used.
slideshowOptimizePhotoSelection		Not used.
slideShowIncludeCameraRoll		Not used.
slideShowDuration		Not used.

Name	Type	Description
syncRootRelativePaths		Not used.

Lunar calendar

Settings related to the lunar Calendar in the task bar.

Type: Windows.Data.LunarCalendar structure

LunarCalendar Properties

Expand table

Name	Type	Description
languageType	**LunarCalendarLanguageType**	A member of the **LunarCalendarLanguageType** enumeration. The default value is **Default**.

Type: Windows.Data.LunarCalendarPerDevice structure

This type inherits from **LunarCalendar**. The scope of this type is per device.

Type: Windows.Data.LunarCalendarLanguageType enumeration

LunarCalendarLanguageType Values

Expand table

Name	Value	Description
Default	0	The default lunar calendar configuration.
None	1	No lunar calendar.
SimplifiedChinese	2	The Simplified Chinese lunar calendar.
TraditionalChinese	3	The Traditional Chinese calendar.

Multiple displays

Settings related to multiple displays.

Type: Windows.Data.Settings.DisplaySettings.MultipleDisplays structure

The scope of this type is per device.

MultipleDisplays Properties

Expand table

Name	Type	Description
rememberWindowLocationsPerMonitorConnection	nullable<bool>	Enables remember window locations per monitor connection
minimizeWindowsOnMonitorDisconnect	nullable<bool>	Enables minimize windows on monitor disconnect.
easeCursorMovementBetweenDisplays	nullable<bool>	Enables minimize windows on monitor disconnect.

Nightlight

Settings related to changing screen lighting or set brightness for certain hours.

Type: Windows.Data.BlueLightReduction.ScheduleTime structure

The scope of this type is per device.

ScheduleTime Properties

Expand table

Name	Type	Description
hour	int8	Hours.
minute	int8	Minutes.

Type: Windows.Data.BlueLightReduction.Settings structure

BlueLightReduction.Settings Properties

Expand table

Name	Type	Description
automaticOnSchedule	bool	Specifies whether blue light reduction is automatically turned on or off based on a schedule.
automaticOnSunset	bool	Specifies if blue light reduction schedule is automatically set based on sunrise and sunset.
manualScheduleBlueLightReductionOnTime	**ScheduleTime**	The start time of blue light reduction for a user manually setting their schedule.
manualScheduleBlueLightReductionOffTime	**ScheduleTime**	The end time of blue light reduction for a user manually setting their schedule.
targetColorTemperature	**int16**	The target color temperature (in Kelvin) for blue light reduction.
sunsetTime	**ScheduleTime**	The scheduled sunset time for blue light reduction.
sunriseTime	**ScheduleTime**	he scheduled sunrise time for blue light reduction.
previewColorTemperatureChanges	bool	Specifies whether blue light reduction color temperature changes should be previewed.
darkMode	bool	Specifies whether app mode should change when blue light reduction is turned on or off

Type: Windows.Data.BlueLightReduction.ActiveState enumeration

ActiveState values

Expand table

Name	Value	Description
BlueLightReductionOn	0	
BlueLightReductionOff	1	

Type: Windows.Data.BlueLightReduction.ChangeSource enumeration

ChangeSource values

Expand table

Name	Value	Description
Schedule	0	
User	1	

Type: Windows.Data.BlueLightReduction.BlueLightReductionState structure

BlueLightReductionState Properties

Expand table

Name	Type	Description
state	**ActiveState**	The current state of blue light reduction.
source	**ChangeSource**	Where the change came from, user change or scheduled change.
timestampUTC	int64	The time the change in active state was applied.
isSupported	bool	Whether or not current configuration supports blue light reduction.

NlmSignature

Settings related to Nlm signatures. Each network is uniquely identified with a network signature based on the uniquely identifiable properties of that network.

Type: Windows.Data.Nlm.NlmSignature structure

This type is multi-instance and must be retrieved using the following collection names:

- "wificloudstore3"
- "wifi3_wpa3"
- "wifi3_owe"

The following is an example command line for retrieving this type:

```
readCloudDataSettings.exe enum -type:windows.data.nlm.nlmsignature -collection:wificloudstore3
```

For more information on retrieving multi-instance settings, see Cloud Data Store Settings Reader Tool.

NlmSignature Properties

Expand table

Name	Type	Description
category	uint64	Category of signature.

Pen and Windows Ink

Settings related to pen and Windows Ink.

Type: Windows.Data.Input.Devices.PenPerDevice structure

PenPerDevice Properties

Expand table

Name	Type	Description
singleClickOverride	uint64	Any integer value 0 - 13. Value corresponds to options available for Single Click setting dropdown.
singleClickPenWorkspaceVerb	uint64	Any integer value 0-3. Value corresponds to options available for Single Click setting dropdown.
doubleClickOverride	uint64	Any integer value 0 - 13. Value corresponds to options available for Double Click setting dropdown.
doubleClickPenWorkspaceVerb	uint64	Any integer value 0-3. Value corresponds to options available for Double Click setting dropdown.
longPressOverride	uint64	Any integer value 0 - 13. Value corresponds to options available for long press setting dropdown.
longPressPenWorkspaceVerb	uint64	Any integer value 0-3. Value corresponds to options available for Long Press setting dropdown.
penWorkspaceAppLaunchOnPenDetachEnabled	bool	Specifies whether the pen menu is shown after pen is removed from storage.
penEnablePenButtonOverride	bool	Specifies whether apps are allowed to override the shortcut button behavior.
singleClickCustomAppPath	wstring	Specifies the path to the app opened on single-click.
doubleClickCustomAppPath	wstring	Specifies the path to the app opened on double-click.
longPressCustomAppPath	wstring	Specifies the path to the app opened on long-press.
singleClickCustomAppID	wstring	Specifies the app ID app opened on single-click.
doubleClickCustomAppID	wstring	Specifies the app ID app opened on double-click.
longPressCustomAppID	wstring	Specifies the app ID app opened on long-press.

Supported override and verb values:

Expand table

Dropdown Option Selected	Override	PenWorkspaceVerb	Override (Win10)	PenWorkspaceVerb (Win10)
Nothing	1	0/1	1	0
Screen Snipping	8	0/1	8	0
Pen Menu	5	0	5	0
Whiteboard	9	0	5	2
One Note	6	0/1	0	0
One Note Quick Notes	13	0/1	NA	NA
Sticky Notes	5	1	5	1
Snip & Sketch	NA	NA	5	3
Open Program	3	0	3	0
Open App	2	0	2	0

Personalization - colors

Settings related to system colors.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Themes\Personalize

Expand table

Registry value	Type	Data	Description
AppsUseLightTheme	REG_DWORD	0/1	Signifies light/dark color mode for an app.
SystemUsesLightTheme	REG_DWORD	0/1	Signifies light/dark color mode for Windows.
EnableTransparency	REG_DWORD	0/1	Signifies the transparency effect on windows and surfaces.
ColorPrevalance	REG_DWORD	0/1	Signifies the toggle state for "Show Accent Color on Start and Taskbar".

Registry values under HKCU\Control Panel\Desktop

Expand table

Registry value	Type	Data	Description
AutoColorization	REG_DWORD	0/1	signifies auto-apply accent color based on background or manually.

Registry values under HKCU\Software\Microsoft\Windows\DWM

Expand table

Registry value	Type	Data	Description
ColorPrevalence	REG_DWORD	0/1	Signifies toggle state for "show Accent color in title bar and windows borders".

Personalization - Start - Folders

Specifies the folders that are shown at the bottom of the Start menu.

Registry values under HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Start

Expand table

Registry value	Type	Data	Description
VisiblePlaces	REG_BINARY	A vector of GUIDs.	A list of GUIDs indicating the folders that are shown at the bottom of Start.

The folder GUIDs that are supported for *Start_Layout* are:

- Documents: {2D34D5CE-FA5A-4543-82F2-22E6EAF7773C}
- Downloads: {E367B32F-89DE-4355-BFCE-61F37B18A937}
- Music: {B00B0620-7F51-4C32-AA1E-34CC547F7315}
- Pictures: {383F07A0-E80A-4C80-B05A-86DB845DBC4D}
- Videos: {42B3A5C5-7D86-42F4-80A4-93FACA7A88B5}
- Network: {FE758144-080D-42AE-8BDA-34ED97B66394}
- UserProfile: {74BDB04A-F94A-4F68-8BD6-4398071DA8BC}
- Explorer: {148A24BC-D60C-4289-A080-6ED9BBA24882}
- Settings: {52730886-51AA-4243-9F7B-2776584659D4}

Personalization - Start - Layout

Specifies the start layout.

Registry values for layout under HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Start

Expand table

Registry value	Type	Data	Description
Config	REG_BINARY	A JSON file.	The contents of the Pinned Layout in JSON.

Personalization - Start - Show recent apps

Specifies whether apps that were recently installed in Start in various surfaces are shown on Start.

Registry values for recent apps under HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Start

Expand table

Registry value	Type	Data	Description
ShowRecentList	REG_BOOL	0 or 1	Specifies whether apps that were recently installed in Start in various surfaces are shown on Start.

Personalization - Start - Show recommended apps

Specifies whether recommended files in Start, recent files in File Explorer, and items in Jump Lists are shown.

Registry values under HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Ad

Expand table

Registry value	Type	Data	Description
Start_TrackDocs	REG_BOOL	0 or 1	Specifies whether recommended files in Start, recent files in File Explorer, and items in Jump Lists are shown.

Personalization - Taskbar - Badges

This setting enables badges for apps on the taskbar.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\TaskbarE

Expand table

Registry value	Type	Data	Description
SystemSettings_Taskbar_Badging	REG_SZ	0 or 1	Enables badges for apps on the taskbar.
SystemSettings_DesktopTaskbar_Badging	REG_SZ	0 or 1	Enables badges for apps on the taskbar.

Personalization - Taskbar - Combine buttons, mult-monitor

This setting enables combining buttons and hiding labels on the taskbar on multiple monitors.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\MMTask

Expand table

Registry value	Type	Data	Description
SystemSettings_DesktopTaskbar_GroupingMode	REG_SZ	0, 1, or 2	0: Always, 1: When taskbar is full, 2: Never.

Personalization - Taskbar - Multi-monitor

Enables showing the taskbar on multiple displays.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\MMTask

Expand table

Registry value	Type	Data	Description
SystemSettings_Taskbar_MultiMon	REG_SZ	0 or 1	Enables showing the taskbar on multiple displays.
SystemSettings_DesktopTaskbar_MultiMon	REG_SZ	0 or 1	Enables showing the taskbar on multiple displays.

Personalization - Taskbar - Multi-monitor taskbar mode

Specifies the behavior of the taskbar when displayed on multiple monitors.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced\MMTask

Expand table

Registry value	Type	Data	Description
SystemSettings_Taskbar_MultiMonTaskbarMode	REG_SZ	0, 1, or 2	0: Duplicate, 1: Primary and monitor window is on, 2: Monitor window is on.
SystemSettings_DesktopTaskbar_MultiMonTaskbarMode	REG_SZ	0 1	0: Duplicate, 1: Primary and monitor window is on, 2: Monitor window is on.

Personalization - Taskbar - Pinned apps from other devices

Specifies the set of apps pinned to the taskbar from another device.

Registry values under HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Taskband\Favorites

Expand table

Registry value	Type	Data	Description
FavoritesMigration	REG_BLOB	A binary blob.	This is an opaque binary blob copied from the following location on the backed up.
Favorites	REG_SZ	0 1	The format of this key is undocumented.

Secondary accounts

Provides information about Microsoft accounts (MSA) and work or school accounts added to the device to sign in to apps or online services, in addition to the account used to log on to the device. On Windows 11, backup and restore of this setting is supported. On Windows 10, backup is supported but restore is not.

Type: Windows.Data.Account.AccountType enumeration

AccountType values

Expand table

Name	Value	Description
MSA	0	Microsoft account.
AAD	1	Azure Active Directory account,
Others	2	Oher account.

Type: Windows.Data.Account.AccountInfo structure

AccountInfo Properties

Expand table

Name	Type	Description
accountName	wstring	The user name of the account such as "example@outlook.com".
accountId	wstring	The unique identifier of the account.
accountType	**AccountType**	The type of the account.
country	wstring	The code of the country or region in which a MSA is registered.
safeCustomerId	wstring	An alternative identifier for an MSA.
ageGroup	wstring	Age group of an MSA, based on the registered birth date of the MSA user. Current values are 0 = unknown, 1 = child, 2 = teen, 3 = adult.
scope	wstring	Represents the "Sign in options" setting state.

Type: Windows.Data.Account.SecondaryAccounts structure

SecondaryAccounts Properties

Expand table

Name	Type	Description
accountDetails	vector<AccountInfo>	A vector of **AccountInfo** objects representing secondary accounts.

Spelling dictionary

The user's custom spelling dictionary is stored in a file in the following file path:

`%userprofile%\AppData\Roaming\Microsoft\Spelling\neutral\default.dic`

USB

This setting controls toggles such as connection notifications, battery saver and other notifications related to charging of PC.

Registry values under HKCU\Software\Microsoft\Shell\USB

Expand table

Registry value	Type	Data	Description
NotifyOnUsbErrors	REG_DWORD	0 or 1	Enables notifications for USB errors.
NotifyOnWeakCharger	REG_DWORD	0 or 1	Enables notifications for weak charger detected.

Windows Backup - Remember my preferences

Helps user to synchronize the settings and Data on multiple machines.

Type: Windows.Data.WindowsBackup.Preference.CloudStorePolicyEnum enumeration

CloudStorePolicyEnum values

Expand table

Name	Value	Description
SyncEnabled	0	Sync enabled.
SyncDisabledByUser	1	Sync disabled by user.
SyncDisabledByGroupPolicy	2	Sync disabled by group policy.

Toggles properties

Expand table

Name	Type	Description
toggleState	map<wstring, **CloudStorePolicyEnum**>	The user preference in terms of mapping of sync policy and CloudStorePolicy option. Allowed values for sync policy names are listed below.

Valid sync policy names:

- Microsoft
- MicrosoftDevice
- MicrosoftUserProfile
- Microsoft.Accessibility
- MicrosoftDevice.Accessibility
- MicrosoftUserProfile.Accessibility
- Microsoft.Credentials
- MicrosoftDevice.Credentials
- MicrosoftUserProfile.Credentials
- Microsoft.Personalization
- MicrosoftDevice.Personalization
- MicrosoftUserProfile.Personalization
- Microsoft.Language
- MicrosoftDevice.Language
- MicrosoftUserProfile.Language
- Microsoft.Default
- MicrosoftDevice.Default
- MicrosoftUserProfile.Default
- Microsoft.Ink
- MicrosoftDevice.Ink
- MicrosoftUserProfile.Ink

Windows Update

Settings related to Windows Update.

Registry values under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsUpdate\UX\Settings

Expand table

Registry value	Type	Data	Description
IsContinuousInnovationOptedIn	REG_DWORD	0/1	Enables devices to get the latest updates as soon as they're released.
AllowMUUpdateService	REG_DWORD	0/1	Allows users to get other Microsoft products alongside with Windows Updates.
IsExpedited	REG_DWORD	0/1	User selects this to invoke device restart 15 min after all updates finished installing.
RestartNotificationsAllowed2	REG_DWORD	0/1	Users decides if they want to be notified about the updates pending restart.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Cloud Data Store Settings Reader tool (readCloudDataSettings.exe)

Article • 03/06/2024

This article describes the Cloud Data Store Settings Reader Tool, readCloudDataSettings.exe, that can be used to fetch data stored within the Windows Cloud Data Store component on the local device. For settings documented in Reference for Windows 11 settings or Reference for Windows 11 and Windows 10 settings, if the setting lists a type name rather than a registry key, then you must use this tool to retrieve the data.

Usage

Single instance items

```
readCloudDataSettings.exe get -type:<type name> [-account:<secondary account id>]
```

Multi-instance items

```
readCloudDataSettings.exe enum -type:<type name> [-collection:<collection name>] [-account:<secondary account id>]
```

Command line parameter descriptions

Expand table

Parameter	Description
`<type name>`	The name of a Cloud Data Store type whose data is to be retrieved (e.g., "windows.data.platform.diagnostics.diagnosticdata")
`<collection name>`	The optional name of a collection for a Cloud Data Store multi-instance type. This must be specified if the multi-instance type has a named collection and must not be specified if either the collection has no name or the type is single-instance. Cloud Data Store does has no support for enumerating either the data or names of all collections of a type.
`<secondary account id>`	The optional id (in the form of user@domain) of a secondary account associated with the current user whose data is to be fetched. This must be a secondary

Parameter	Description
	account associated with the currently logged in Windows user; it does not provide access to data for other Windows users that might be sharing the device.

Errors

If the data does not exist or an error occurs, the output will report a pair of square brackets with nothing between; example:

PowerShell

```
[
]
```

Examples

Single-instance type

Command line:

PowerShell

```
C:\Windows\System32>readCloudDataSettings.exe get -
type:windows.data.settings.settingsusagehistory
```

Output:

PowerShell

```
/type: windows.data.settings.settingsusagehistory

[
{"Data":{"pageUsages":{"\"SettingsPageAppsNotifications\"":{"Data":
{"frequency":1,"lastUsedTime":1.3353819570909629E+17,"lastUsedSettingId":"Sy
stemSettings_Notifications_QuietHours_MuteNotification_Enabled"}},"\"Setting
sPageOtherUsers\"":{"Data":
{"frequency":1,"lastUsedTime":1.3353709031552022E+17,"lastUsedSettingId":"Sy
stemSettings.UserAccountsHandlers.RemoveOtherUserSetting"}}}}}
]
```

Single-instance type, alternate account

Command line:

```PowerShell
C:\Windows\System32>readCloudDataSettings.exe get -
type:windows.data.settings.settingsusagehistory -
account:otheruser@contoso.com
```

Output:

```PowerShell
/type: windows.data.settings.settingsusagehistory

[
{"Data":{"pageUsages":{"\"SettingsPageAppsNotifications\"":{"Data":
{"frequency":1,"lastUsedTime":1.3353819570909629E+17,"lastUsedSettingId":"Sy
stemSettings_Notifications_QuietHours_MuteNotification_Enabled"}},"\"Setting
sPageOtherUsers\"":{"Data":
{"frequency":1,"lastUsedTime":1.3353709031552022E+17,"lastUsedSettingId":"Sy
stemSettings.UserAccountsHandlers.RemoveOtherUserSetting"}}}}}
]
```

Multi-instance type with collection name

Command line:

```PowerShell
readclouddatasettings.exe enum -type:windows.data.wifi.wifiprofile -
collection:wificloudstore3
```

Output:

```PowerShell
/type: windows.data.wifi.wifiprofile
/collection: wificloudstore3

[
    {"Data":{"profileXml":"<?xml version=\"1.0\"?>\r\n<WLANProfile
xmlns=\"http://www.microsoft.com/networking/WLAN/profile/v1\">\r\n\t<name>My
SpectrumWiFi98-
5G</name>\r\n\t<SSIDConfig>\r\n\t\t<SSID>\r\n\t\t\t<hex>4D79537065637472756D
5769466939382D3547</hex>\r\n\t\t\t<name>MySpectrumWiFi98-
5G</name>\r\n\t\t</SSID>\r\n\t</SSIDConfig>\r\n\t<connectionType>ESS</connec
```

```
tionType>\r\n\t<connectionMode>auto</connectionMode>\r\n\t<MSM>\r\n\t\t<secu
rity>\r\n\t\t\t<authEncryption>\r\n\t\t\t\t<authentication>WPA2PSK</authenti
cation>\r\n\t\t\t\t<encryption>AES</encryption>\r\n\t\t\t\t<useOneX>false</u
seOneX>\r\n\t\t\t</authEncryption>\r\n\t\t\t<sharedKey>\r\n\t\t\t\t<keyType>
passPhrase</keyType>\r\n\t\t\t\t<protected>false</protected>\r\n\t\t\t\t<key
Material>ancientzebra274</keyMaterial>\r\n\t\t\t</sharedKey>\r\n\t\t</securi
ty>\r\n\t</MSM>\r\n\t<MacRandomization
xmlns=\"http://www.microsoft.com/networking/WLAN/profile/v3\">\r\n\t\t<enabl
eRandomization>false</enableRandomization>\r\n\t\t<randomizationSeed>1865639
973</randomizationSeed>\r\n\t</MacRandomization>\r\n</WLANProfile>\r\n","las
tModifiedTime":1.3354054522984058E+17}},
    {"Data":{"profileXml":"<?xml version=\"1.0\"?>\r\n<WLANProfile
xmlns=\"http://www.microsoft.com/networking/WLAN/profile/v1\">\r\n\t<name>KI
A Service
Color</name>\r\n\t<SSIDConfig>\r\n\t\t<SSID>\r\n\t\t\t<hex>4B494120536572766
9636520436F6C6F72</hex>\r\n\t\t\t<name>KIA Service
Color</name>\r\n\t\t</SSID>\r\n\t</SSIDConfig>\r\n\t<connectionType>ESS</con
nectionType>\r\n\t<connectionMode>manual</connectionMode>\r\n\t<MSM>\r\n\t\t
<security>\r\n\t\t\t<authEncryption>\r\n\t\t\t\t<authentication>WPA2PSK</aut
hentication>\r\n\t\t\t\t<encryption>AES</encryption>\r\n\t\t\t\t<useOneX>fal
se</useOneX>\r\n\t\t</authEncryption>\r\n\t\t\t<sharedKey>\r\n\t\t\t\t<key
Type>passPhrase</keyType>\r\n\t\t\t\t<protected>false</protected>\r\n\t\t\t\
t<keyMaterial>4258270521</keyMaterial>\r\n\t\t\t</sharedKey>\r\n\t\t</securi
ty>\r\n\t</MSM>\r\n\t<MacRandomization
xmlns=\"http://www.microsoft.com/networking/WLAN/profile/v3\">\r\n\t\t<enabl
eRandomization>false</enableRandomization>\r\n\t\t<randomizationSeed>4088426
234</randomizationSeed>\r\n\t</MacRandomization>\r\n</WLANProfile>\r\n","las
tModifiedTime":1.3354054522999686E+17}}
]
```


Note on serialization

For interoperability settings data structures are serialized to JSON when exported from Windows. In some cases, this can result in unintuitive results. See the examples below.

Basic data types

Basic data types (integer, floating point, string, etc) have no special representation beyond normal JSON encoding.

Structures

All structures (including top level structures) are wrapped in a JSON element called "Data". This includes nested structures (including when structures are used in vectors, maps, and nullables)

Example

This structure definition:

C#

```
struct MyInnerDataType
{
    0: int32 data;
}

struct MyDataType
{
    0: MyInnerDataType innerData;
    1: int64 id;
}
```

Becomes the following JSON:

JSON

```
{
    "Data":
    {
        "innerData":
        {
            "Data":
            {
                "data": 1
            }
        },
        "id": 2
    }
}
```

Vectors / Lists

Vectors and lists are translated into standard JSON arrays (with the caveat from above that a list of structures has each element wrapped in a JSON element named "Data").

Example

This structure definition:

C#

```
struct MyInnerDataType
{
    0: int32 data;
}

struct MyDataType
{
    0: vector<MyInnerDataType> dataList;
    1: vector<int32> idList;
}
```

Becomes the following JSON:

JSON

```
{
    "Data":
    {
        "dataList":
        [
            {
                "Data":
                {
                    "data": 1
                }
            },
            {
                "Data":
                {
                    "data": 2
                }
            },
        ],
        "idList": [ 1, 2 ]
    }
}
```

Blob

Blobs are translated into arrays of integers

Example

This structure definition:

C#

```
Struct mySetting
{
0: blob settingData;
}
```

Becomes the following JSON:

JSON


```
{
    "Data":
    {
        "settingData":
[-103,84,-51,60,-88,-121,16,75,-94,21,96,-120,-120,-35,59,85,4,0,0,0,0,1,0,0
,36,0,0,0,73,0,110,0,116,0,101,0,114,0,110,0,101,0,116,0,32,0,69,0,120,0,112
,0,108,0,111,0,114,0,101,0,114,0,0,0,1,0,0,0,2,0,0,0,7,0,0,0,7,0,0,0,58,0,0,
0,104,0,116,0,116,0,112,0,115,0,58,0,47,0,47,0,103,0,108,0,111,0,98,0,97,0,1
08,0,46,0,115,0,116,0,115,0,46,0,109,0,115,0,102,0,116,0,46,0,110,0,101,0,11
6,0,47,0,0,0,2,0,0,0,2,0,0,0,7,0,0,0,7,0,0,0,34,0,0,0,109,0,105,0,99,0,114,0
,105,0,100,0,101,0,114,0,64,0,103,0,109,0,101,0,46,0,103,0,98,0,108,0,0,0,3,
0,0,0,0,0,0,0,7,0,0,0,7,0,0,0,34,0,0,0,110,0,97,0,109,0,105,0,56,0,48,0,114,
0,117,0,108,0,101,0,115,0,58,0,68,0,103,0,109,0,101,0,0,0,1,0,0,0,100,0,0,0,
0,0,0,0,8,0,0,0,8,0,0,0,16,0,0,0,-43,-74,60,78,86,37,-40,76,-92,-115,-57,85,
-57,55,-53,-90,5,0,0,0,0,0,0,0,8,0,0,0,0,0,0,0,1,0,0,0,7,0,0,0,0,0,0,0,2,0,0
,0,7,0,0,0,0,0,0,0,3,0,0,0,7,0,0,0,0,0,0,0,4,0,0,0,6,0,0,0,-58,-104,104,-79,
72,-95,103,73,-111,113,100,-41,85,-38,-123,32]
    }
}
```


Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Devices and sensors

Article • 10/22/2021

This article provides an index of development features that are related to scenarios involving devices and sensors in Windows apps.

> **Note**
>
> The **Windows App SDK** currently does not provide APIs related to devices and sensors scenarios.

Windows OS features

Windows 10 and later OS releases provide a wide variety of APIs related to data and files scenarios for apps. These features are available via a combination of WinRT and Win32 (C++ and COM) APIs provided by the Windows SDK.

WinRT APIs

The following articles provide information about features available via WinRT APIs provided by the Windows SDK.

Article	Description
Devices, sensors, and power	Learn about the breadth of features available for integrating external devices and sensors into your app.
Enumerate devices	Learn how to find devices that are internally connected to the system, externally connected, or detectable over wireless or networking protocols.
Sensors	Learn how to access sensors such as accelerometers, compasses, and gyrometers.
Bluetooth	Learn how to integrate classic Bluetooth (RFCOMM) and Bluetooth Low Energy (LE) into your apps.
Printing and scanning	Learn how to print and scan from your app.
3D printing	Learn how to add 3D printing functionality to your app.

Article	Description
Get battery information	Learn how to get detailed battery information, including the charge, capacity, and status of a battery or aggregate of batteries.

Win32 (C++ and COM) APIs

The following articles provide information about features available via Win32 (C++ and COM) APIs provided by the Windows SDK.

Article	Description
Sensor API	Learn how to access sensor data using Win32 APIs.
Printing	Learn how to print documents using Win32 APIs.
Power management	Learn how to access power management features using Win32 APIs.

.NET features

The .NET SDK also provides APIs related to device scenarios for WPF and Windows Forms apps.

Article	Description
Printing and print system management (WPF)	Learn how to print documents using .NET APIs for WPF apps.
Print support (Windows Forms)	Learn how to print documents using .NET APIs for Windows Forms apps.
Power management in Windows Forms	Learn how to access power management features using .NET APIs for Windows Forms apps.

Printing and scanning

Article • 09/11/2024

This section describes how to print and scan from your Windows app.

Expand table

Topic	Description
Print from your app	Learn how to print documents from a Windows app. This topic also shows how to print specific pages.
Customize the print preview UI	This section describes how to customize the print options and settings in the print preview UI.
Scan from your app	Learn here how to scan content from your app by using a flatbed, feeder, or auto-configured scan source.

Related topics

- UWP print sample
- Customize the print workflow

Print from your app

Article • 09/11/2024

This topic describes how to print from a Windows app.

For more advanced features, see Customize the print preview UI.

✓ **Important APIs**: Windows.Graphics.Printing namespace, PrintManager class, PrintTask, Microsoft.UI.Xaml.Printing namespace, PrintDocument class

Register for printing

The first step to add printing to your app is to register for printing by getting the PrintManager object for the current window. The PrintManager class is responsible for orchestrating the printing flow for your app. To use this class, you must first call the method that returns the PrintManager object that is specific to the current active window.

- In a non-UWP app, use the PrintManagerInterop.GetForWindow method.
- In a UWP app, use the PrintManager.GetForCurrentView method.

Your app must do this on every screen from which you want your user to be able to print. Only the screen that is displayed to the user can be registered for printing. If one screen of your app has registered for printing, it must unregister for printing when it exits. If it is replaced by another screen, the next screen must register for printing when it opens.

> **Tip**
>
> If you need to support printing from more than one page in your app, you can put this print code in a common helper class and have your app pages reuse it. For an example of how to do this, see the `PrintHelper` class in the **UWP print sample**.

After a user has initiated printing, you use a PrintDocument to prepare the pages to be sent to the printer.The `PrintDocument` type is in the **Microsoft.UI.Xaml.Printing** namespace along with other types that support preparing XAML content for printing.

The PrintDocument class is used to handle much of the interaction between the app and the PrintManager, but it exposes several callbacks of its own. During registration, create instances of `PrintManager` and `PrintDocument` and register handlers for their printing events.

In this example, registration is performed in the `RegisterForPrinting` method, which is called from the page's Loaded event handler.

C#

```
using Microsoft.UI.Xaml.Printing;
using Windows.Graphics.Printing;

PrintDocument printDocument = null;
IPrintDocumentSource printDocumentSource = null;
List<UIElement> printPreviewPages = new List<UIElement>();

private void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    RegisterForPrinting();
}

private void RegisterForPrinting()
{
    var hWnd = WinRT.Interop.WindowNative.GetWindowHandle(App.MainWindow);
    PrintManager printManager = PrintManagerInterop.GetForWindow(hWnd);
    printManager.PrintTaskRequested += PrintTask_Requested;

    printDocument = new PrintDocument();
    printDocumentSource = printDocument.DocumentSource;
    printDocument.Paginate += PrintDocument_Paginate;
    printDocument.GetPreviewPage += PrintDocument_GetPreviewPage;
    printDocument.AddPages += PrintDocument_AddPages;
}
```

> **⚠ Warning**
>
> In UWP printing examples, it's recommended to register for printing from the OnNavigatedTo method override. In non-UWP apps, you need to use the window handle in the PrintManagerInterop.GetForWindow call, so you should use the Loaded event to ensure that the window handle is not `null`, which might be the case in OnNavigatedTo.

Here, the event handlers are unregistered in the `UnregisterForPrinting` method, which is called from the OnNavigatedFrom method.

C#

```
protected override void OnNavigatedFrom(NavigationEventArgs e)
{
    base.OnNavigatedFrom(e);
    UnregisterForPrinting();
}
```

```
private void UnregisterForPrinting()
{
    if (printDocument == null)
    {
        return;
    }

    printDocument.Paginate -= PrintDocument_Paginate;
    printDocument.GetPreviewPage -= PrintDocument_GetPreviewPage;
    printDocument.AddPages -= PrintDocument_AddPages;

    var hWnd = WinRT.Interop.WindowNative.GetWindowHandle(App.MainWindow);
    PrintManager printManager = PrintManagerInterop.GetForWindow(hWnd);
    printManager.PrintTaskRequested -= PrintTask_Requested;
}
```

> **Note**
>
> If you have a multiple-page app and don't disconnect printing, an exception is thrown when the user leaves the page and then returns to it.

Create a print button

Add a print button to your app's screen where you'd like it to appear. Make sure that it doesn't interfere with the content that you want to print.

XAML

```
<Button x:Name="InvokePrintingButton"
        Content="Print"
        Click="InvokePrintingButton_Click"/>
```

In the Button's Click event handler, show the Windows print UI to the user.

- In a non-UWP app, use the PrintManagerInterop.ShowPrintUIForWindowAsync method.
- In a UWP app, use the PrintManager.ShowPrintUIAsync method.

C#

```
var hWnd = WinRT.Interop.WindowNative.GetWindowHandle(App.MainWindow);
await PrintManagerInterop.ShowPrintUIForWindowAsync(hWnd);
```

This method is an asynchronous method that displays the appropriate printing window, so you'll need to add the async keyword to the Click handler. We recommend calling the IsSupported method first in order to check that the app is being run on a device that supports printing (and handle the case in which it is not). If printing can't be performed at that time for any other reason, the method will throw an exception. We recommend catching these exceptions and letting the user know when printing can't proceed.

In this example, a print window is displayed in the event handler for a button click. If the method throws an exception (because printing can't be performed at that time), a ContentDialog control informs the user of the situation.

C#

```
private async void InvokePrintingButton_Click(object sender, RoutedEventArgs
e)
{
    if (PrintManager.IsSupported())
    {
        try
        {
            // Show system print UI.
            var hWnd =
WinRT.Interop.WindowNative.GetWindowHandle(App.MainWindow);
            await
Windows.Graphics.Printing.PrintManagerInterop.ShowPrintUIForWindowAsync(hWnd
);
        }
        catch
        {
            // Printing cannot proceed at this time.
            ContentDialog noPrintingDialog = new ContentDialog()
            {
                Title = "Printing error",
                Content = "\nSorry, printing can' t proceed at this time.",
                PrimaryButtonText = "OK"
            };
            await noPrintingDialog.ShowAsync();
        }
    }
    else
    {
        // Printing is not supported on this device.
        ContentDialog noPrintingDialog = new ContentDialog()
        {
            Title = "Printing not supported",
            Content = "\nSorry, printing is not supported on this device.",
            PrimaryButtonText = "OK"
        };
        await noPrintingDialog.ShowAsync();
```

```
    }
}
```

Format your app's content

When `ShowPrintUIForWindowAsync` is called, the PrintTaskRequested event is raised. In the `PrintTaskRequested` event handler, you create a PrintTask by calling the PrintTaskRequest.CreatePrintTask method. Pass the title for the print page and the name of a PrintTaskSourceRequestedHandler delegate. The title is shown in the print preview UI. The `PrintTaskSourceRequestedHandler` links the `PrintTask` with the `PrintDocument` that will provide the content.

In this example, a completion handler is also defined to catch errors. It's a good idea to handle completion events because then your app can let the user know if an error occurred and provide possible solutions. Likewise, your app could use the completion event to indicate subsequent steps for the user to take after the print job is successful.

C#

```
private void PrintTask_Requested(PrintManager sender,
PrintTaskRequestedEventArgs args)
{
    // Create the PrintTask.
    // Defines the title and delegate for PrintTaskSourceRequested.
    PrintTask printTask = args.Request.CreatePrintTask("WinUI 3 Printing
example", PrintTaskSourceRequested);

    // Handle PrintTask.Completed to catch failed print jobs.
    printTask.Completed += PrintTask_Completed;

    DispatcherQueue.TryEnqueue(DispatcherQueuePriority.Normal, () =>
    {
        InvokePrintingButton.IsEnabled = false;
    });
}

private void PrintTaskSourceRequested(PrintTaskSourceRequestedArgs args)
{
    // Set the document source.
    args.SetSource(printDocumentSource);
}

private void PrintTask_Completed(PrintTask sender,
PrintTaskCompletedEventArgs args)
{
    string StatusBlockText = string.Empty;

    // Notify the user if the print operation fails.
```

```
    if (args.Completion == PrintTaskCompletion.Failed)
    {
        StatusBlockText = "Failed to print.";
    }
    else if (args.Completion == PrintTaskCompletion.Canceled)
    {
        StatusBlockText = "Printing canceled.";
    }
    else
    {
        StatusBlockText = "Printing completed.";
    }


    DispatcherQueue.TryEnqueue(DispatcherQueuePriority.Normal, () =>
    {
        StatusBlock.Text = StatusBlockText;
        InvokePrintingButton.IsEnabled = true;
    });
}
```

After the print task is created, the PrintManager requests a collection of print pages to show in the print preview UI by raising the Paginate event. (This corresponds with the `Paginate` method of the `IPrintPreviewPageCollection` interface.) The event handler you created during registration is called at this time.

> **ⓘ Important**
>
> If the user changes print settings, the paginate event handler will be called again to allow you to reflow the content. For the best user experience, we recommend checking the settings before you reflow the content and avoid reinitializing the paginated content when it's not necessary.

In the Paginate event handler, create the pages to show in the print preview UI and to send to the printer. The code you use to prepare your app's content for printing is specific to your app and the content you print.

This example shows the basic steps to create a single print page that prints an image and caption from the page shown on the screen.

- Create a list to hold the UI elements (pages) to be printed.
- Clear the list of preview pages so that pages aren't duplicated each time pagination occurs.
- Use the PrintPageDescription to get the size of the printer page.
- Format your XAML content to fit the printer page. Each page to be printed is a XAML UI element (typically a container element that contains other content). In

this example, elements are created in code and use the same data as the elements shown on the screen.

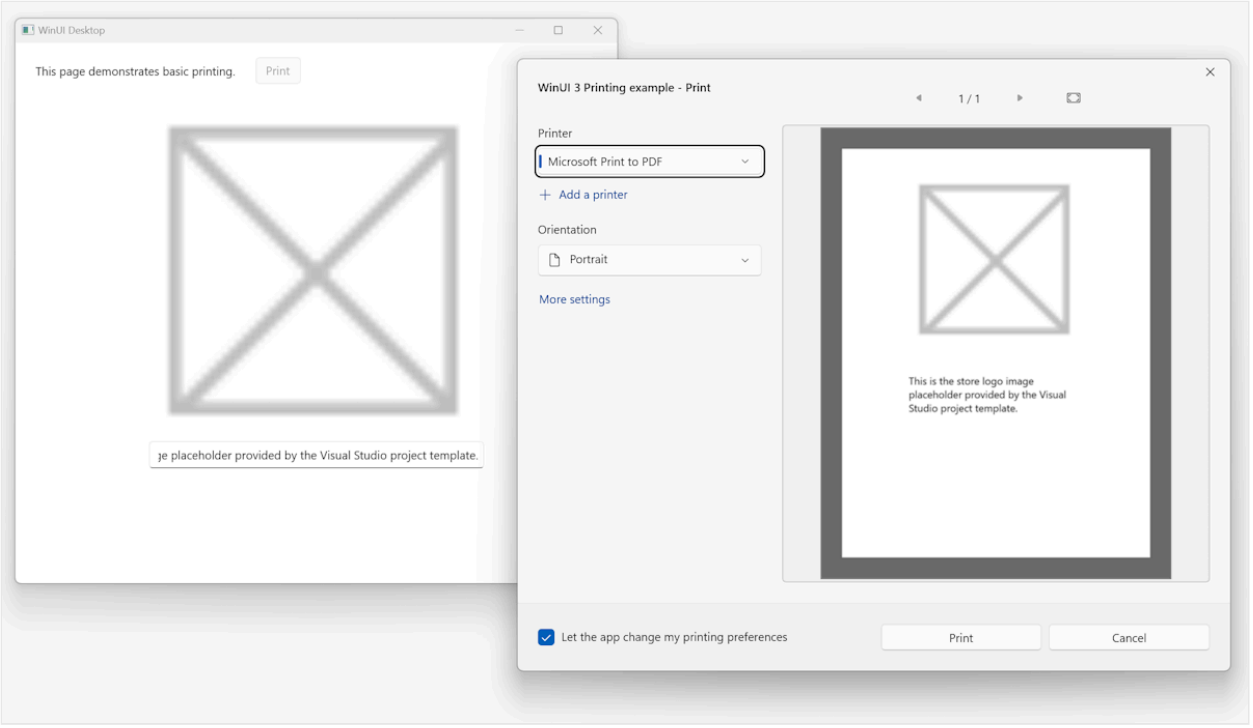
- Flow the content onto additional pages as needed. Multiple pages are not shown in this basic example, but dividing the content into pages is an important part of the Paginate event.
- Add each page to the list of pages to print.
- Set the count of preview pages on the PrintDocument.

C#

```
List<UIElement> printPreviewPages = new List<UIElement>();

private void PrintDocument_Paginate(object sender, PaginateEventArgs e)
{
    // Clear the cache of preview pages.
    printPreviewPages.Clear();

    // Get the PrintTaskOptions.
    PrintTaskOptions printingOptions =
((PrintTaskOptions)e.PrintTaskOptions);
    // Get the page description to determine the size of the print page.
    PrintPageDescription pageDescription =
printingOptions.GetPageDescription(0);

    // Create the print layout.
    StackPanel printLayout = new StackPanel();
    printLayout.Width = pageDescription.PageSize.Width;
    printLayout.Height = pageDescription.PageSize.Height;
    printLayout.BorderBrush = new
Microsoft.UI.Xaml.Media.SolidColorBrush(Microsoft.UI.Colors.Black);
    printLayout.BorderThickness = new Thickness(48);

    Image printImage = new Image();
    printImage.Source = printContent.Source;

    printImage.Width = pageDescription.PageSize.Width / 2;
    printImage.Height = pageDescription.PageSize.Height / 2;

    TextBlock imageDescriptionText = new TextBlock();
    imageDescriptionText.Text = imageDescription.Text;
    imageDescriptionText.FontSize = 24;
    imageDescriptionText.HorizontalAlignment = HorizontalAlignment.Center;
    imageDescriptionText.Width = pageDescription.PageSize.Width / 2;
    imageDescriptionText.TextWrapping = TextWrapping.WrapWholeWords;

    printLayout.Children.Add(printImage);
    printLayout.Children.Add(imageDescriptionText);

    // Add the print layout to the list of preview pages.
    printPreviewPages.Add(printLayout);

    // Report the number of preview pages created.
```

```
    PrintDocument printDocument = (PrintDocument)sender;
    printDocument.SetPreviewPageCount(printPreviewPages.Count,

PreviewPageCountType.Intermediate);
}
```

Here's a screenshot of the app UI and how the content appears in the print preview UI.



When a particular page is to be shown in the print preview window, the PrintManager raises the GetPreviewPage event. This corresponds with the `MakePage` method of the `IPrintPreviewPageCollection` interface. The event handler you created during registration is called at this time.

In the GetPreviewPage event handler, set the appropriate page on the print document.

C#

```
private void PrintDocument_GetPreviewPage(object sender,
GetPreviewPageEventArgs e)
{
    PrintDocument printDocument = (PrintDocument)sender;
    printDocument.SetPreviewPage(e.PageNumber,
printPreviewPages[e.PageNumber - 1]);
}
```

Finally, once the user clicks the print button, the PrintManager requests the final collection of pages to send to the printer by calling the `MakeDocument` method of the

`IDocumentPageSource` interface. In XAML, this raises the AddPages event. The event handler you created during registration is called at this time.

In the AddPages event handler, add pages from the page collection to the PrintDocument object to be sent to the printer. If a user specifies particular pages or a range of pages to print, you use that information here to add only the pages that will actually be sent to the printer.

C#

```
private void PrintDocument_AddPages(object sender, AddPagesEventArgs e)
{
    PrintDocument printDocument = (PrintDocument)sender;

    // Loop over all of the preview pages and add each one to be printed.
    for (int i = 0; i < printPreviewPages.Count; i++)
    {
        printDocument.AddPage(printPreviewPages[i]);
    }

    // Indicate that all of the print pages have been provided.
    printDocument.AddPagesComplete();
}
```

Related topics

- Printing and scanning
- Customize the print preview UI

Customize the print preview UI

Article • 09/11/2024

This topic describes how to customize the print options and settings in the print preview UI. For more info about printing, see Print from your app.

- ✓ **Important APIs**: Windows.Graphics.Printing namespace, PrintTask, StandardPrintTaskOptions class, Microsoft.UI.Xaml.Printing namespace, PrintDocument class

Customize print options

By default, the print preview UI shows the **ColorMode**, **Copies**, and **Orientation** print options. In addition to those, there are several other common printer options that you can add to the print preview UI:

- **Binding**
- **Bordering**
- **Collation**
- **CustomPageRanges**
- **Duplex**
- **HolePunch**
- **InputBin**
- **MediaSize**
- **MediaType**
- **NUp**
- **PrintQuality**
- **Staple**

These options are defined in the StandardPrintTaskOptions class. You can add to or remove options from the list of options displayed in the print preview UI. You can also change the order in which they appear, and set the default settings that are shown to the user.

However, the modifications that you make in this way affect only the print preview UI. The user can always access all of the options that the printer supports by tapping **More settings** in the print preview UI.

Define the options to display

When you register your app for printing (see Print from your app), part of that registration includes defining the PrintTaskRequested event handler. The code to customize the options displayed in the print preview UI is added to the PrintTaskRequested event handler.

After you create the PrintTask in the PrintTaskRequested event handler , you can get the DisplayedOptions list, which contains the option items shown in the print preview UI. You can modify this list by inserting, appending, removing, or re-ordering options.

> **Note**
>
> Although your app can specify any print options to be displayed, only those that are supported by the selected printer are shown in the print preview UI. The print UI won't show options that the selected printer doesn't support.

C#

```
private void PrintTask_Requested(PrintManager sender,
PrintTaskRequestedEventArgs args)
{
    // Create the PrintTask.
    // Defines the title and delegate for PrintTaskSourceRequested.
    PrintTask printTask = args.Request.CreatePrintTask("WinUI 3 Printing
example", PrintTaskSourceRequested);

    // Handle PrintTask.Completed to catch failed print jobs.
    printTask.Completed += PrintTask_Completed;

    DispatcherQueue.TryEnqueue(DispatcherQueuePriority.Normal, () =>
    {
        InvokePrintingButton.IsEnabled = false;
    });

    // Customize options displayed in print preview UI.
    // Get the list of displayed options.
    IList<string> displayedOptions = printTask.Options.DisplayedOptions;

    // Choose the printer options to be shown.
    // The order in which the options are appended determines
    // the order in which they appear in the UI.
    displayedOptions.Clear();
    displayedOptions.Add(StandardPrintTaskOptions.Copies);
    displayedOptions.Add(StandardPrintTaskOptions.Orientation);
    displayedOptions.Add(StandardPrintTaskOptions.MediaSize);
    displayedOptions.Add(StandardPrintTaskOptions.Collation);
    displayedOptions.Add(StandardPrintTaskOptions.Duplex);

    // Preset the default value of the print media size option.
```

```
    printTask.Options.MediaSize = PrintMediaSize.NorthAmericaLegal;
}
```

Specify default options

You can also set the default values of the options in the print preview UI. The following line of code, from the last example, sets the default value of the MediaSize option.

C#

```
// Preset the default value of the print media size option.
printTask.Options.MediaSize = PrintMediaSize.NorthAmericaLegal;
```

> **Note**
>
> When you set the values in the DisplayedOptions list, you use the name you get from StandardPrintTaskOptions (for example, StandardPrintTaskOptions.MediaSize).
>
> When you set the default value of an option, you use PrintTaskOptions (for example, PrintTaskOptions.MediaSize).

Add custom print options

Here, we demonstrate how to create a new custom print option, define a list of values that the option supports, and then add the option to the print preview. In this example, the custom print option lets the user specify whether to print only the text on the page, only the image, or both the text and image. The options are presented in a drop-down list.

In order to ensure a good user experience, the system requires that the app handle the PrintTaskRequested event within the time specified by PrintTaskRequestedEventArgs.Request.Deadline. Therefore, we use the PrintTaskRequested handler only to create the print task. The print settings customization can be done when the print document source is requested. Here, we use a lambda expression to define the PrintTaskSourceRequestedHandler inline, which provides easier access to the PrintTask.

First, get a PrintTaskOptionDetails object and its list of DisplayedOptions. You use this to add the new print option to the print preview UI.

Next, to present the custom print options in a drop-down list, call PrintTaskOptionDetails.CreateItemListOption to create a PrintCustomItemListOptionDetails object. Create the new print option and initialize the list of option values. Finally, add the new option to the DisplayedOptions list and assign a handler for the OptionChanged event. Because you are just adding a new print option at the end of the list of default options, you don't need to clear the DisplayedOptions list; just add the new option.

C#

```
private void PrintTask_Requested(PrintManager sender,
PrintTaskRequestedEventArgs args)
{
    // Create the PrintTask.
    PrintTask printTask = null;
    printTask = args.Request.CreatePrintTask("WinUI 3 Printing example",
sourceRequestedArgs =>
    {
        PrintTaskSourceRequestedDeferral deferral =
            sourceRequestedArgs.GetDeferral();
        PrintTaskOptionDetails printDetailedOptions =

PrintTaskOptionDetails.GetFromPrintTaskOptions(printTask.Options);
        IList<string> displayedOptions =
printDetailedOptions.DisplayedOptions;

        // Create a new list option.
        PrintCustomItemListOptionDetails pageFormat =
            printDetailedOptions.CreateItemListOption("PageContent", "Page
content");
        pageFormat.AddItem("PicturesText", "Pictures and text");
        pageFormat.AddItem("PicturesOnly", "Pictures only");
        pageFormat.AddItem("TextOnly", "Text only");

        // Add the custom option to the option list
        displayedOptions.Add("PageContent");

        printDetailedOptions.OptionChanged +=
PrintDetailedOptions_OptionChanged;
        sourceRequestedArgs.SetSource(printDocumentSource);

        deferral.Complete();
    });

    // Handle PrintTask.Completed to catch failed print jobs.
    printTask.Completed += PrintTask_Completed;

    DispatcherQueue.TryEnqueue(DispatcherQueuePriority.Normal, () =>
    {
        InvokePrintingButton.IsEnabled = false;
```

```
    });
}
```

The options appear in the print preview UI in the same order they are appended, with the first option shown at the top of the window. In this example, the custom option is appended last so that it appears at the bottom of the list of options. However, you could put it anywhere in the list; custom print options don't need to be added last.

When the user changes the selected option in your custom option, use the selected option to update the print preview image. After the print preview layout in updated, call the InvalidatePreview method to redraw the image in the print preview UI, as shown here.

C#

```
void PrintDetailedOptions_OptionChanged(PrintTaskOptionDetails sender,
                                        PrintTaskOptionChangedEventArgs
args)
{
    string optionId = args.OptionId as string;
    if (string.IsNullOrEmpty(optionId))
    {
        return;
    }

    if (optionId == "PageContent")
    {
        PrintCustomItemListOptionDetails pageContentOption =
            (PrintCustomItemListOptionDetails)sender.Options["PageContent"];
        string pageContentValue = pageContentOption.Value.ToString();

        if (pageContentValue == "PicturesOnly")
        {
            pageLayoutOption = PageLayoutOption.Images;
        }
        else if (pageContentValue == "TextOnly")
        {
            pageLayoutOption = PageLayoutOption.Text;
        }
        else
        {
            pageLayoutOption = PageLayoutOption.TextAndImages;
        }

        DispatcherQueue.TryEnqueue(DispatcherQueuePriority.Normal, () =>
        {
            printDocument.InvalidatePreview();
        });
    }
}
```

To support use of the custom options in your page, add an enum for the available options and a page-level variable to hold the selected option.

C#

```
internal enum PageLayoutOption : int
{
    Text = 1,
    Images = 2,
    TextAndImages = 3
}

PageLayoutOption pageLayoutOption = PageLayoutOption.TextAndImages;
```

Then, use the selected option in your Paginate handler where you add the content for the print preview.

C#

```
private void PrintDocument_Paginate(object sender, PaginateEventArgs e)
{
    // Clear the cache of preview pages.
    printPreviewPages.Clear();

    // Get the PrintTaskOptions.
    PrintTaskOptions printingOptions =
((PrintTaskOptions)e.PrintTaskOptions);
    // Get the page description to determine the size of the print page.
    PrintPageDescription pageDescription =
printingOptions.GetPageDescription(0);

    // Create the print layout.
    StackPanel printLayout = new StackPanel();
    printLayout.Width = pageDescription.PageSize.Width;
    printLayout.Height = pageDescription.PageSize.Height;
    printLayout.BorderBrush = new
Microsoft.UI.Xaml.Media.SolidColorBrush(Microsoft.UI.Colors.DimGray);
    printLayout.BorderThickness = new Thickness(48);

    // Use the custom print layout options to determine
    // which elements to add to the print page.
    if (pageLayoutOption == PageLayoutOption.Images ||
        pageLayoutOption == PageLayoutOption.TextAndImages)
    {
        Image printImage = new Image();
        printImage.Source = printContent.Source;

        printImage.Width = pageDescription.PageSize.Width / 2;
        printImage.Height = pageDescription.PageSize.Height / 2;
        printLayout.Children.Add(printImage);
    }
```

```
    if (pageLayoutOption == PageLayoutOption.Text ||
        pageLayoutOption == PageLayoutOption.TextAndImages)
    {
        TextBlock imageDescriptionText = new TextBlock();
        imageDescriptionText.Text = imageDescription.Text;
        imageDescriptionText.FontSize = 24;
        imageDescriptionText.HorizontalAlignment =
HorizontalAlignment.Center;
        imageDescriptionText.Width = pageDescription.PageSize.Width / 2;
        imageDescriptionText.TextWrapping = TextWrapping.WrapWholeWords;

        printLayout.Children.Add(imageDescriptionText);
    }

    // Add the print layout to the list of preview pages.
    printPreviewPages.Add(printLayout);

    // Report the number of preview pages created.
    PrintDocument printDocument = (PrintDocument)sender;
    printDocument.SetPreviewPageCount(printPreviewPages.Count,

PreviewPageCountType.Intermediate);
}
```

See also

- Printing and scanning
- Print from your app

Scan from your app

Article • 09/11/2024

This topic describes how to scan content from your app by using a flatbed, feeder, or auto-configured scan source.

Important APIs

- Windows.Devices.Scanners
- DeviceInformation
- DeviceClass

To scan from your app, you must first list the available scanners by declaring a new DeviceInformation object and getting the DeviceClass type. Only scanners that are installed locally with WIA drivers are listed and available to your app.

After your app has listed available scanners, it can use the auto-configured scan settings based on the scanner type, or just scan using the available flatbed or feeder scan source. To use auto-configured settings, the scanner must be enabled for auto-configuration and must not be equipped with both a flatbed and a feeder scanner. For more info, see Auto-Configured Scanning.

Enumerate available scanners

Windows does not detect scanners automatically. You must perform this step in order for your app to communicate with the scanner. In this example, the scanner device enumeration is done using the Windows.Devices.Enumeration namespace.

1. First, add these using statements to your class definition file.

C#

```
using Windows.Devices.Enumeration;
using Windows.Devices.Scanners;
```

2. Next, implement a device watcher to start enumerating scanners. For more info, see Enumerate devices.

C#

```
void InitDeviceWatcher()
{
   // Create a Device Watcher class for type Image Scanner for
```

```
enumerating scanners
        scannerWatcher =
DeviceInformation.CreateWatcher(DeviceClass.ImageScanner);

        scannerWatcher.Added += OnScannerAdded;
        scannerWatcher.Removed += OnScannerRemoved;
        scannerWatcher.EnumerationCompleted += OnScannerEnumerationComplete;
    }
```

3. Create an event handler for when a scanner is added.

C#

```
    private async void OnScannerAdded(DeviceWatcher sender,
DeviceInformation deviceInfo)
    {
        await
        MainPage.Current.Dispatcher.RunAsync(
              Windows.UI.Core.CoreDispatcherPriority.Normal,
              () =>
              {
                 MainPage.Current.NotifyUser(String.Format("Scanner with
device id {0} has been added", deviceInfo.Id), NotifyType.StatusMessage);

                 // search the device list for a device with a matching
device id
                 ScannerDataItem match = FindInList(deviceInfo.Id);

                 // If we found a match then mark it as verified and return
                 if (match != null)
                 {
                    match.Matched = true;
                    return;
                 }

                 // Add the new element to the end of the list of devices
                 AppendToList(deviceInfo);
              }
        );
    }
```

Scan

1. **Get an ImageScanner object**

For each ImageScannerScanSource enumeration type, whether it's **Default**, **AutoConfigured**, **Flatbed**, or **Feeder**, you must first create an ImageScanner object by calling the ImageScanner.FromIdAsync method, like this.

```
C#

   ImageScanner myScanner = await ImageScanner.FromIdAsync(deviceId);
```

2. **Just scan**

To scan with the default settings, your app relies on the **Windows.Devices.Scanners** namespace to select a scanner and scans from that source. No scan settings are changed. The possible scanners are auto-configure, flatbed, or feeder. This type of scan will most likely produce a successful scan operation, even if it scans from the wrong source, like flatbed instead of feeder.

Note If the user places the document to scan in the feeder, the scanner will scan from the flatbed instead. If the user tries to scan from an empty feeder, the scan job won't produce any scanned files.

```
C#

    var result = await
myScanner.ScanFilesToFolderAsync(ImageScannerScanSource.Default,
        folder).AsTask(cancellationToken.Token, progress);
```

3. **Scan from Auto-configured, Flatbed, or Feeder source**

Your app can use the device's Auto-Configured Scanning to scan with the most optimal scan settings. With this option, the device itself can determine the best scan settings, like color mode and scan resolution, based on the content being scanned. The device selects the scan settings at run time for each new scan job.

Note Not all scanners support this feature, so the app must check if the scanner supports this feature before using this setting.

In this example, the app first checks if the scanner is capable of auto-configuration and then scans. To specify either flatbed or feeder scanner, simply replace **AutoConfigured** with **Flatbed** or **Feeder**.

```
C#

    if
(myScanner.IsScanSourceSupported(ImageScannerScanSource.AutoConfigured))
    {
        ...
        // Scan API call to start scanning with Auto-Configured settings.
        var result = await myScanner.ScanFilesToFolderAsync(
            ImageScannerScanSource.AutoConfigured,
folder).AsTask(cancellationToken.Token, progress);
```

```
        ...
    }
```

Preview the scan

You can add code to preview the scan before scanning to a folder. In the example below, the app checks if the **Flatbed** scanner supports preview, then previews the scan.

C#

```
if (myScanner.IsPreviewSupported(ImageScannerScanSource.Flatbed))
{
    rootPage.NotifyUser("Scanning", NotifyType.StatusMessage);
                // Scan API call to get preview from the flatbed.
                var result = await myScanner.ScanPreviewToStreamAsync(
                    ImageScannerScanSource.Flatbed, stream);
```

Cancel the scan

You can let users cancel the scan job midway through a scan, like this.

C#

```
void CancelScanning()
{
    if (ModelDataContext.ScenarioRunning)
    {
        if (cancellationToken != null)
        {
            cancellationToken.Cancel();
        }
        DisplayImage.Source = null;
        ModelDataContext.ScenarioRunning = false;
        ModelDataContext.ClearFileList();
    }
}
```

Scan with progress

1. Create a **System.Threading.CancellationTokenSource** object.

C#

```
cancellationToken = new CancellationTokenSource();
```

2. Set up the progress event handler and get the progress of the scan.

```
C#
```

```
    rootPage.NotifyUser("Scanning", NotifyType.StatusMessage);
    var progress = new Progress<UInt32>(ScanProgress);
```

Scanning to the pictures library

Users can scan to any folder dynamically using the **FolderPicker** class, but you must declare the *Pictures Library* capability in the manifest to allow users to scan to that folder. For more info on app capabilities, see App capability declarations.

Graphics

Article • 05/25/2023

This article provides an index of development features that are related to scenarios involving graphics in Windows apps.

Win2D

Win2D is an easy-to-use Windows Runtime API for immediate mode 2D graphics rendering with GPU acceleration.

Windows App SDK features

The Windows App SDK provides the following features related to graphics scenarios for Windows 10 and later OS releases.

Expand table

Feature	Description
Render text with DWriteCore	Use the C++/COM APIs in the DWriteCore headers of the Windows App SDK to render text using a device-independent text layout system, high quality sub-pixel Microsoft ClearType text rendering, hardware-accelerated text, multi-format text, wide language support, and much more.

Windows OS features

Windows 10 and later OS releases provide a wide variety of APIs related to graphics scenarios for apps. These features are available via a combination of WinRT and Win32 (C++ and COM) APIs provided by the Windows SDK.

WinRT APIs

The following articles provide information about features available via WinRT APIs provided by the Windows SDK.

Expand table

Article	Description
Visual layer	Learn how to use the visual layer in UWP apps. The visual layer provides a high performance, retained-mode API for graphics, effects and animations, and is the foundation for all WinRT XAML-based UI in Windows apps.
Using the Visual layer in desktop apps	Learn how to use the visual layer in WPF, Windows Forms, and C++ Win32 apps.
XAML platform	Learn the fundamentals of the XAML language and concepts for WinRT-based graphic components Windows apps.

Win32 (C++ and COM) APIs

The following articles provide information about features available via Win32 (C++ and COM) APIs provided by the Windows SDK.

Expand table

Article	Description
Graphics and gaming	Learn about the breadth of Win32 APIs related to graphics, gaming, and imaging.
DirectX	DirectX graphics provides a set of APIs that you can use to create games and other high-performance multimedia apps.
Direct2D	Direct2D is a hardware-accelerated, immediate-mode, 2-D graphics API that provides high performance and high-quality rendering for 2-D geometry, bitmaps, and text.
Direct3D	Direct3D enables you to create 3-D graphics for games and scientific apps.
DirectWrite	DirectWrite supports high-quality text rendering, resolution-independent outline fonts, and full Unicode text and layouts.
Windows GDI	The graphics device interface (GDI) is a foundational API that enables apps to use graphics and formatted text on both the video display and the printer.

.NET features

The .NET SDK also provides APIs related to graphics scenarios for WPF and Windows Forms apps.

Expand table

Article	Description
Graphics (WPF)	Learn about ways to integrate support for multimedia, vector graphics, animation, and content composition in WPF apps.
Graphics and drawing (Windows Forms)	Learn about ways to create graphics, draw text, and manipulate graphical images in Windows Forms apps.

Enhance UI with the Visual layer (Windows App SDK/WinUI 3)

Article • 08/19/2024

The Visual layer in the Windows App SDK/WinUI 3 provides a high performance, retained-mode API for graphics, effects, animations, and input. It's the foundation for all UI across Windows devices.

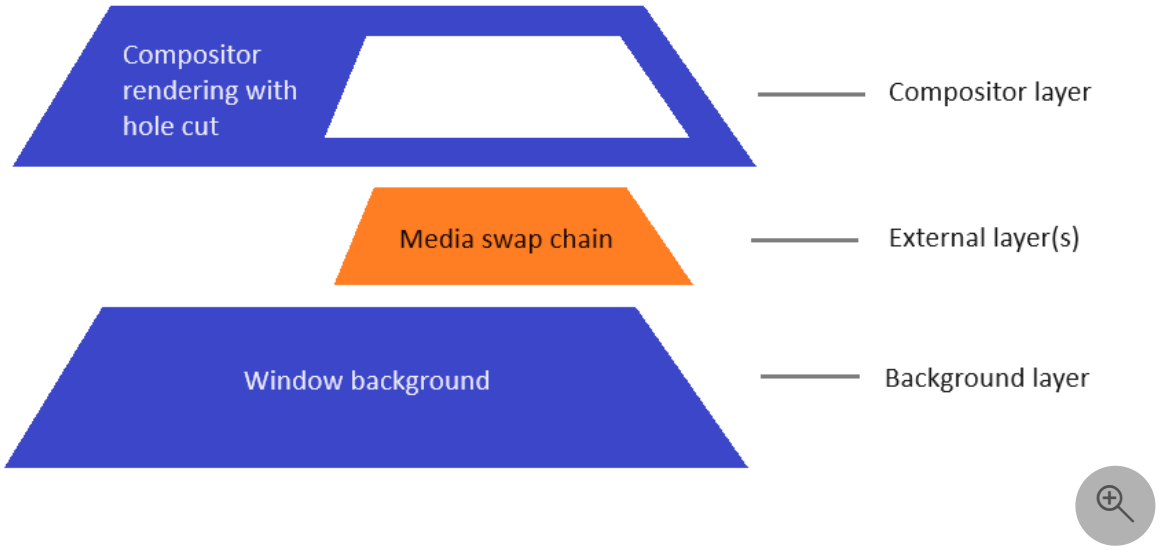
The types in Microsoft.UI.Composition form the Windows App SDK/WinUI 3 implementation of the Visual layer. It's very similar to the UWP Visual layer, which is implemented in the **Windows.UI.Composition** namespace. The differences between the two and the missing functionality from the Windows App SDK/WinUI 3 Visual layer are detailed below.

Differences from UWP

The most obvious difference between **Microsoft.UI.Composition** and the UWP Visual layer is the namespace. **Microsoft.UI.Composition** provides access to functionality that's nearly identical to that of the UWP Visual layer, in the most commonly used scenarios. But there are exceptions, and differences.

The Windows App SDK/WinUI 3 uses **Microsoft.UI.Composition** to describe a tree of individual Visuals. Those Visuals are then composited to create the complete rendering of the window. It's very similar to how **Windows.UI.Composition** works (for UWP apps). But one big difference is that the **Microsoft.UI.Composition** compositor runs entirely within a Windows App SDK app; and it has access only to pixels that it drew. That means that any *external* content (content that wasn't drawn by the compositor) is unknown to the compositor. Which creates certain limitations.

An example of external content is the (**Microsoft.UI.Xaml.Controls**) MediaPlayerElement. The Windows media stack provides to XAML an opaque media swap chain handle. XAML gives that handle to the compositor, which in turn hands it off to Windows (via **Windows.UI.Composition**) to display. More specifically, since the compositor can't see any of the pixels in the media swap chain, it can't composite that as part of the overall rendering for the window. Instead, it gives the media swap chain to Windows to render it below the compositor's rendering, with a hole cut out of the compositor's rendering in order to allow the media swap chain below it to be visible. Here's a visualization of this setup.



In general, the approach allows external content (such as swap chains) to be part of the overall rendering for the window; and it still allows other content drawn by the compositor to overlap on top of the external content—for example, XAML's MediaTransportControls rendering on top of the media. But it doesn't allow more advanced effect interactions with this external content. That applies to all approaches that one could use to plug a swap chain into a view: either via XAML (for example, via SwapChainPanel), or via Composition (for example, using ICompositorInterop::CreateCompositionSurfaceForSwapChain).

In the Windows App SDK/WinUI 3, the following APIs all create external content.

- MediaPlayerElement
- SwapChainPanel
- WebView2
- MicaBackdrop and DesktopAcrylicBackdrop, as well as the underlying MicaController and DesktopAcrylicController that they use.

The model, described above, of handling external content creates these limitations:

- It's not possible to have compositor content behind external content (that is, behind/below the external layer(s) in the diagram above). So, for example, it's not possible to give a **WebView2** a transparent background in order to be able to see XAML buttons or images behind it. The only things that can be behind external content are *other* external content and the window background. Because of that, we discourage/disable transparency for external content.
- It's not possible to have compositor content sample from external content. For example, AcrylicBrush isn't able to sample and blur any pixels from a **MediaPlayerElement**. **AcrylicBrush** will sample from the compositor's image, which

is transparent black for external content areas. So that transparent black will be what **AcrylicBrush** blurs. Similarly, **AcrylicBrush** in front of a **MicaBackdrop** or **DesktopAcrylicBackdrop** can't see any colors that those backdrops will draw; and instead, the brush will blur the transparent black.

- Another scenario is known as *destination invert*, which is used for the caret of text box controls to invert the pixels that the text insertion caret is in front of. That invert similarly samples from the compositor surface, and it will be impacted if the text box doesn't have an opaque background that's drawn by the compositor.
- Because the WinUI 3 SwapChainPanel creates external content, it doesn't support transparency. Nor does it support applying AcrylicBrush and other effects that use a CompositionBackdropBrush in front of it. **AcrylicBrush** and these other **CompositionBackdropBrush**-based effects are not able to sample from a **SwapChainPanel**.
- In desktop apps (a WinUI 3 app is a desktop app), **Window.Current** is `null`. So you can't retrieve an instance of Compositor from `Window.Current.Compositor`. In XAML apps (both WinUI 3 and UWP), we recommend that you call ElementCompositionPreview.GetElementVisual(UIElement) to get a Composition Visual, and retrieve the `Compositor` from the visual's Compositor property. In cases where you don't have access to a **UIElement** (for example, if you create a CompositionBrush in a class library), you can call CompositionTarget.GetCompositorForCurrentThread.

For more details about the Visual layer, see the Visual layer overview in the UWP documentation. In most cases, the documentation and samples are also applicable to **Microsoft.UI.Composition**.

Prerequisites to using Microsoft.UI.Composition

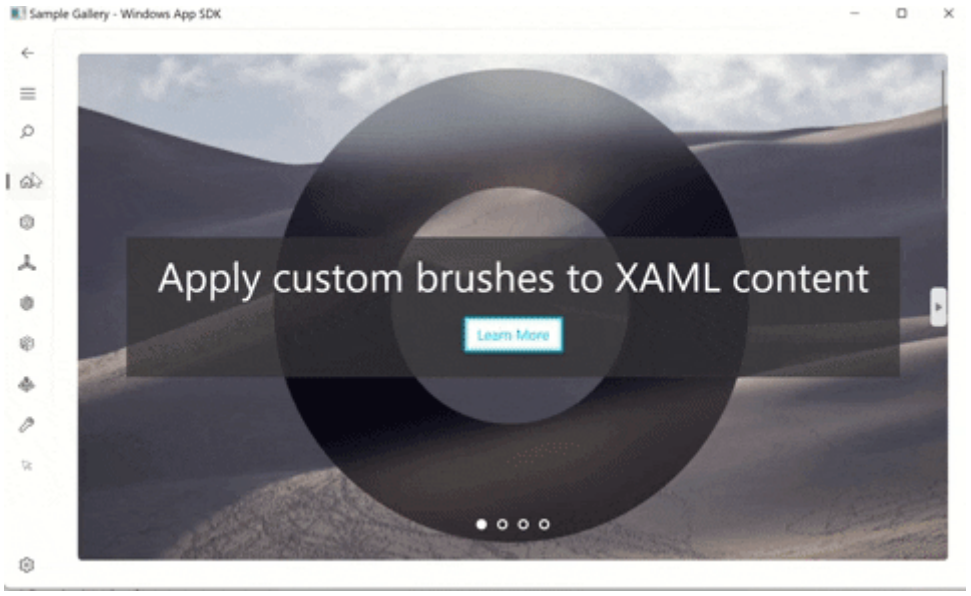
To use **Microsoft.UI.Composition** APIs in the Windows App SDK/WinUI 3:

1. Download and install the latest release of the Windows App SDK. For more information, see Start developing Windows apps.
2. Create a new project as described in Start developing Windows apps, or follow the instructions to use the Windows App SDK in an existing project.

To learn more about the availability of **Microsoft.UI.Composition** in the Windows App SDK/WinUI 3, see release channels.

Windows Composition Samples Gallery

We've updated the Windows Composition Samples Gallery to now take a dependency on the Windows App SDK Composition APIs. Visit WindowsCompositionSamples to see the Microsoft.UI.Composition APIs in action.



Related topics

- UWP Visual layer overview
- Using the Visual layer in desktop apps

Render text with DWriteCore

Article • 08/21/2024

DWriteCore is the Windows App SDK implementation of DirectWrite (DirectWrite is the DirectX API for high-quality text rendering, resolution-independent outline fonts, and full Unicode text and layout support).

DWriteCore provides access to all current DirectWrite features for text rendering, including a device-independent text layout system, hardware-accelerated text, multi-format text, and wide language support.

For more details about DWriteCore—including guidance for setting up your development environment, and code examples for programming with it—see DWriteCore overview in the DirectWrite documentation.

Prerequisites to using DWriteCore

To use DWriteCore APIs in the Windows App SDK:

1. Download and install the latest release of the Windows App SDK. For more information, see Start developing Windows apps.
2. Create a new project as described in Start developing Windows apps, or follow the instructions to use the Windows App SDK in an existing project.

To learn more about the availability of DWriteCore in the Windows App SDK, see release channels.

Overview of Win2D

Article • 10/25/2023

Win2D is an easy-to-use Windows Runtime (WinRT) API for immediate-mode 2D graphics rendering with GPU acceleration. It's ideal for creating simple games, displays such as charts, and other simple 2D graphics.

You can use Win2D in your WinUI 3 (Windows App SDK) apps or Universal Windows Platform (UWP) apps, using either C#, C++, or VB. Win2D utilizes the power of Direct2D, and it integrates seamlessly with XAML on both WinUI 3 (Windows App SDK) and UWP.

> **ⓘ Important**
>
> Win2D for WinUI 3 (Windows App SDK) is a work-in-progress, and some features aren't supported. This documentation refers to both the WinUI 3 and UWP versions, which mostly share the same API surface and functionality. Whenever there's any relevant differences between the two, we'll call it out in the documentation. But otherwise, the info being presented applies to both platforms in the same way.

Get started

Win2D is available as a NuGet package, or as source code (for the source code, see the Win2D repo on GitHub).

Reference the Win2D NuGet package

In a WinUI 3 or UWP project in Visual Studio, click **Tools** > **NuGet Package Manager** > **Manage NuGet Packages for Solution...** > **Browse**. Make sure that **Include prerelease** is unchecked, and type or paste into the search box:

- For a WinUI 3 project, *Microsoft.Graphics.Win2D*.
- For a UWP project, *Win2D.uwp*.

Select the correct item in search results, check your project, and click **Install** to install the package into that project. Accept the license agreement.

> **ⓘ Important**
>
> If you see any error messages, then try updating the version of the Windows App SDK NuGet package that you're referencing (if appropriate). Or try going into

project properties, and setting **Target OS version** to the latest version.

Next steps

Next, to learn about creating a simple app, try out the Win2D "Hello, World!" quickstart, or the Build a simple Win2D app tutorial. You can also consult the features list to discover all the things Win2D can do. To learn more about advanced topics, you can refer to the collection of articles included in the documentation here as well.

Reference

The Win2D APIs.

Win2D "Hello, World!" quickstart

Article • 11/17/2023

In this topic you'll create a very simple "Hello, World!" project for Win2D.

In Visual Studio, create a new project from one of the following project templates:

- **WinUI 3 (Windows App SDK)**. To create a new WinUI 3 project, use the **Blank App, Packaged (WinUI 3 in Desktop)** project template. You can find that project template by choosing language: either *C#* or *C++*; platform: *Windows*; project type: *Desktop*.
- **Universal Windows Platform (UWP)**. To create a new UWP project, use the **Blank App (Universal Windows)** or **Blank App (C++/WinRT)** or **Blank App (Universal Windows - C++/CX)** project template. For language, choose: either *C#* or *C++*; platform: *Windows*; project type: *UWP*.

> **ⓘ Important**
>
> For info about how to set up your project to use Win2D, see **Reference the Win2D NuGet package**.

To host Win2D content, you'll need to add a Win2D **CanvasControl** to your project's `MainWindow.xaml` (or `MainPage.xaml`, for a UWP project).

First, add the following xml namespace declaration:

XAML

```
xmlns:win2dcanvas="using:Microsoft.Graphics.Canvas.UI.Xaml"
```

And then add a **CanvasControl**, prefixed with that xml namespace. For example, you could add a **Grid** as your layout root, like this:

XAML

```
<Grid>
    <win2dcanvas:CanvasControl Draw="CanvasControl_Draw"
ClearColor="CornflowerBlue"/>
</Grid>
```

The project won't build at the moment, due to the referenced-but-not-implemented **Draw** event handler. So we'll remedy that next, while we add some drawing code to

interact with the **CanvasControl**.

For a WinUI 3 (Windows App SDK) project

For a C# project, add the following event handler to `MainWindow.xaml.cs`:

C#

```
// MainWindow.xaml.cs
...
public sealed partial class MainWindow : Window
{
    ...
    void CanvasControl_Draw(
        Microsoft.Graphics.Canvas.UI.Xaml.CanvasControl sender,
        Microsoft.Graphics.Canvas.UI.Xaml.CanvasDrawEventArgs args)
    {
        args.DrawingSession.DrawEllipse(155, 115, 80, 30,
Microsoft.UI.Colors.Black, 3);
        args.DrawingSession.DrawText("Hello, Win2D World!", 100, 100,
Microsoft.UI.Colors.Yellow);
    }
}
...
```

For a C++/WinRT project, add the following event handler to `MainWindow.xaml.h` and `MainWindow.xaml.cpp`:

C++/WinRT

```
// MainWindow.xaml.h
...
void CanvasControl_Draw(
    winrt::Microsoft::Graphics::Canvas::UI::Xaml::CanvasControl const&
sender,
    winrt::Microsoft::Graphics::Canvas::UI::Xaml::CanvasDrawEventArgs const&
args);
...

// MainWindow.xaml.cpp
...
namespace winrt::MYPROJECT::implementation
{
    ...
    void MainWindow::CanvasControl_Draw(
        winrt::Microsoft::Graphics::Canvas::UI::Xaml::CanvasControl const&
sender,
        winrt::Microsoft::Graphics::Canvas::UI::Xaml::CanvasDrawEventArgs
const& args)
    {
```

```
        args.DrawingSession().DrawEllipse(155, 115, 80, 30,
winrt::Microsoft::UI::Colors::Black(), 3);
        args.DrawingSession().DrawText(L"Hello, Win2D World!", 100, 100,
winrt::Microsoft::UI::Colors::Yellow());
    }
}
```

You can now build and run the project. You'll see some Win2D content—a black ellipse with a yellow "Hello, World!" message in front of it.

For a UWP project

For a C# project, you can use the same C# code as for a WinUI 3 project (see the For a WinUI 3 project section above). The only differences are that you'll be editing `MainPage.xaml.cs` instead of `MainWindow.xaml.cs`. And you'll need to change `Microsoft.UI.Colors` to `Windows.UI.Colors`.

For a C++/WinRT project, add the following code to `pch.h`, `MainPage.h`, and `MainPage.cpp`:

C++/WinRT

```
// pch.h
...
#include <winrt/Microsoft.Graphics.Canvas.UI.Xaml.h>

// MainPage.h
...
void CanvasControl_Draw(
    winrt::Microsoft::Graphics::Canvas::UI::Xaml::CanvasControl const&
sender,
    winrt::Microsoft::Graphics::Canvas::UI::Xaml::CanvasDrawEventArgs const&
args);
...

// MainPage.cpp
...
namespace winrt::MYPROJECT::implementation
{
    ...
    void MainPage::CanvasControl_Draw(
        winrt::Microsoft::Graphics::Canvas::UI::Xaml::CanvasControl const&
sender,
        winrt::Microsoft::Graphics::Canvas::UI::Xaml::CanvasDrawEventArgs
const& args)
    {
        args.DrawingSession().DrawEllipse(155, 115, 80, 30,
winrt::Windows::UI::Colors::Black(), 3);
        args.DrawingSession().DrawText(L"Hello, Win2D World!", 100, 100,
```

```
winrt::Windows::UI::Colors::Yellow());
    }
}
```

For a C++/CX project, add the following event handler to `MainPage.xaml.h` and `MainPage.xaml.cpp`:

C++/CX

```
// MainPage.xaml.h
...
void CanvasControl_Draw(
    Microsoft::Graphics::Canvas::UI::Xaml::CanvasControl^ sender,
    Microsoft::Graphics::Canvas::UI::Xaml::CanvasDrawEventArgs^ args);
...

// MainWindow.xaml.cpp
...
void MainPage::CanvasControl_Draw(
    Microsoft::Graphics::Canvas::UI::Xaml::CanvasControl^ sender,
    Microsoft::Graphics::Canvas::UI::Xaml::CanvasDrawEventArgs^ args)
{
    args->DrawingSession->DrawEllipse(155, 115, 80, 30,
Windows::UI::Colors::Black, 3);
    args->DrawingSession->DrawText("Hello, Win2D World!", 100, 100,
Windows::UI::Colors::Yellow);
}
```

See Also

- Overview of Win2D
- Win2D quickstart

Build a simple Win2D app

Article • 10/25/2023

This tutorial introduces some of the basic drawing capabilities of Win2D. You'll learn how to:

- Add Win2D to a C# XAML Windows project.
- Draw text and geometry.
- Apply filter effects.
- Animate your Win2D content.
- Follow Win2D best practices.

Set up your dev machine

Make sure to set up your machine with all the necessary tools:

- Install Visual Studio
  - Include the UWP or Windows SDK (depending on your needs), 17763+
- If using UWP, ensure to also enable developer mode

Create a new Win2D project

Follow the steps in the Win2D "Hello, World!" quickstart to create a new project using Win2D, and to add a reference to the Win2D NuGet package. You can use either WinUI 3 (Windows App SDK) or the Universal Windows Platform (UWP).

Add a Win2D CanvasControl to your app's XAML

1. In order to use Win2D, you need somewhere to draw your graphics. In a XAML app, the simplest way to do this is to add a CanvasControl to your XAML page.

Before you continue, first ensure that the project's Architecture option is set to `x86` or `x64` and **not** to `Any CPU`. Win2D is implemented in C++, and therefore projects that use Win2D need to be targeted to a specific CPU architecture.

1. Navigate to `MainPage.xaml` in your project by double clicking on it in Solution Explorer. This will open the file. For convenience, you can double click on the XAML

button in the Designer tab; this will hide the visual designer and reserve all of the space for the code view.

2. Before you add the control, you first have to tell XAML where **CanvasControl** is defined. To do this, go to the definition of the **Page** element, and add this directive: `xmlns:canvas="using:Microsoft.Graphics.Canvas.UI.Xaml"`. Your XAML should now look like this:

XAML

```
<Page
    ...
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:canvas="using:Microsoft.Graphics.Canvas.UI.Xaml"
    mc:Ignorable="d">
```

4. Now, add a new `canvas:CanvasControl` as a child element to the root **Grid** element. Give the control a name—for example, "canvas". Your XAML should now look like this:

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <canvas:CanvasControl x:Name="canvas"/>
</Grid>
```

5. Next, define an event handler for the Draw event. **CanvasControl** raises `Draw` whenever your app needs to draw or redraw its content. The easiest way is to let Visual Studio AutoComplete assist you. In the **CanvasControl** definition, begin typing a new attribute for the `Draw` event handler:

XAML

```
<canvas:CanvasControl x:Name="canvas" Draw="canvas_Draw" />
```

Note

Once you have entered in `Draw="`, Visual Studio should pop up a box prompting you to let it automatically fill out the right definition for the event handler. Press TAB to accept Visual Studio's default event handler. This will also automatically add a correctly formatted event handler method in your code behind

(`MainPage.xaml.cs``). Don't worry if you didn't use AutoComplete; you can manually add the event handler method in the next step.

Draw your first text in Win2D

1. Now, let's go to the C# code behind. Open `MainPage.xaml.cs` from Solution Explorer.

2. At the top of the C# file are various namespace definitions. Add the following namespaces:

C#

```
using Windows.UI;
using System.Numerics;
using Microsoft.Graphics.Canvas;
using Microsoft.Graphics.Canvas.Effects;
```

3. Next, you should see the following blank event handler which was inserted by AutoComplete:

C#

```
private void canvas_Draw(
    Microsoft.Graphics.Canvas.UI.Xaml.CanvasControl sender,
    Microsoft.Graphics.Canvas.UI.Xaml.CanvasDrawEventArgs args)
{
}
```

(If you didn't use AutoComplete in the previous step, add this code now.)

4. The CanvasDrawEventArgs parameter exposes a member, **DrawingSession**, which is of the type CanvasDrawingSession. This class provides most of the basic drawing functionality in Win2D: it has methods such as CanvasDrawingSession.DrawRectangle, CanvasDrawingSession.DrawImage, and the method you need to draw text, CanvasDrawingSession.DrawText.

Add the following code to the `canvas_Draw` method:

C#

```
args.DrawingSession.DrawText("Hello, World!", 100, 100, Colors.Black);
```

The first argument, `"Hello, World!"`, is the string that you want Win2D to display. The two "100"s tell Win2D to offset this text by 100 DIPs (device-independent pixels) to the right and down. Finally, `Colors.Black` defines the color of the text.

5. Now you are ready to run your first Win2D app. Press the F5 key to compile and launch. You should see an empty window with "Hello, world!" in black.

Correctly dispose of Win2D resources

1. Before continuing on to draw other kinds of content, you first should add some code to ensure your app avoids memory leaks. Most Win2D applications written in a .NET language and using a Win2D control like CanvasControl need to follow the below steps. Strictly speaking, your simple "Hello, world" app is not affected, but this is a good practice to follow in general.

For more information, see Avoiding memory leaks.

2. Open `MainPage.xaml` and find the Page XAML element that contains your **CanvasControl**. It should be the first element in the file.

3. Add a handler for the `Unloaded` event. Your XAML should look like this:

XAML

```
<Page
    ...
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:canvas="using:Microsoft.Graphics.Canvas.UI.Xaml"
    mc:Ignorable="d"
    Unloaded="Page_Unloaded">
```

4. Go to `MainPage.xaml.cs` and find the `Page_Unloaded` event handler. Add the following code:

C#

```
void Page_Unloaded(object sender, RoutedEventArgs e)
{
    this.canvas.RemoveFromVisualTree();
    this.canvas = null;
}
```

5. If your app contains multiple Win2D controls, then you need to repeat the above steps for each XAML page that contains a Win2D control. Your app currently only

has a single **CanvasControl** so you're all done.

Draw some shapes

1. It's just as easy to add 2D geometry to your app. Add the following code to the end of `canvas_Draw`:

```
C#

args.DrawingSession.DrawCircle(125, 125, 100, Colors.Green);
args.DrawingSession.DrawLine(0, 0, 50, 200, Colors.Red);
```

The arguments to these two methods are similar to `DrawText`. A circle is defined by a center point (125, 125), a radius (100), and a color (Green). A line is defined by a beginning (0, 0), an end (50, 200) and a color (Red).

12. Now, press F5 to run the app. You should see "Hello, world!" along with a green circle and red line.

You may be wondering how to control more advanced drawing options, such as line thickness and dashes, or more complex fill options such as using brushes. Win2D provides all of these options and more, and makes it easy to use them when you want. All of the `Draw(...)` methods offer many overloads that can accept additional parameters such as CanvasTextFormat (font family, size, etc) and CanvasStrokeStyle (dashes, dots, endcaps, etc). Feel free to explore the API surface to learn more about these options.

Dynamically generate drawing parameters

1. Now, let's add some variety by drawing a bunch of shapes and text with randomized colors.

Add the following code to the top of your `MainPage` class. This is helper functionality to generate random values that you will use when drawing:

```
C#

Random rnd = new Random();
private Vector2 RndPosition()
{
    double x = rnd.NextDouble() * 500f;
    double y = rnd.NextDouble() * 500f;
    return new Vector2((float)x, (float)y);
}
```

```
private float RndRadius()
{
    return (float)rnd.NextDouble() * 150f;
}

private byte RndByte()
{
    return (byte)rnd.Next(256);
}
```

2. Modify your `canvas_Draw` method to draw using these random parameters:

C#

```
private void canvas_Draw(
    Microsoft.Graphics.Canvas.UI.Xaml.CanvasControl sender,
    Microsoft.Graphics.Canvas.UI.Xaml.CanvasDrawEventArgs args)
{
    args.DrawingSession.DrawText("Hello, World!", RndPosition(),
Color.FromArgb(255, RndByte(), RndByte(), RndByte()));
    args.DrawingSession.DrawCircle(RndPosition(), RndRadius(),
Color.FromArgb(255, RndByte(), RndByte(), RndByte()));
    args.DrawingSession.DrawLine(RndPosition(), RndPosition(),
Color.FromArgb(255, RndByte(), RndByte(), RndByte()));
}
```

Let's break down how `DrawText` has changed. `"Hello, World!"` remains the same as before. The x and y offset parameters have been replaced with a single System.Numerics.Vector2 which is generated by `RndPosition`. Finally, instead of using a predefined color, `Color.FromArgb` allows you to define a color using A, R, G and B values. A is alpha, or the opacity level; in this case you always want fully opaque (255).

`DrawCircle` and `DrawLine` operate similarly to `DrawText`.

3. Finally, wrap your drawing code in a `for` loop. You should end up with the following `canvas_Draw` code:

C#

```
for (int i = 0; i < 100; i++)
{
    args.DrawingSession.DrawText("Hello, World!", RndPosition(),
Color.FromArgb(255, RndByte(), RndByte(), RndByte()));
    args.DrawingSession.DrawCircle(RndPosition(), RndRadius(),
Color.FromArgb(255, RndByte(), RndByte(), RndByte()));
    args.DrawingSession.DrawLine(RndPosition(), RndPosition(),
```

```
Color.FromArgb(255, RndByte(), RndByte(), RndByte()));
}
```

4. Run the app again. You should see a whole bunch of text, lines and circles with random positions and sizes.

Apply an image effect to your content

Image effects, also known as filter effects, are graphical transformations that are applied to pixel data. Saturation, hue rotation, and Gaussian blur are some common image effects. Image effects can be chained together, producing sophisticated visual appearance for minimal effort.

You use image effects by providing a source image (the content you're starting with), creating an effect such as GaussianBlurEffect, setting properties such as BlurAmount, and then drawing the effect's output with `DrawImage`.

To apply an image effect to your text and shapes, you need to first render that content into a CanvasCommandList. This object is usable as an input to your effect.

2. Change your `canvas_Draw` method to use the following code:

C#

```
CanvasCommandList cl = new CanvasCommandList(sender);

using (CanvasDrawingSession clds = cl.CreateDrawingSession())
{
    for (int i = 0; i < 100; i++)
    {
        clds.DrawText("Hello, World!", RndPosition(), Color.FromArgb(255,
RndByte(), RndByte(), RndByte()));
        clds.DrawCircle(RndPosition(), RndRadius(), Color.FromArgb(255,
RndByte(), RndByte(), RndByte()));
        clds.DrawLine(RndPosition(), RndPosition(), Color.FromArgb(255,
RndByte(), RndByte(), RndByte()));
    }
}
```

Just like how you obtain a `CanvasDrawingSession` from `CanvasDrawEventArgs` which you can draw with, you can create a `CanvasDrawingSession` from a `CanvasCommandList`. The only difference is that when you draw to the command list's drawing session (clds), you are not directly rendering to the **CanvasControl**. Instead, the command list is an intermediate object that stores the results of rendering for later use.

You may have noticed the `using` block that wraps the command list's drawing session. Drawing sessions implement IDisposable and must be disposed when you are done rendering (the `using` block does this). The `CanvasDrawingSession` that you obtain from `CanvasDrawEventArgs` automatically is closed for you, but you must dispose any drawing sessions that you explicitly created.

3. Finally, define the `GaussianBlurEffect` by adding the following code to the end of the `canvas_Draw` method:

C#

```
GaussianBlurEffect blur = new GaussianBlurEffect();
blur.Source = cl;
blur.BlurAmount = 10.0f;
args.DrawingSession.DrawImage(blur);
```

4. Run the app again. You should see your lines, text and circles with a blurry appearance.

Animate your app with CanvasAnimatedControl

. Win2D gives you the ability to update and animate your content in realtime, for example by changing the blur radius of the Gaussian blur with every frame. To do this, you will use CanvasAnimatedControl.

CanvasControl is best suited for mostly static graphics content - it only raises the `Draw` event when your content needs to be updated or redrawn. If you have continually changing content then you should consider using `CanvasAnimatedControl` instead. The two controls operate very similarly, except `CanvasAnimatedControl` raises the `Draw` event on a periodic basis; by default it is called 60 times per second.

2. To switch to `CanvasAnimatedControl`, go to `MainPage.xaml`, delete the **CanvasControl** line, and replace it with the following XAML:

XAML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">
    <canvas:CanvasAnimatedControl x:Name="canvas" Draw="canvas_DrawAnimated"
CreateResources="canvas_CreateResources"/>
</Grid>
```

Just like with **CanvasControl**, let AutoComplete create the `Draw` event handler for you. By default, Visual Studio will name this handler `canvas_Draw_1` because `canvas_Draw`

already exists; here, we've renamed the method `canvas_AnimatedDraw` to make it clear that this is a different event.

In addition, you are also handling a new event, CreateResources. Once again, let AutoComplete create the handler.

Now that your app will be redrawing at 60 frames per second, it is more efficient to create your Win2D visual resources once and reuse them with every frame. It is inefficient to create a `CanvasCommandList` and draw 300 elements into it 60 times per second when the content remains static. `CreateResources` is an event that is fired only when Win2D determines you need to recreate your visual resources, such as when the page is loaded.

3. Switch back to `MainPage.xaml.cs`. Find your `canvas_Draw` method which should look like this:

C#

```
private void canvas_Draw(
    Microsoft.Graphics.Canvas.UI.Xaml.CanvasControl sender,
    Microsoft.Graphics.Canvas.UI.Xaml.CanvasDrawEventArgs args)
{
    CanvasCommandList cl = new CanvasCommandList(sender);
    using (CanvasDrawingSession clds = cl.CreateDrawingSession())
    {
        for (int i = 0; i < 100; i++)
        {
            clds.DrawText("Hello, World!", RndPosition(),
Color.FromArgb(255, RndByte(), RndByte(), RndByte()));
            clds.DrawCircle(RndPosition(), RndRadius(), Color.FromArgb(255,
RndByte(), RndByte(), RndByte()));
            clds.DrawLine(RndPosition(), RndPosition(), Color.FromArgb(255,
RndByte(), RndByte(), RndByte()));
        }
    }

    GaussianBlurEffect blur = new GaussianBlurEffect();
    blur.Source = cl;
    blur.BlurAmount = 10.0f;
    args.DrawingSession.DrawImage(blur);
}
```

The vast majority of this existing draw code does not need to be executed with every frame: the command list containing the text, lines and circles remains the same with every frame, and the only thing that changes is the blur radius. Therefore, you can move this "static" code into `CreateResources`.

To do this, first cut (CTRL+X) the entire contents of `canvas_Draw`, except for the very last line (`args.DrawingSession.DrawImage(blur);`). You can now delete the remainder of `canvas_Draw` as it is no longer needed: recall that `CanvasAnimatedControl` has its own distinct `Draw` event.

4. Find the automatically generated `canvas_CreateResources` method:

```
C#

private void canvas_CreateResources(
    Microsoft.Graphics.Canvas.UI.Xaml.CanvasAnimatedControl sender,
    Microsoft.Graphics.Canvas.UI.CanvasCreateResourcesEventArgs args)
{}
```

Paste (CTRL+V) your previously cut code into this method. Next, move the declaration of `GaussianBlurEffect` outside the method body so the variable becomes a member of the MainPage class. Your code should now look like the following:

```
C#

GaussianBlurEffect blur;
private void canvas_CreateResources(
    Microsoft.Graphics.Canvas.UI.Xaml.CanvasAnimatedControl sender,
    Microsoft.Graphics.Canvas.UI.CanvasCreateResourcesEventArgs args)
{
    CanvasCommandList cl = new CanvasCommandList(sender);
    using (CanvasDrawingSession clds = cl.CreateDrawingSession())
    {
        for (int i = 0; i < 100; i++)
        {
            clds.DrawText("Hello, World!", RndPosition(),
Color.FromArgb(255, RndByte(), RndByte(), RndByte()));
            clds.DrawCircle(RndPosition(), RndRadius(), Color.FromArgb(255,
RndByte(), RndByte(), RndByte()));
            clds.DrawLine(RndPosition(), RndPosition(), Color.FromArgb(255,
RndByte(), RndByte(), RndByte()));
        }
    }

    blur = new GaussianBlurEffect()
    {
        Source = cl,
        BlurAmount = 10.0f
    };
}
```

5. Now you can animate the Gaussian blur. Find the `canvas_DrawAnimated` method and add the following code:

C#

```
private void canvas_DrawAnimated(
    Microsoft.Graphics.Canvas.UI.Xaml.ICanvasAnimatedControl sender,
    Microsoft.Graphics.Canvas.UI.Xaml.CanvasAnimatedDrawEventArgs args)
{
    float radius = (float)(1 + Math.Sin(args.Timing.TotalTime.TotalSeconds))
* 10f;
    blur.BlurAmount = radius;
    args.DrawingSession.DrawImage(blur);
}
```

This reads the total elapsed time provided by CanvasAnimatedDrawEventArgs and uses this to calculate the desired blur amount; the sine function provides an interesting variation over time. Finally, the `GaussianBlurEffect` is rendered.

6. Run the app to see the blurred content change over time.

Congratulations on completing this quick start tutorial! Hopefully you've seen how you can use Win2D to create a rich, animated visual scene with just a few lines of C# and XAML code.

Win2D in a Core App project

Article • 11/20/2023

The Win2D quickstart and Build a simple Win2D app topics both show Win2D being used in projects that use the XAML user-interface (UI) framework. But you can *also* use Win2D in a project that doesn't use XAML at all.

This topic shows you how, in a UWP Core App project, you can use a CanvasSwapChain to display Win2D content. And you can run the resulting app on both Windows and Xbox. This type of app is also referred to as a *CoreApplication app*, or a *UWP DirectX app*.

> **Platform APIs:** IFrameworkViewSource, IFrameworkView, CoreWindow, CoreApplication, CoreApplicationView, CanvasSwapChain, CanvasDrawingSession

Create and configure a new project

We'll be creating a XAML project just to get started. But we'll be removing all of the XAML from it as we proceed.

1. In Visual Studio, create a new project from the **Blank App (Universal Windows)** project template. For language, choose: *C#*; platform: *Windows*; project type: *UWP*.
2. Then, to set up the project to use Win2D, see Reference the Win2D NuGet package.

Modify the project files

1. Since we won't be using XAML, go ahead and delete the following files from the project: `App.xaml`, `App.xaml.cs`, `MainPage.xaml`, and `MainPage.xaml.cs`.
2. And then add to the project a new *Class*; name the file `Program.cs`.

Add code to Program.cs

1. Replace the entire contents of `Program.cs` with the following code listing:

```
C#

// Program.cs
using Microsoft.Graphics.Canvas;
```

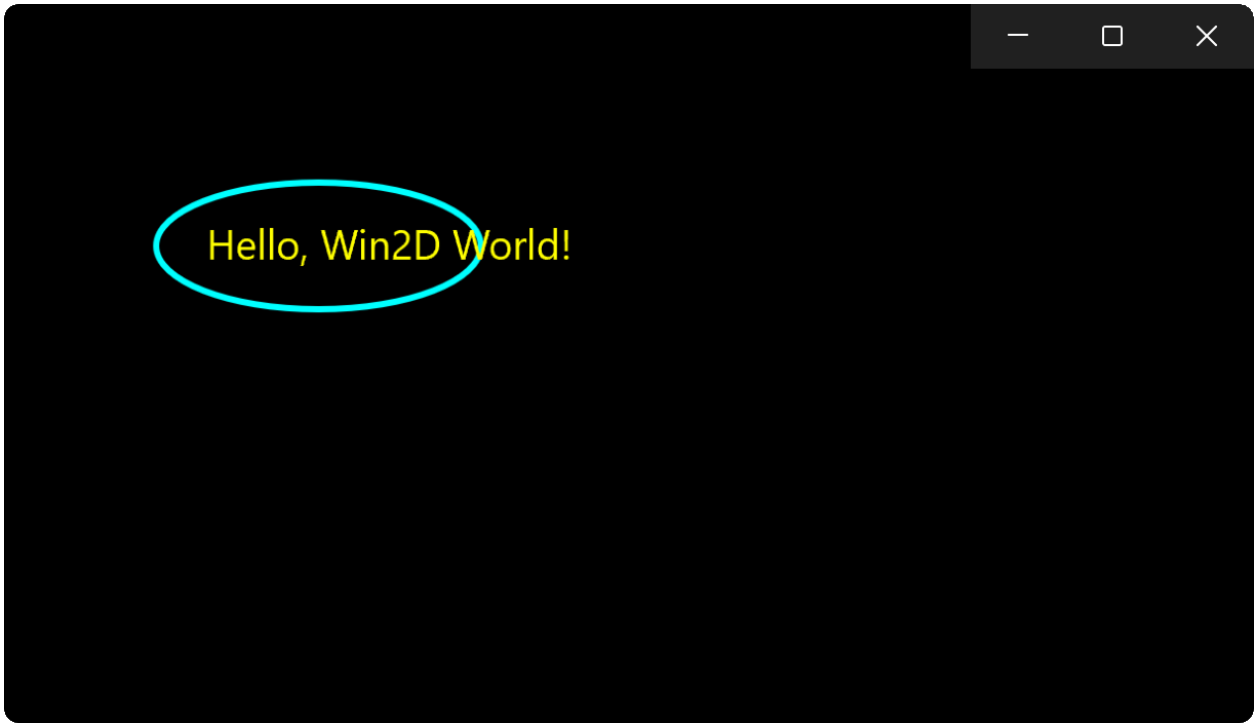
```
using Windows.ApplicationModel.Core;
using Windows.Graphics.Display;
using Windows.UI;
using Windows.UI.Core;

public sealed class App : IFrameworkViewSource, IFrameworkView
{
    private CoreApplicationView view;
    private CoreWindow window;
    private CanvasSwapChain swapChain;

    public IFrameworkView CreateView()
    {
        return this;
    }

    public void Initialize(CoreApplicationView applicationView)
    {
        this.view = applicationView;
    }

    public void SetWindow(CoreWindow window)
    {
        this.view.TitleBar.ExtendViewIntoTitleBar = true;
        this.window = window;
    }

    public void Load(string entryPoint)
    {
    }

    public void Run()
    {
        this.swapChain = CanvasSwapChain.CreateForCoreWindow(
            resourceCreator: CanvasDevice.GetSharedDevice(),
            coreWindow: this.window,
            dpi: DisplayInformation.GetForCurrentView().LogicalDpi);

        this.swapChain.ResizeBuffers((float)this.window.Bounds.Width,
(float)this.window.Bounds.Height);

        using (CanvasDrawingSession session =
this.swapChain.CreateDrawingSession(Colors.Black))
        {
            session.DrawEllipse(155, 115, 80, 30, Colors.Cyan, 3);
            session.DrawText("Hello, Win2D World!", 100, 100,
Colors.Yellow);
        }

        this.swapChain.Present();

        this.window.Activate();

this.window.Dispatcher.ProcessEvents(CoreProcessEventsOption.ProcessUntilQui
t);
```

```
    }

    public void Uninitialize()
    {
    }
}

public static class Program
{
    public static void Main()
    {
        CoreApplication.Run(new App());
    }
}
```

> **Note**
>
> This example includes a separate **Program** class containing the **Main** entry point for the program, which then starts the actual **CoreApplication** app. But it's not necessary to place the entry point in a separate class; you could instead just move the **Main** method inside of the **App** class if you prefer structuring code that way. If you're using C# 10 or above, then the `CoreApplication.Run(new App());` expression statement could also be in a top-level statement.

Finishing up

Now you can go ahead and build and run the app, and see the following output:



In terms of the app model, the code in the previous section has an **App** class that implements the IFrameworkViewSource and IFrameworkView interfaces, which are necessary for a Core App. And the **Main** function entry point calls **CoreApplication.Run** to run an instance of that **App** class.

In terms of the graphics, the code creates a CanvasSwapChain for the main CoreWindow, and draws content into it.

You can extend the sample app above in any way that you like. For example, this could be a good starting point for a simple 2D game.

See Also

- Overview of Win2D

Features

Article • 05/29/2023

Win2D includes an extensive set of features to support lots of different scenarios. Here's a list of most of them, with useful links to related docs and API references to learn more.

Bitmap graphics

- Load, save and draw bitmap images
- Render to texture
- Use bitmaps as opacity masks
- Sprite batch API for efficiently drawing large numbers of bitmaps
- Use block compressed bitmap formats to save memory
- Load, save, and draw virtual bitmaps, which can be larger than the maximum GPU texture size and are automatically split into tiles

Vector graphics

- Draw primitive shapes (lines, rectangles, circles, etc.) or arbitrarily complex geometry
- Fill shapes using solid colors, image brushes, or linear and radial gradients
- Draw lines of any width with flexible stroke styles (dotted, dashed, etc.)
- High quality antialiasing
- Rich geometry manipulation (union, intersect, compute point on path, tessellate, etc.)
- Clip drawing to arbitrary geometric regions
- Capture drawing operations in command lists for later replay
- Rasterize ink strokes (from a stylus)
- Load, draw, and manipulate SVG vector graphics

Powerful image processing effects

- Blurs
- Blends
- Color adjustments (brightness, contrast, exposure, highlights & shadows, etc.)
- Filters (convolve, edge detection, emboss, sharpen)
- Lighting

- Custom pixel shaders
- Fully custom effects
- And many more ...

Text

- Fully internationalized Unicode text rendering
- Text layouts can be drawn, measured, or hit-tested against
- Convert text outlines to geometry
- Enumerate fonts and query their metrics
- Draw or manipulate individual glyph runs to create custom text layouts

Interop with Direct2D

Article • 05/29/2023

Win2D is implemented as a layer on top of Direct2D, and supports interop in both directions. If you have a Win2D object, you can access the native Direct2D object that is used to implement it. If you have a Direct2D object, you can look up the Win2D object that wraps it, or create a new wrapper if one did not already exist.

Interop allows you to mix and match Win2D with native DirectX APIs. You can write an app that mostly uses Win2D, but drop down to native DirectX at any point - perhaps to call into some other API or 3rd party component that requires native interfaces. Or your app can be mostly native DirectX, yet you can switch over to Win2D in specific places where you want its extra convenience or C# support.

Interop APIs

C++/CX interop APIs are defined in the header Microsoft.Graphics.Canvas.native.h:

C++

```
#include <Microsoft.Graphics.Canvas.native.h>

using namespace Microsoft::Graphics::Canvas;
```

To get the native Direct2D object that is wrapped by a Win2D object:

C++

```
template<typename T, typename U>
Microsoft::WRL::ComPtr<T> GetWrappedResource(U^ wrapper);

template<typename T, typename U>
Microsoft::WRL::ComPtr<T> GetWrappedResource(CanvasDevice^ device, U^ wrapper);

template<typename T, typename U>
Microsoft::WRL::ComPtr<T> GetWrappedResource(CanvasDevice^ device, U^ wrapper,
float dpi);
```

For most types `GetWrappedResource` can be called with only a Win2D wrapper object as parameter. For a few types (see below table) it must also be passed a device and/or DPI value. It is not an error to pass a device or DPI when using `GetWrappedResource` with types that do not require them.

To get a Win2D object wrapping a native Direct2D object:

C++

```
template<typename WRAPPER>
WRAPPER^ GetOrCreate(IUnknown* resource);

template<typename WRAPPER>
WRAPPER^ GetOrCreate(CanvasDevice^ device, IUnknown* resource);

template<typename WRAPPER>
WRAPPER^ GetOrCreate(ID2D1Device1* device, IUnknown* resource);

template<typename WRAPPER>
WRAPPER^ GetOrCreate(CanvasDevice^ device, IUnknown* resource, float dpi);

template<typename WRAPPER>
WRAPPER^ GetOrCreate(ID2D1Device1* device, IUnknown* resource, float dpi);
```

`GetOrCreate` returns an existing wrapper instance if one already exists, or creates a new wrapper if one does not. Calling it repeatedly on the same native object will return the same wrapper each time, as long as that wrapper instance continues to exist. If all references to the wrapper are released such that its reference count goes to zero and it is destroyed, any later call to `GetOrCreate` will have to create a new wrapper.

For some types `GetOrCreate` can be called with only a Direct2D resource object as parameter, while for other types (see below table) it must also be passed a device and DPI value. It is not an error to pass a device or DPI when using `GetOrCreate` with types that do not require them. If a Win2D wrapper already exists, it is ok to omit the device and DPI even for types that would normally need them: these parameters are only used when creating new wrapper instances.

`GetOrCreate` understands inheritance hierarchies and will always create the appropriate most-derived wrapper type. For instance if you call `GetOrCreate<CanvasBitmap>(ID2D1Bitmap1*)` with an `ID2D1Bitmap1` that has the `D2D1_BITMAP_OPTIONS_TARGET` flag, the returned wrapper instance will in fact be a `CanvasRenderTarget` (which derives from `CanvasBitmap`). The other way around, if you call `GetOrCreate<CanvasRenderTarget>(ID2D1Bitmap1*)` with an `ID2D1Bitmap1` that does not have `D2D1_BITMAP_OPTIONS_TARGET`, this will throw an invalid cast exception.

Taking this to the extreme, it is valid to call `GetOrCreate<Object>(IUnknown*)`, and also `GetWrappedResource<IUnknown>(Object^)`.

Types that support interop

Win2D type	Direct2D type	`GetOrCreate` parameters	`GetWrappedResource` parameters

Win2D type	Direct2D type	`GetOrCreate` parameters	`GetWrappedResource` parameters
CanvasBitmap	`ID2D1Bitmap1` without `D2D1_BITMAP_OPTIONS_TARGET`	Device	-
CanvasCachedGeometry	`ID2D1GeometryRealization`	Device	-
CanvasCommandList	`ID2D1CommandList`	Device	-
CanvasDevice	`ID2D1Device1`	-	-
CanvasDrawingSession	`ID2D1DeviceContext1`	-	-
CanvasFontFace	`IDWriteFontFaceReference`	-	-
CanvasFontSet	`IDWriteFontSet`	-	-
CanvasGeometry	`ID2D1Geometry`, or one of its derived interfaces `ID2D1PathGeometry`, `ID2D1RectangleGeometry`, `ID2D1RoundedRectangleGeometry`, `ID2D1EllipseGeometry`, `ID2D1TransformedGeometry`, or `ID2D1GeometryGroup`	Device	-
CanvasGradientMesh	`ID2D1GradientMesh`	Device	-
CanvasImageBrush	`ID2D1BitmapBrush1` (if the image is a `CanvasBitmap` and `SourceRectangle` is `null`) or `ID2D1ImageBrush` (if it is any other type of `ICanvasImage`, or if `SourceRectangle is set`)	Device	Optional DPI^1
CanvasLinearGradientBrush	`ID2D1LinearGradientBrush`	Device	-
CanvasNumberSubstitution	`IDWriteNumberSubstitution`	-	-
CanvasRadialGradientBrush	`ID2D1RadialGradientBrush`	Device	-
CanvasRenderTarget	`ID2D1Bitmap1` with `D2D1_BITMAP_OPTIONS_TARGET`	Device	-
CanvasSolidColorBrush	`ID2D1SolidColorBrush`	Device	-
CanvasStrokeStyle	`ID2D1StrokeStyle1`	-	Device
CanvasSvgDocument	`ID2D1SvgDocument`^2	Device	-
CanvasSwapChain	`IDXGISwapChain1`	Device, DPI	-

Win2D type	Direct2D type	`GetOrCreate` parameters	`GetWrappedResource` parameters
CanvasTestFormat	`IDWriteTextFormat1`	-	-
CanvasTextLayout	`IDWriteTextLayout3`	Device	-
CanvasTextRenderingParameters	`IDWriteRenderingParams3`	-	-
CanvasTypography	`IDWriteTypography`	-	-
CanvasVirtualBitmap	`ID2D1ImageSource` or `ID2D1TransformedImageSource`	Device	-
ColorManagementProfile	`ID2D1ColorContext`	Device	Device
EffectTransferTable3D	`ID2D1LookupTable3D`	Device	-
Microsoft.Graphics.Canvas.Effects.* (multiple Win2D classes map to the same D2D type)	`ID2D1Effect` with the appropriate `D2D1_PROPERTY_TYPE_CLSID`	Device	Device, optional DPI^1

> **Note**
>
> ^1: optional DPI means it is valid to call `GetWrappedResource` for this type without specifying a DPI value, but if you do specify DPI, Win2D may be able to more efficiently configure effect graphs by leaving out redundant DPI compensation nodes. This applies when calling `GetWrappedResource` on an effect, or on a `CanvasImageBrush` that has an effect as its source image.

> **Note**
>
> ^2: when a `CanvasSvgDocument` is produced from an `ID2D1SvgDocument` using native C++ interop, the `ID2D1SvgDocument`'s viewport size is ignored.

Interop using C++/CX

Here's an example of how Win2D can interoperate with Direct2D, using C++:

C++

```
#include <Microsoft.Graphics.Canvas.native.h>
#include <d2d1_2.h>

using namespace Microsoft::Graphics::Canvas;
```

```
using namespace Microsoft::WRL;

// Interop Win2D -> Direct2D.
CanvasDevice^ canvasDevice = ...;
CanvasBitmap^ canvasBitmap = ...;

ComPtr<ID2D1Device> nativeDevice = GetWrappedResource<ID2D1Device>
(canvasDevice);
ComPtr<ID2D1Bitmap1> nativeBitmap = GetWrappedResource<ID2D1Bitmap1>
(canvasBitmap);

// Interop Direct2D -> Win2D.
canvasDevice = GetOrCreate<CanvasDevice>(nativeDevice.Get());
bitmap = GetOrCreate<CanvasBitmap>(canvasDevice, nativeBitmap.Get());
```

> **Note**
>
> Interop is also possible via C#, through various means (eg. using built-in COM/WinRT interop, via **CsWinRT** 's APIs, or using blittable bindings). For more info on manual interop in C#, refer to the docs about CsWinRT.

Interop using C++/WinRT

You can also perform interop using C++/WinRT with some modification of the above. Note that the C++/WinRT headers for the Win2D Windows Runtime Components should be generated automatically when you add the Win2D NuGet package to your C++/WinRT project. However, for interop you will still need to use the header file Microsoft.Graphics.Canvas.native.h which contains the low-level ABI interface `ICanvasFactoryNative` in the namespace `ABI::Microsoft::Graphics::Canvas`. The interface has the following functions which you can use to perform interop.

C++

```
HRESULT GetOrCreate(ICanvasDevice* device, IUnknown* resource, float dpi,
IInspectable** wrapper);
HRESULT GetNativeResource(ICanvasDevice* device, float dpi, REFIID iid, void**
resource);
```

Here is an example showing how to create a `CanvasVirtualBitmap` from an `IWICBitmapSource` starting with the `IWICBitmapSource` and the shared `CanvasDevice`.

C++

```
#include "pch.h"
#include <wincodec.h>
#include <wincodecsdk.h>
```

```
#include <winrt/Microsoft.Graphics.Canvas.h> //This defines the C++/WinRT
interfaces for the Win2D Windows Runtime Components
#include <Microsoft.Graphics.Canvas.h> //This defines the low-level ABI
interfaces for the Win2D Windows Runtime Components
#include <Microsoft.Graphics.Canvas.native.h> //This is for interop
#include <d2d1_3.h>

using namespace winrt::Microsoft::Graphics::Canvas;
namespace abi {
  using namespace ABI::Microsoft::Graphics::Canvas;
}

namespace winrt::Win2DInteropTest::implementation {
  CanvasVirtualBitmap
CreateVirtualBitmapFromBitmapSource(com_ptr<IWICBitmapSource> const&
pBitmapSource){
    CanvasDevice sharedDevice = CanvasDevice::GetSharedDevice();

    //First we need to get an ID2D1Device1 pointer from the shared CanvasDevice
    com_ptr<abi::ICanvasResourceWrapperNative> nativeDeviceWrapper =
sharedDevice.as<abi::ICanvasResourceWrapperNative>();
    com_ptr<ID2D1Device1> pDevice{ nullptr };
    check_hresult(nativeDeviceWrapper->GetNativeResource(nullptr, 0.0f,
guid_of<ID2D1Device1>(), pDevice.put_void()));

    //Next we need to call some Direct2D functions to create the
ID2D1ImageSourceFromWic object
    com_ptr<ID2D1DeviceContext1> pContext{ nullptr };
    check_hresult(pDevice-
>CreateDeviceContext(D2D1_DEVICE_CONTEXT_OPTIONS_NONE, pContext.put()));
    com_ptr<ID2D1DeviceContext2> pContext2 = pContext.as<ID2D1DeviceContext2>
();
    com_ptr<ID2D1ImageSourceFromWic> pImage{ nullptr };
    check_hresult(pContext2->CreateImageSourceFromWic(pBitmapSource.get(),
D2D1_IMAGE_SOURCE_LOADING_OPTIONS_RELEASE_SOURCE, pImage.put()));

    //Finally we need to wrap the ID2D1ImageSourceFromWic object inside
    com_ptr<::IInspectable> pInspectable{ nullptr };
    auto factory = winrt::get_activation_factory<CanvasDevice,
abi::ICanvasFactoryNative>(); //abi::ICanvasFactoryNative is the activation
factory for the CanvasDevice class
    check_hresult(factory->GetOrCreate(sharedDevice.as<abi::ICanvasDevice>
().get(), pImage.as<::IUnknown>().get(), 0.0f, pInspectable.put())); //Note
abi::ICanvasDevice is defined in the header Microsoft.Graphics.Canvas.h
    CanvasVirtualBitmap cvb = pInspectable.as<CanvasVirtualBitmap>();
    return cvb;
  }
}
```

Remember to include the header `<unknwn.h>` in pch.h before any WinRT headers (required in SDK 17763 and later).

Offscreen drawing

Article • 06/29/2024

Apps occasionally need to draw graphics to a target, where that target is not intended for immediate display. This type of drawing is sometimes called "offscreen rendering", or "drawing to a texture". This is useful when, for example, an app's output of a drawing operation is to be saved to a file, returned as an array of pixels, or used as an input to a later operation.

Win2D supports these scenarios, and they are made easy with CanvasRenderTarget.

`CanvasRenderTarget` extends CanvasBitmap, and has the method CreateDrawingSession(). Use `CreateDrawingSession` to draw graphics content to a `CanvasRenderTarget`. For example:

C#

```
CanvasDevice device = CanvasDevice.GetSharedDevice();
CanvasRenderTarget offscreen = new CanvasRenderTarget(device, width, height,
96);

using (CanvasDrawingSession ds = offscreen.CreateDrawingSession())
{
    ds.Clear(Colors.Black);
    ds.DrawRectangle(100, 200, 5, 6, Colors.Red);
}
```

Note that there is a method call to `Clear`. Without this, the bitmap will be initialized with undefined content. Drawing sessions created through `CanvasRenderTarget` are different from those created on Win2D's XAML controls, in terms of the `Clear` behavior. Controls are always cleared automatically by Win2D when a drawing session is created. `CanvasRenderTargets` are not. This way, apps have the ability to make incremental changes to `CanvasRenderTarget`-s, and avoid redrawing an entire scene every time.

To draw a `CanvasRenderTarget` to another drawing session, simply use `DrawImage(ICanvasImage)` or one of its overloads. For example:

C#

```
void canvasControl_Draw(CanvasControl sender, CanvasDrawEventArgs args)
{
    args.DrawingSession.DrawImage(offscreen, 23, 34);
}
```

Or, to use a `CanvasRenderTarget` as an input to an effect, pass it in wherever the effect expects to use an `IGraphicsEffectSource` as a source. For example:

```
C#

GaussianBlurEffect blurEffect = new GaussianBlurEffect()
{
    Source = offscreen,
    BlurAmount = 3.0f
};
```

An app can close, and re-open drawing sessions on a `CanvasRenderTarget` arbitrarily many times.

Drawing operations are not committed to the `CanvasRenderTarget` until the drawing session object is disposed. In C#, a `using` block can organize this.

It's worth pointing out that `CanvasRenderTarget` is not a XAML control, and does not involve the XAML tree at all. It is suitable for both XAML and non-XAML-based apps.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

DPI and DIPs

Article • 05/29/2023

This article explains the difference between physical pixels and device independent pixels (DIPs), and how DPI (dots per inch) is handled in Win2D.

Win2D is designed in such a way that many apps can ignore this distinction, as it provides sensible default behaviors that will do the right thing when run on both low and high DPI devices. If your app has more specialized needs, or if you have a different opinion about what "sensible default" means, read on for the gory details...

What is DPI?

DPI stands for "dots per inch". This is an approximate measure of the pixel density of an output display such as a computer monitor or phone screen. The higher the DPI, the more, smaller dots make up the display.

DPI is only an approximate measure because not all display hardware can be relied on to report accurate information. Some computer monitors do not report DPI to the operating system at all, or the user may have configured their system to render using a different DPI from the actual hardware (for instance to change the size of UI text elements). Applications can use DPI to choose how large things should be drawn, but should not rely on it as an exact physical measurement of the display size.

A DPI value of 96 is considered to be a neutral default.

What is a pixel?

A pixel is a single colored dot. Images in computer graphics are made up of many pixels arranged in a two dimensional grid. You can think of pixels as the atoms out of which all images are built.

The physical size of a pixel can vary greatly from one display to another. When a computer is connected to a large but low resolution monitor or external display, pixels can be quite large, but on a phone with a 1080p display only a few inches across, pixels are tiny.

In Win2D, whenever you see an API that specifies a position or size using integer data types (or a struct such as BitmapSize that contains integers), this means the API is operating in pixel units.

Most Win2D APIs work with DIPs rather than pixels.

What is a DIP?

DIP stands for "device independent pixel". This is a virtualized unit that may be the same as, larger, or smaller than a physical pixel.

The ratio between pixels and DIPs is determined by DPI:

```
pixels = dips * dpi / 96
```

When DPI is 96, pixels and DIPs are the same. When using higher DPI, a single DIP may correspond to more than one pixel (or parts of pixels in the common case where DPI is not an exact multiple of 96).

Most Windows Runtime APIs, including Win2D, use DIPs rather than pixels. This has the advantage of keeping graphics approximately the same physical size no matter what display an app is run on. For instance if an app specifies that a button is 100 DIPs wide, when run on a high DPI device such as a phone or 4k monitor this button will automatically scale to be more than 100 pixels in width, so it remains a sensible size that is possible for the user to click on. If the button size was specified in pixels, on the other hand, it would appear ridiculously small on this kind of high DPI display, so the app would have to do more work to adjust layouts differently for each kind of screen.

In Win2D, whenever you see an API that specifies a position or size using floating point data types (or structs such as Vector2 or Size that contain floating point values), this means the API is operating in DIPs.

To convert between DIPs and pixels, use the methods ConvertDipsToPixels(Single, CanvasDpiRounding) and ConvertPixelsToDips(Int32).

Win2D resources that have DPI

All Win2D resources that contain a bitmap image also have an associated DPI property:

- CanvasBitmap
- CanvasRenderTarget
- CanvasSwapChain
- CanvasControl
- CanvasVirtualControl

- CanvasAnimatedControl
- CanvasImageSource

All other resource types are independent of DPI. For instance a single CanvasDevice instance can be used to draw to controls or rendertargets of many different DPIs, therefore the device has no DPI of its own.

Similarly, CanvasCommandList does not have a DPI, because it contains vector drawing instructions rather than a bitmap image. DPI only comes into play during the rasterization process, when the command list is drawn to a rendertarget or control (which do have DPI).

Control DPI

The Win2D controls (CanvasControl, CanvasVirtualControl and CanvasAnimatedControl) automatically use the same DPI as the display the app is running on. This matches the coordinate system used by XAML, CoreWindow, and other Windows Runtime APIs.

If the DPI changes (for instance if the app is moved to a different display), the control will raise the `CreateResources` event and pass a CanvasCreateResourcesReason of DpiChanged. Apps should respond to this event by recreating any resources (such as rendertargets) that depend on the DPI of the control.

Rendertarget DPI

Things that can be drawn onto (which includes not just CanvasRenderTarget but also the rendertarget-like types CanvasSwapChain and CanvasImageSource) have a DPI of their own, but unlike the controls these types are not directly connected to a display, so Win2D cannot automatically determine what the DPI should be. If you are drawing to a rendertarget which will later be copied to the screen, you probably want that rendertarget to use the same DPI as the screen, but if you are drawing for some other purpose (eg. generating images for upload to a website) a default 96 DPI would be more appropriate.

To make both these usage patterns easy, Win2D provides two types of constructor overload:

```
CanvasRenderTarget(ICanvasResourceCreator, width, height, dpi)
CanvasRenderTarget(ICanvasResourceCreatorWithDpi, width, height)
```

The ICanvasResourceCreator interface is implemented by `CanvasDevice` as well as the Win2D controls. Because a device does not have any specific DPI of its own, you must explicitly specify the DPI when creating a rendertarget from one.

For instance to create a default DPI rendertarget where DIPs and pixels will always be the same thing:

```
C#

const float defaultDpi = 96;
var rtWithFixedDpi = new CanvasRenderTarget(canvasDevice, width, height,
defaultDpi);
```

ICanvasResourceCreatorWithDpi extends `ICanvasResourceCreator` by adding a DPI property. This interface is implemented by the Win2D controls, and makes it easy to create a rendertarget which will automatically inherit the same DPI as the control it was created from:

```
C#

var rtWithSameDpiAsDisplay = new CanvasRenderTarget(canvasControl, width,
height);
```

Bitmap DPI

CanvasBitmap, unlike a rendertarget, does not automatically inherit DPI from a control. The methods for creating and loading bitmaps include overloads to explicitly specify DPI, but if you leave this out, bitmap DPI defaults to 96 regardless of the current display configuration.

The reason bitmaps are different to other types is that they are a source of input data, rather than an output which will be drawn onto. So the important thing for bitmaps is not the DPI of where that output will end up, but the DPI of the source image, which is entirely unrelated to the current display settings.

If you load say a 100x100 default DPI bitmap and then draw it onto a rendertarget, the bitmap will be scaled from 100 DIPs at 96 DPI (which is 100 pixels) to 100 DIPs at the DPI of the destination rendertarget (which could be a larger number of pixels if it is a high DPI rendertarget). The resulting image will always be 100 DIPs in size (so there will be no unpleasant layout surprises), but it may suffer some blurring if a low DPI source bitmap was scaled up to a higher DPI destination.

For maximum clarity at high DPI, some applications may wish to provide multiple sets of bitmap images at different resolutions, and at load time select whichever version most closely matches the DPI of the destination control. Other apps may prefer to ship only high DPI bitmaps, and let Win2D scale these down when running on lower DPI displays (scaling down can often look better than scaling up). In either case, the bitmap DPI can be specified as a parameter to LoadAsync(ICanvasResourceCreator, String, Single).

Note that some bitmap file formats contain DPI metadata of their own, but Win2D ignores this since it is often set incorrectly. Instead, DPI must be explicitly specified when loading the bitmap.

CanvasDrawingSession DPI

CanvasDrawingSession inherits its DPI from whatever control, rendertarget, swapchain, etc, it is drawing onto.

By default, all drawing operations operate in DIPs. If you prefer to work in pixels, this can be changed via the Units property.

Effect DPI

The image effect pipeline inherits its DPI from whatever CanvasDrawingSession an effect is being drawn onto. Internally, effect processing always operates in pixels. Parameter values such as sizes or positions are specified in DIPs, but these units are converted to pixels before any actual image manipulation takes place.

When a bitmap of different DPI than the target drawing session is used as an effect source image, an internal DpiCompensationEffect is automatically inserted in between the bitmap and the effect. This scales the bitmap to match the target DPI, which is usually what you want. If it's not what you want, you can insert your own instance of DpiCompensationEffect to customize the behavior.

> **Note**
>
> If implementing a custom effect, consider applying an equivalent DPI handling scheme to ensure consistent behavior when used together with built-in Win2D effects.

Composition API

The `Microsoft.Graphics.Canvas.Composition` APIs operate at a lower level than Win2D XAML controls, so they do not attempt to automatically handle DPI on your behalf. It is up to you to decide what units you prefer to operate in, and set whatever transforms are necessary to achieve that as part of your composition visual tree.

`Windows.UI.Composition` APIs such as `CreateDrawingSurface` always specify sizes in pixel units. When using Win2D to draw onto a composition surface, you can specify whatever DPI you want to use when calling CreateDrawingSession(CompositionDrawingSurface, Rect, Single). All drawing performed through the returned `CanvasDrawingSession` will be scaled up or down accordingly.

How to test DPI handling

The easiest way to test that your app will do the right thing in response to changing display DPI is to run on Windows 10 or Windows 11 and change display settings while the app is running:

- Right-click on the desktop background and choose 'Display settings'
- Move the slider labeled 'Change the size of text, apps, and other items'
- Click the 'Apply' button
- Choose 'Sign out later'

If you do not have Windows 10 or Windows 11, you can also test with the Windows Simulator. In the Visual Studio toolbar, change the "Local Machine" setting to "Simulator", then use the Change Resolution icon to switch the simulated display between:

- 100% (DPI = 96)
- 140% (DPI = 134.4)
- 180% (DPI = 172.8)

Choosing control resolution

Article • 05/29/2023

This article explains how to configure the resolution used by Win2D's XAML controls. It explains how to:

- Make Win2D controls run at a fixed resolution.
- Adjust control DPI to improve performance by rendering fewer pixels.

Resolution and control sizing

"Resolution", as used in this document, is another word for the size of a bitmap. It consists of a width, and height.

The objects that Win2D's XAML controls draw to have a resolution. They also have a DPI. An object's DPI is a measurement of how dense the pixels of that object are, when drawn. DPI behaves like a scale factor- a high DPI increases the number of pixels that comprise the drawn object. On the other hand, lowering the DPI of an object means that it will span fewer pixels. For more information about Win2D's handling of DPI in general, see this page.

DPI-independent size is sometimes called "logical size". And a DPI-dependent size, in pixels, is called "physical size".

In terms of resolution and sizing, a control's default behavior when it's loaded is:

- The control's logical size is determined by its layout, as determined by where it falls in the XAML tree.
- A DPI is queried from the environment. The control's DPI is set to that.
- The amount of physical pixels that comprise the control's drawable area is determined by the control's size, scaled by its DPI.
  - On high DPI, the physical size will be greater (more pixels) compared to the logical size.
  - On low DPI, the physical size will be smaller (fewer pixels) compared to the logical size.
  - On default DPI, the physical size and logical size of the drawable area are the same.
- The control's drawing resource (`CanvasImageSource` for `CanvasControl`, `CanvasVirtualImageSource` for `CanvasVirtualControl` and `CanvasSwapChain` for `CanvasAnimatedControl`) is set to match the size and DPI of the control.

Most Win2D operations are in dips (DPI-independent units), and Win2D's XAML controls' drawing resources are automatically sized to take DPI into account. This means applications can often ignore DPI. Sizes and co-ordinates are always DPI-independent unless specified otherwise. An application can hard-code various sizes and co-ordinates at which things are drawn into the controls, and that content gets scaled when the app is run in environments with different DPIs.

But for some applications, the default behavior isn't sufficient. This article outlines a couple scenarios where the default is not sufficient, and what apps can do to fix it.

Scenario: the control's contents must be a fixed, lower-than-normal resolution

This scenario may arise, for instance, on a 2D sprite game that should always render at a fixed 640x480 resolution, regardless of what actual display hardware it is run on.

Solving this doesn't strictly require writing any new Win2D code at all.

The Viewbox XAML object lets you constrain the sizes of its child visual elements, automatically adding scaling, with letterboxing or pillarboxing to preseve aspect ratios as necessary.

Simply ensure your `CanvasControl`, `CanvasVirtualControl` or `CanvasAnimatedControl` is a child element of a `ViewBox`, and restrict the size of that control.

For example, to constrain the size of the control to 320 pixels wide, and 224 pixels high, regardless of DPI, then instead of:

XAML

```
<canvas:CanvasAnimatedControl/>
```

Use:

XAML

```
<Viewbox>
    <canvas:CanvasAnimatedControl  Width="320" Height="224"/>
</Viewbox>
```

If your app should not preserve the aspect ratio by adding letterboxing/pillarboxing, then add the `Stretch` attribute:

XAML

```
<Viewbox Stretch="Fill">
    <canvas:CanvasAnimatedControl Width="320" Height="224"/>
</Viewbox>
```

Note that the scaling performed by the `Viewbox` element does not guarantee any control over the interpolation mode. The filtering method may look like `CanvasInterpolationMode.Linear`, or something similar. If your app needs a particular interpolation mode, then don't use `ViewBox` with a fixed-size control. Instead, draw to an intermediate, fixed-size `CanvasRenderTarget`, and use the desired interpolation mode to draw the scaled intermediate to the target.

Scenario: the app cannot perform well at high resolutions

Some devices have very high-resolution displays, but their graphics processing unit is not powerful enough to animate that many pixels smoothly. Developers may not be readily aware of how their apps perform on these devices without testing them.

One option is to use the `DpiScale` property of the control to reduce the control's DPI.

For example, to fix the control at half-resolution, use:

XAML

```
<canvas:CanvasAnimatedControl DpiScale="0.5" />
```

The actual DPI scale factor depends upon the needs of your app. One option is to compute a scale factor that will fix the app's DPI at 96, and no higher.

For example:

C#

```
float dpiLimit = 96.0f;

if(control.Dpi > dpiLimit)
{
    control.DpiScale = dpiLimit / control.Dpi;
}
```

To ensure this setting works across DPI changes, the application should subscribe to DisplayInformation.DpiChanged and use this logic in the handler to set the DPI scale against the new DPI.

This saves the app some perf overhead, exploiting the fact that users may not be able to easily percieve the reduced resolution on a high-DPI display.

The scaling performed in having a lower-than-native resolution control resource cannot guarantee control over the interpolation mode, similar to `ViewBox` mentioned above. If your app needs a particular interpolation mode, use an intermediate instead.

Pixel formats

Article • 05/29/2023

The [`DirectXPixelFormat`](https://msdn.microsoft.com/library/windows/apps/windows.graphics.directx.directxpixelformat.aspx) enum includes all the many and varied pixel formats used by Direct3D and DXGI, but only a few of these options are supported by Win2D (or by Direct2D upon which Win2D is built).

If in doubt, pixel format `B8G8R8A8UIntNormalized` and CanvasAlphaMode.Premultiplied are good defaults for most purposes.

All the formats listed below are supported by Direct3D feature level 11 GPU hardware (used in desktop computers and most tablets). Feature level 9 GPUs (which are found in phones) only support a subset.

> **Note**
>
> If you want to use one of the formats marked as "Not supported on all devices", you should first check **IsPixelFormatSupported(DirectXPixelFormat)**, or catch exceptions if the resource creation fails and be prepared to fall back on one of the universally available options.

CanvasBitmap formats

`DirectXPixelFormat`	**Compatible CanvasAlphaMode-s**	**Notes**
`B8G8R8A8UIntNormalized`	`Premultiplied`, `Ignore`	The default format
`B8G8R8A8UIntNormalizedSrgb`	`Premultiplied`, `Ignore`	Not supported on all devices
`B8G8R8X8UIntNormalized`	`Ignore`	
`R8G8B8A8UIntNormalized`	`Premultiplied`, `Ignore`	
`R8G8B8A8UIntNormalizedSrgb`	`Premultiplied`, `Ignore`	Not supported on all devices
`R16G16B16A16Float`	`Premultiplied`, `Ignore`	Not supported on all devices
`R32G32B32A32Float`	`Premultiplied`, `Ignore`	Not supported on all devices

`DirectXPixelFormat`	**Compatible CanvasAlphaMode - s**	**Notes**
`R10G10B10A2UIntNormalized`	`Premultiplied`, `Ignore`	Not supported on all devices
`R16G16B16A16UIntNormalized`	`Premultiplied`, `Ignore`	Not supported on all devices
`R8G8UIntNormalized`	`Ignore`	Not supported on all devices. Bitmaps of this format can be used as effect sources, but are not directly drawable.
`R8UIntNormalized`	`Ignore`	Not supported on all devices. Bitmaps of this format can be used as effect sources, but are not directly drawable.
`A8UIntNormalized`	`Premultiplied`, `Straight`	Bitmaps of this format can be used as opacity masks or effect sources, but are not directly drawable.
`BC1UIntNormalized`	`Premultiplied`, `Ignore`	
`BC2UIntNormalized`	`Premultiplied`, `Ignore`	
`BC3UIntNormalized`	`Premultiplied`, `Ignore`	

CanvasRenderTarget formats

`DirectXPixelFormat`	**Compatible CanvasAlphaMode - s**	**Notes**
`B8G8R8A8UIntNormalized`	`Premultiplied`, `Ignore`	The default format
`B8G8R8A8UIntNormalizedSrgb`	`Premultiplied`, `Ignore`	Not supported on all devices
`R8G8B8A8UIntNormalized`	`Premultiplied`, `Ignore`	
`R8G8B8A8UIntNormalizedSrgb`	`Premultiplied`, `Ignore`	Not supported on all devices
`R16G16B16A16Float`	`Premultiplied`, `Ignore`	Not supported on all devices
`R32G32B32A32Float`	`Premultiplied`, `Ignore`	Not supported on all devices
`R16G16B16A16UIntNormalized`	`Premultiplied`, `Ignore`	Not supported on all devices

`DirectXPixelFormat`	**Compatible CanvasAlphaMode -s**	**Notes**
`A8UIntNormalized`	`Premultiplied`, `Straight`	

CanvasSwapChain formats

`DirectXPixelFormat`	**Compatible CanvasAlphaMode -s**	**Notes**
`B8G8R8A8UIntNormalized`	`Premultiplied`, `Ignore`	The default format
`R8G8B8A8UIntNormalized`	`Premultiplied`, `Ignore`	
`R16G16B16A16Float`	`Premultiplied`, `Ignore`	Not supported on all devices

Premultiplied alpha

Article • 05/29/2023

In computer graphics there are two different ways to represent the opacity of a color value. Win2D uses both methods. This article explains the difference, and which is used where.

Straight alpha

When using straight, also known as linear, alpha:

- RGB values specify the color of the thing being drawn
- The alpha value specifies how solid it is

In this world, RGB and alpha are independent. You can change one without affecting the other. To make an object fade out, you would gradually reduce its alpha value while leaving RGB unchanged.

To perform a source-over blend between two colors that use straight alpha format:

```
result = (source.RGB * source.A) + (dest.RGB * (1 - source.A))
```

Premultiplied alpha

When using premultiplied alpha:

- RGB specifies how much color the thing being drawn contributes to the output
- The alpha value specifies how much it obscures whatever is behind it

In this world, RGB and alpha are linked. To make an object transparent you must reduce both its RGB (to contribute less color) and also its alpha (to obscure less of whatever is behind it). Fully transparent objects no longer have any color at all, so there is only one value that represents 100% transparency: RGB and alpha all zero.

To perform a source-over blend between two colors that use premultiplied alpha format:

```
result = source.RGB + (dest.RGB * (1 - source.A))
```

Premultiplied alpha is used in graphics rendering because it gives better results than straight alpha when filtering images or composing different layers. For more information see these blog posts:

- Premultiplied alpha
- Premultiplied alpha and image composition

Alpha in Win2D

Win2D uses straight alpha in its API surface, but premultiplied alpha for internal rendering operations.

`Windows.UI.Color` values use straight alpha. Whenever you pass a color to a `Draw*` or `Fill*` method, set the color of a brush, or clear to a color value, this color is specified using straight alpha.

The pixel values stored in a bitmap or rendertarget, and the drawing or blending operations that operate on these surfaces, use premultiplied alpha. When bitmaps are loaded from a file their contents are automatically converted into premultiplied format. When you call a Win2D drawing method, its color parameter is converted from straight to premultiplied before the actual drawing takes place.

Win2D image effects use a mixture of straight and premultiplied alpha. Some effects operate on one format, some on the other, and some provide a property to choose. The documentation for each effect type describes which alpha mode it uses. Effect input data is always assumed to be premultiplied, so when an effect needs to work with straight alpha it will first apply an unpremultiply transform, compute the effect, and then re-premultiply the output.

The bitmap APIs `GetPixelBytes`, `SetPixelBytes`, `GetPixelColors`, and `SetPixelColors`, do **not** perform any alpha format conversions. They just directly transfer bit values to or from the underlying GPU texture. This allows you to observe what alpha format Win2D is using internally:

- Create a drawing session on a rendertarget
- Call `drawingSession.Clear(Colors.Tranparent)`
- `Colors.Tranparent` is defined as R = 255, G = 255, B = 255, A = 0
- Win2D will convert this value to premultiplied format, yielding R = 0, G = 0, B = 0, A = 0
- Use `GetPixelColors` to read back the contents of the rendertarget
- Observe that it contains premultiplied format RGB = 0, not RGB = 255 like the original straight alpha `Colors.Tranparent` value

Converting between alpha formats

To convert a straight alpha color value to premultiplied format, multiply its R, G, and B values by A. To convert premultiplied to straight, divide R, G, and B by A.

Note that color information is often represented as byte values ranging from 0 to 255 (for example the `Windows.UI.Color` structure consists of 4 bytes). This representation is scaled up by a factor of 255, so a byte value of 255 actually means 1, while 128 is half intensity. That scaling factor must be taken into account during format conversions, so to convert a `Windows.UI.Color` from straight to premultiplied:

C#

```
premultiplied.R = (byte)(straight.R * straight.A / 255);
premultiplied.G = (byte)(straight.G * straight.A / 255);
premultiplied.B = (byte)(straight.B * straight.A / 255);
premultiplied.A = straight.A;
```

If you have image data that is using the wrong alpha format, PremultiplyEffect or UnPremultiplyEffect can be used to convert it.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Bitmap block compression

Article • 05/29/2023

`CanvasBitmap` supports block compressed bitmaps. These can be loaded from a DDS file, or created with CreateFromBytes(ICanvasResourceCreator,Byte[], Int32, Int32, DirectXPixelFormat).

Block compressed bitmaps are great for bitmap heavy applications (such as games) because they take up less memory and can be drawn more efficiently. A block compressed bitmap uses up to 1/8th of the memory of an uncompressed bitmap. As a result, the GPU needs to access much less memory when drawing the bitmap, resulting in faster drawing.

About block compression

Block compression is different to the compression employed by PNG or JPG files. Compressed file formats are stored compressed but when they are loaded by Win2D they are decompressed into a bitmap with a format such as `DirectXPixelFormat::B8G8R8A8UIntNormalized`, which uses 32 bits (4 bytes) per pixel. So a 256x256 bitmap would take up 256 * 256 * 4 = 262,144 bytes.

A block compressed bitmap uses 8 or 16 bytes (depending on the format -- more on this later) to store a block of 4x4 pixels. So in this case a 256x256 bitmap would take up 256 * 256 / (4 * 4) * 8 = 32,768 bytes, or 256 * 256 / (4 * 4) * 16 = 65,536 bytes. That's up to 8 times smaller! Since block compression is supported directly by the GPU hardware, the bitmap can be kept compressed in memory and drawn directly from the compressed format without ever having to completely uncompress it.

Win2D supports three block compressed formats. The table below describes these formats, along with an uncompressed format for comparison.

`DirectXPixelFormat`	**Size of 4x4 bitmap**	**Size of 256x256 bitmap**	**Alpha**
`BC1Unorm`	8 bytes	32,768 bytes	1 bit
`BC2Unorm`	16 bytes	65,536 bytes	4 bit
`BC3Unorm`	16 bytes	65,536 bytes	~8 bit (compressed)
`B8G8R8A8UintNormalized`	64 bytes	262,144 bytes	8 bit

`BC1Unorm`, `BC2Unorm` and `BC3Unorm` differ mostly in how they support alpha. BC1 supports only 1-bit alpha. BC2 supports each pixel in the block having a unique 4-bit alpha value. BC3 compresses the alpha values.

See Direct2D Block Compression documentation and Direct3D Block Compression documentation for more information about how block compression works.

Restrictions

- All block compressed textures must have a width and height that is a multiple of 4. This is because block compression works on blocks of 4x4 pixels.
- Any operation on a sub-rectangle of a block compressed texture (using `GetPixelBytes()`, `SetPixelBytes(Byte[])`, `CopyPixelsFromBitmap(CanvasBitmap, Int32, Int32)`) require that the sub-rectangle is 4-pixel aligned.
- Win2D requires premultiplied alpha when using block compressed formats.

Authoring DDS files

Block compressed images can be saved in DDS files. Although these can be generated by plugins to applications such as Photoshop or Paint.NET, care must be taken to ensure that the resulting file is saved with premultiplied alpha. Win2D will load any DDS file containing a `BC1Unorm`, `BC2Unorm` or `BC3Unorm` image and assume that it is authored with premultiplied alpha.

If you are authoring a C++ project then you can use the Image Content Pipeline to convert the image, as described on MSDN.

Alternatively, you can use `texconv.exe` from https://github.com/Microsoft/DirectXTex. `texconv.exe` can be built using the DirectXTex_Desktop_2015.sln solution. The following command converts "smoke.png" to a `BC3Unorm` with premultiplied alpha:

Windows Command Prompt

```
texconv -pmalpha -m 1 -f BC3_UNORM smoke.png
```

Effect precision and clamping

Article • 05/29/2023

Care must be taken while rendering effects using Win2D to achieve the desired level of quality and predictability with respect to numerical precision.

You need to understand these details if:

1. Your effect graph relies on high numerical precision or colors outside of the [0, 1] range, and you want to make sure these will always be available.
2. Or your effect graph relies on the rendering implementation to clamp intermediate colors to the [0, 1] range, so you must ensure this clamping always occurs.

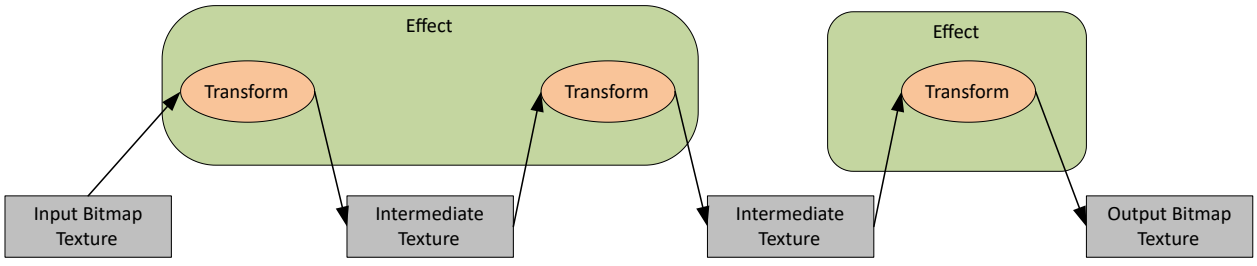
Win2D often divides an effect graph into sections, and renders each section in a separate step. The output of some steps may be stored in intermediate Direct3D textures which by default have limited numerical range and precision. Win2D makes no guarantees about if or where these intermediate textures are used. This behavior may vary according to GPU capabilities as well as between Windows versions.

In Windows 10, Win2D uses fewer intermediate textures due to its use of shader linking. Win2D may therefore produce different results with default settings than in prior Windows releases. This primarily affects scenarios where shader linking is possible in an effect graph, and that graph also contains effects that produce extended-range output colors.

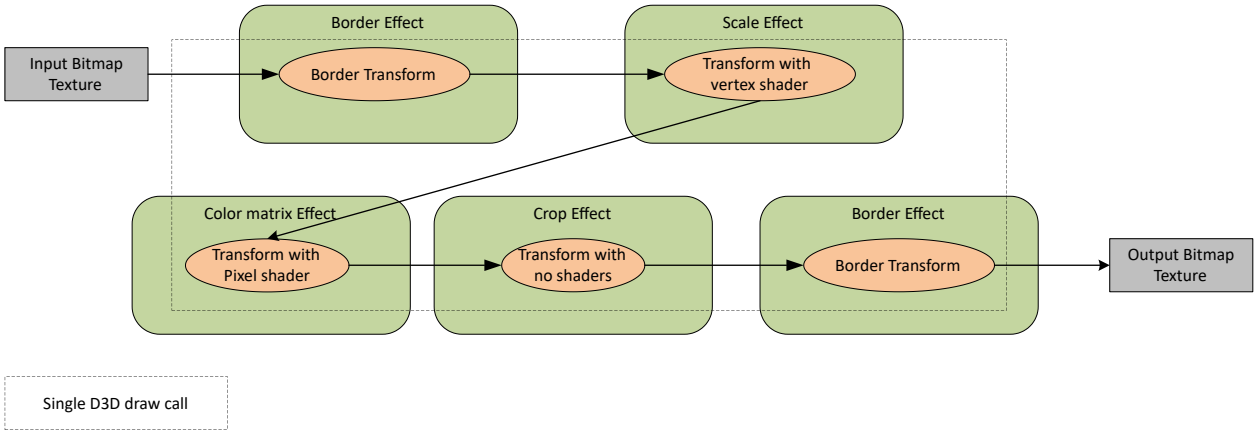
Overview of effect rendering and intermediates

To render an effect graph, Win2D first finds the underlying graph of "transforms", where a transform is a graph node used within an effect to apply a specific processing operation. Each image effect may internally be implemented using one or several transforms. There are different types of transforms, including those which provide Direct3D shaders for Direct2D to use.

For example, Win2D may render an effect graph as follows:

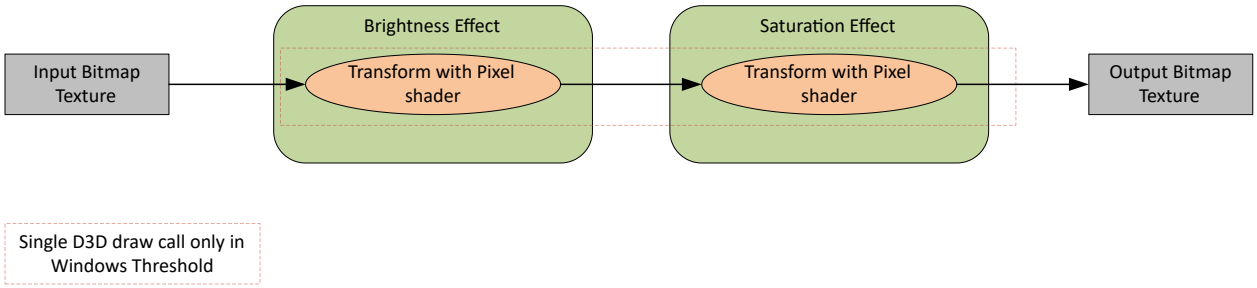


Win2D looks for opportunities to avoid using intermediate textures. This logic is opaque to applications, and may reduce the number of intermediate textures used. For example, the following graph can be rendered by Win2D using one Direct3D draw call and no intermediate textures:



Prior to Windows 10, Win2D would always use intermediate textures if multiple pixel shaders were used within the same effect graph. Most built-in effects that simply adjust color values (for example, Brightness or Saturation) do so using pixel shaders.

In Windows 10, Win2D may now avoid using intermediate textures in such cases. It does this by internally linking adjacent pixel shaders. For example:



Note that not all adjacent pixel shaders in a graph may be linked together, and therefore only certain graphs will produce different output on Windows 10. For full details see Effect Shader Linking. The primary restrictions are:

- An effect will not be linked with effects consuming its output, if the first effect is connected as an input to multiple effects.
- An effect will not be linked with an effect set as its input, if the first effect samples its input at a different logical position than its output. For example, a `ColorMatrix` effect might be linked with its input, but a `Convolution` effect will not be.

Built-in effect behavior

Many built-in effects may produce colors outside of the [0, 1] range in unpremultiplied space, even when their input colors are within that range. When this happens, such colors may be subject to numerical clamping. Note that it's important to consider the color range in unpremultiplied space, even though built-in effects typically produce colors in premultiplied space. This ensures that colors stay within range, even if other effects subsequently unpremultiply them.

Some of the effects which emit these out-of-range colors offer a `ClampOutput` property. These include:

- `ColorMatrix`
- `ArithmeticComposite`
- `Convolve`
- `Transfer` effects

Setting the `ClampOutput` property to true on these effects ensures a consistent result will be achieved regardless of factors such as shader linking. Note that clamping occurs in unpremultiplied space.

Other built-in effects may also produce output colors beyond the [0, 1] range in unpremultiplied space, even when their color pixels (and `Color properties if any) are within that range. These include:

- `Transform` and `Scale` effects (when the `InterpolationMode` property is `Cubic` or `HighQualityCubic`)
- Lighting effects
- `EdgeDetection` (when the `OverlayEdges` property is `true`)
- `Exposure`
- `Composite` (when the `Mode` property is `Add`)
- `Saturation`
- `Sepia`
- `TemperatureAndTint`

Forcing numerical clamping within an effect graph

While using effects listed above which do not have a `ClampOutput` property, applications should consider forcing numerical clamping. This can be done by inserting an additional effect into the graph that clamps its pixels. A `ColorMatrix` effect may be used, with its `ClampOutput` property set to `true`.

A second option to achieve consistent results is to request that Win2D use intermediate textures which have greater precision. This is described below.

Controlling precision of intermediate textures

Win2D provides a few ways to control the precision of a graph.

Before using high precision formats in Win2D, applications must ensure they are supported sufficiently by the GPU. To check this, use IsBufferPrecisionSupported(CanvasBufferPrecision).

Applications may create a Direct3D device using WARP (software emulation) to ensure that all buffer precisions are supported. This is recommended in scenarios such as applying effects to a photo while saving to disk. Even if Win2D supports high precision buffer formats on the GPU, using WARP is recommended in this scenario on feature level 9.X GPUs, due to limited precision of shader arithmetic and sampling on some low-power mobile GPUs. To use software rendering, specify `ForceSoftwareRenderer = true` on your Win2D XAML controls or when creating your `CanvasDevice`.

In each case below, the requested precision is actually the minimum precision Win2D will use. Higher precision may be used if intermediates are not required. Win2D may also share intermediate textures for different parts of the same graph or different graphs entirely. In this case Win2D uses the maximum precision requested for all involved operations.

Precision selection from the drawing session

The simplest way to control the precision of Win2D's intermediate textures is to use the EffectBufferPrecision property. This controls the precision of all intermediate textures, as long as a precision is not also set manually on effects directly.

C#

```
if
(canvasDevice.IsBufferPrecisionSupported(CanvasBufferPrecision.Precision32Fl
oat))
{
    drawingSession.EffectBufferPrecision =
CanvasBufferPrecision.Precision32Float;
}
```

Precision selection from inputs and render targets

Applications may also rely on the precision of the inputs to an effect graph to control the precision of intermediate textures.

The precisions of inputs to effects are propagated through the graph to select the precision of downstream intermediates. Where different branches in the effect graph meet, the greatest precision of any input is used.

The precision selected based on a Win2D bitmap is determined from its pixel format.

It is possible that Win2D cannot assign an effect a precision based on its inputs. This happens when an effect has no inputs, or when a command list is used, which has no specific precision. In this case, the precision of intermediate textures is determined from the current render target.

Precision selection directly on effects

The minimum precision for intermediate textures may also be set at explicit locations within an effect graph. This is only recommended for advanced applications.

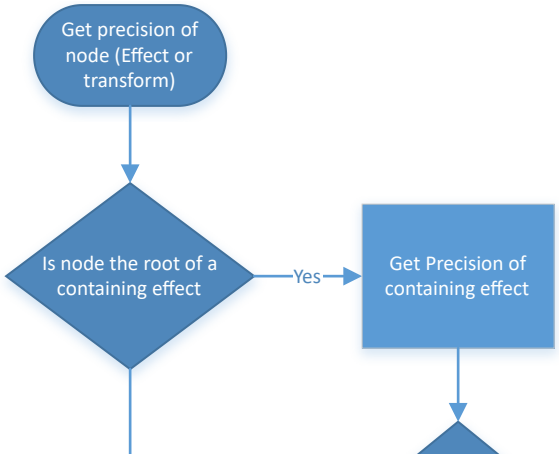
The minimum precision may be set using a property on an effect as follows:

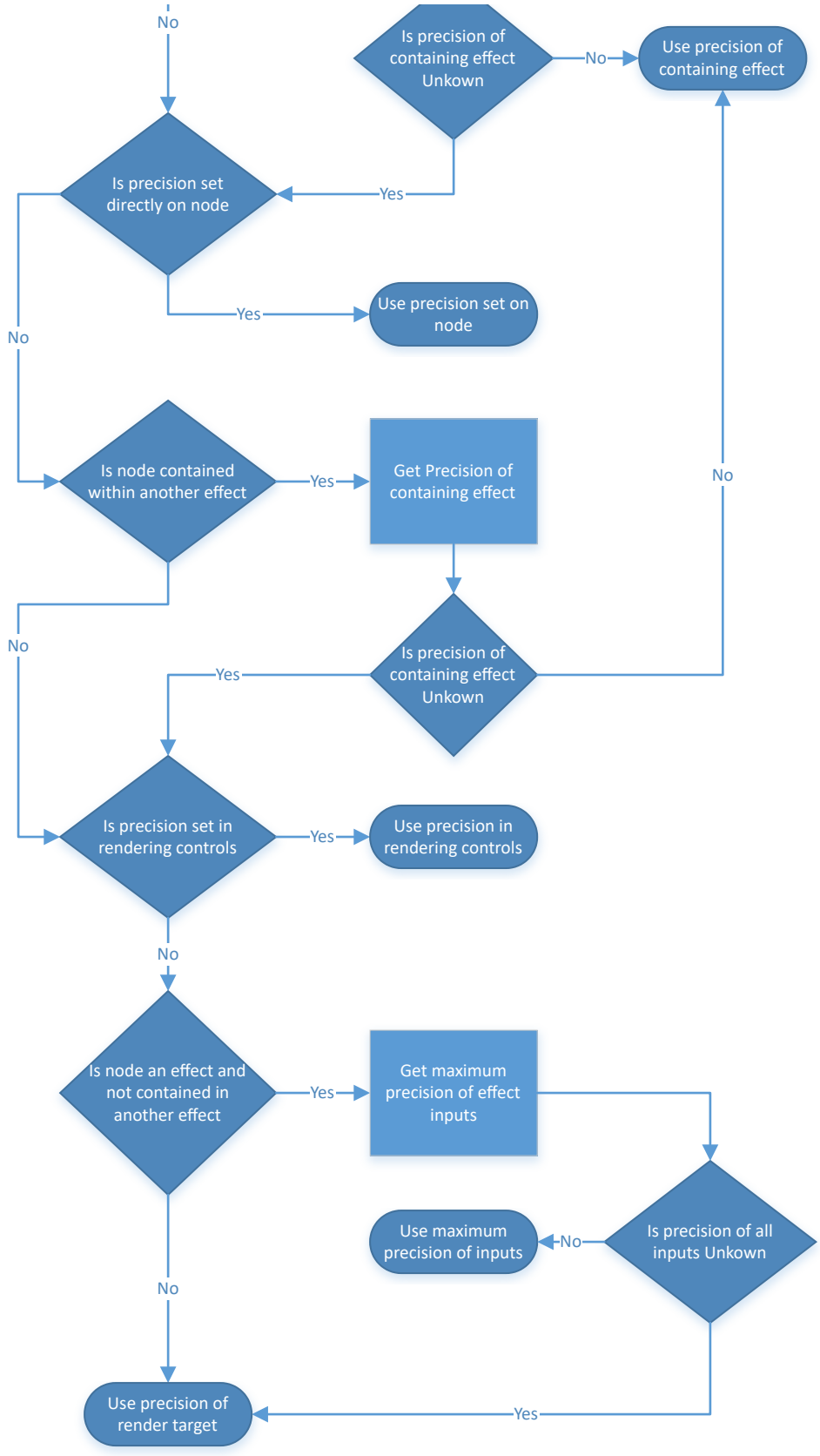
C#

```
if
(canvasDevice.IsBufferPrecisionSupported(CanvasBufferPrecision.Precision32Fl
oat))
{
    blurEffect.BufferPrecision = CanvasBufferPrecision.Precision32Float;
}
```

Note that the precision set on an effect will also apply to effects downstream in the same effect graph, unless a different precision is set on those downstream effects.

Below is the full recursive logic used to determine the minimum precision for an intermediate buffer storing the output of a given transform node:




No
Is precision of containing effect Unkown
No
Use precision of containing effect
Is precision set directly on node
Yes
Yes
Use precision set on node
No
Is node contained within another effect
Yes
Get Precision of containing effect
No
No
Is precision of containing effect Unkown
Yes
Is precision set in rendering controls
Yes
Use precision in rendering controls
No
Is node an effect and not contained in another effect
Yes
Get maximum precision of effect inputs
Use maximum precision of inputs
No
Is precision of all inputs Unkown
No
Use precision of render target
Yes

Handling device lost

Article • 05/29/2023

"Device lost" refers to a situation where the GPU graphics device becomes unusable for further rendering. This can occur due to GPU hardware malfunction, driver bugs, driver software updates, or switching the app from one GPU to another. A lost device can no longer be used, and any attempt to do so from Win2D will throw an exception. To recover from this situation, the app must create a new device and then recreate all its graphics resources.

Not all apps bother trying to recover from device lost. This is a (hopefully!) rare situation, so some developers will just let their apps crash if it occurs. For those who prefer to handle device lost in a robust way, this article explains how.

Device lost when using XAML controls

The Win2D controls (CanvasControl, CanvasVirtualControl and CanvasAnimatedControl) attempt to automatically handle device lost on behalf of the app.

When device lost is detected, these controls will recreate their `CanvasDevice` and then raise the `CreateResources` event passing a CanvasCreateResourcesReason of NewDevice. Apps should respond to this event by recreating all their graphics resources using the new device, and updating any data structures that may contain references to the old, no-longer-valid resources.

The controls can automatically catch and handle device lost exceptions that are thrown by their `CreateResources`, `Update`, `Draw`, or `RegionsInvalidated` event handlers. If you call Win2D drawing APIs from other places (for instance in a pointer or keyboard input event handler) then see the next section.

Manually handling device lost

If you are using Win2D without the built-in controls or from outside the `CreateResources`, `Update` or `Draw` event handlers, then it is your responsibility to catch and report lost device exceptions. This is done with the IsDeviceLost(Int32) and RaiseDeviceLost() methods:

```
C#
```

```
try
{
    DrawStuff();
}
catch (Exception e) where canvasDevice.IsDeviceLost(e.ErrorCode)
{
    canvasDevice.RaiseDeviceLost();
}
```

Calling `RaiseDeviceLost` tells any controls that are sharing this device to initiate their lost device recovery path. If you got your device from somewhere other than a control, use the DeviceLost event to be informed when it is lost.

How to test device lost handling

The easiest way to test that your app will do the right thing in response to device lost is to disable your hardware GPU while the app is running:

- Open Device Manager (Explorer window -> right-click 'This PC' -> 'Properties' -> 'Device Manager')
- Expand the 'Display adapters' node
- Right-click the entry for your GPU and click 'Disable' This will bypass your hardware GPU, at which point Windows switches over to a WARP software rendering device, and all active apps experience a device lost.

Don't forget to reenable your GPU after performing this test! If you want to test device lost more than once, it's best to restart your app each time (always launch it with the GPU enabled). Repeatedly toggling the GPU on and off while an app is running doesn't reliably cause lost devices as it may just continue using the WARP software device.

While testing lost device handling, you probably also want to make sure your app deals correctly with dynamic display DPI changes.

Using Win2D without built-in controls

Article • 05/29/2023

CanvasControl, CanvasVirtualControl and CanvasAnimatedControl are XAML controls - they extend `UserControl` and can exist alongside other controls in an app's XAML tree. They are good choice for many WinRT apps that use XAML and produce graphical content using Win2D. While these controls are versatile, they do impose policies pertaining to layout, resource re-creation, and device lost. Apps may want to implement their own XAML controls, or not use XAML at all.

Win2D is built to support this. This document describes how to use Win2D to draw graphics without use of `CanvasControl`, `CanvasVirtualControl` or `CanvasAnimatedControl`.

Layering

The Win2D XAML controls are built of top of a low level Win2D type. Each control contains an instance of a lower-level type:

Expand table

Control	Low-level type
`CanvasControl`	CanvasImageSource
`CanvasVirtualControl`	CanvasVirtualImageSource
`CanvasAnimatedControl`	CanvasSwapChainPanel and CanvasSwapChain

The controls consume only the public interfaces of these lower-level types. This implementation detail lends some confidence that apps can implement their own XAML controls which are equivalently as powerful as the built-in Win2D controls.

Choosing a lower-level type

An application is free to bypass the provided Win2D XAML controls and use the lower-level types directly. This section provides some guidance on choosing the appropriate lower-level type.

If there is existing C++ code that works with `IDXGISwapChain` then C++ interop can be used to wrap this with a `CanvasSwapChain`. `CanvasSwapChainPanel` can be used to display

a `CanvasSwapChain` in a XAML app.

If the app needs to add graphics to a XAML element which expects an ImageSource, it should use `CanvasImageSource` or `CanvasVirtualImageSource`.

Aside from that, it's worth considering:

- Swap chains are suited for content which will animate very frequently and the animation should be smooth. Direct3D's swap chains are designed for this purpose. The content of a swap chain can be re-drawn with a low latency, as it is not tied to the XAML framework refresh timer.
- `CanvasSwapChain` has heavier resource costs than `CanvasImageSource`. It is generally not desirable to have more than one or two swap chains onscreen at a time. For example, if an application has a page full of widgets, where each widget is a standalone graphical element, it is more appropriate to make each widget use a `CanvasImageSource` resource than a `CanvasSwapChain`.
- `CanvasImageSource` can be manipulated by other XAML UI elements such as transforms or opacity changes, while `CanvasSwapChain` cannot.
- `CanvasVirtualImageSource` can be used to display images that are much larger than the screen (for example in a `ScrollViewer`).

CanvasSwapChain and CoreWindow

CanvasSwapChain wraps a Direct3D swap chain. `CanvasSwapChain` is not a XAML type, but the fact that it has a swap chain means it has a built-in mechanism for being displayed. That said, CoreWindow apps may use `CanvasSwapChain` for displaying graphical content.

To create a `CanvasSwapChain` for use with a `CoreWindow`, in C#:

```
C#

float currentDpi = DisplayInformation.GetForCurrentView().LogicalDpi;

SwapChain = CanvasSwapChain.CreateForCoreWindow(device, window, currentDpi);
```

To draw content using a `CanvasSwapChain`, call its CreateDrawingSession(Color) method:

```
C#

using (CanvasDrawingSession ds =
swapChain.CreateDrawingSession(Colors.Black))
```

```
{
    ds.FillRectangle(100, 200, 3, 5);
}

swapChain.Present();
```

The size of the swap chain should match the size of the `CoreWindow`. If the size of the window changes, call ResizeBuffers(Size) on the swap chain with the new size. For more information, see the `CoreWindow` Win2D sample.

> **Note**
>
> `CoreWindow` is only supported on UWP. For WinAppSDK apps, use either composition or HWND swapchains.

CanvasSwapChainPanel

CanvasSwapChainPanel is a XAML type, and a relatively thin wrapper around `CanvasSwapChain`. It is suitable for XAML apps that require swap chain rendering, but do not want to use the policies that exist in `CanvasAnimatedControl`. To create a swap chain in XAML, use the namespace:

XAML

```
xmlns:canvas="using:Microsoft.Graphics.Canvas.UI.Xaml"
```

And declare:

XAML

```
<canvas:CanvasSwapChainPanel x:Name="canvasSwapChainPanel"/>
```

A `CanvasSwapChainPanel` does not have a swap chain automatically assigned to it. As an example, to assign one:

C#

```
CanvasDevice device = CanvasDevice.GetSharedDevice();
CanvasSwapChain swapChain = new CanvasSwapChain(device, width, height, 96);

canvasSwapChainPanel.SwapChain = swapChain;
```

To draw to the swap chain panel, in C#, use its `SwapChain` property:

```
C#

using (CanvasDrawingSession ds =
canvasSwapChainPanel.SwapChain.CreateDrawingSession(Colors.Black))
{
    ds.FillRectangle(200, 300, 5, 6, Colors.Blue);
}

canvasSwapChainPanel.SwapChain.Present();
```

The application decides on the frequency of redrawing. In the same manner as using `CanvasSwapChain` directly, it is up to the app to resize the swap chain when the control is resized. For example, in C#:

```
C#

canvasSwapChainPanel.SizeChanged += canvasSwapChainPanel_SizeChanged;

void canvasSwapChainPanel_SizeChanged(object sender,
Windows.UI.Xaml.SizeChangedEventArgs e)
{
    canvasSwapChainPanel.SwapChain.ResizeBuffers(e.NewSize);
}
```

Image Sources

CanvasImageSource and CanvasVirtualImageSource provide a way of integrating Win2D graphical content with XAML. They are suitable for content which does not require swap chain rendering.

CanvasImageSource

CanvasImageSource extends XAML's SurfaceImageSource. Apps can create an instance of `CanvasImageSource`, and reference it from a XAML type that consumes an `ImageSource`, such as an Image or ImageBrush. For example, in XAML markup:

```
XAML

<Image x:Name="image"/>
```

Then, in C#:

C#

```
CanvasDevice device = CanvasDevice.GetSharedDevice();
CanvasImageSource imageSource = new CanvasImageSource(device, width,
height);

image.Source = imageSource;
```

Drawing sessions are created directly on the image source object. For example:

C#

```
using (CanvasDrawingSession ds =
imageSource.CreateDrawingSession(Colors.Black))
{
    ds.FillRectangle(200, 300, 5, 6, Colors.Blue);
}
```

Note that a clear color must be passed to `CreateDrawingSession`. Whenever a drawing session is created on a `CanvasImageSource`, the `CanvasImageSource` is cleared.

For an example demonstrating how to use `CanvasImageSource`, see the `ImageSourceUpdateRegion` Win2D ExampleGallery page.

CanvasVirtualImageSource

CanvasVirtualImageSource wraps XAML's VirtualSurfaceImageSource.

The wrapped `VirtualSurfaceImageSource` can be obtained by the Source property. Apart from this difference it can be used in much the same way as `CanvasImageSource`:

XAML

```
<Image x:Name="image"/>
```

C#

```
var virtualImageSource = new CanvasVirtualImageSource(device, width,
height);

image.Source = virtualImageSource.Source;
```

CanvasRenderTarget

`CanvasRenderTarget` represents a drawable bitmap, and does not have any built-in association with XAML. It is suitable for XAML or non-XAML apps which need to use an intermediate bitmap- for example, for saving image data to a file, reading back pixel data, or to be used as an intermediate for another operation.

`CanvasRenderTarget` doesn't have an automatic mechanism to cause it to be displayed. To display the content of a `CanvasRenderTarget` in your app, draw it in the drawing session created from a displayed control, image source, or swap chain.

For more information about using `CanvasRenderTarget`, see Offscreen drawing.

Loading resources outside of CreateResources

Article • 05/29/2023

This document discusses how apps using Win2D's XAML controls, CanvasControl, CanvasVirtualControl and CanvasAnimatedControl, can load resources from outside the CreateResources handler.

Resource loading and CanvasControl / CanvasVirtualControl

Normally, apps are expected to use the `CreateResources` handler for creating controls' resources, so that device dependent resources are re-recreated as necessary if the device is lost. This includes resources that are loaded asynchronously. For asynchronous resource loading, controls are encouraged to use TrackAsyncAction(IAsyncAction) with `CreateResources` to ensure correct behavior.

All of this works well for cases where all resources are loaded at startup only.

But, what about apps which need to load some resources at startup, and some other resources later? For example, consider a game with different levels, and the levels need different graphical assets. Win2D doesn't have something built-in with `CreateResources` to enable this- an app cannot manually tell a control, "Re-issue `CreateResources` now, so that I can load different assets from before". However, the building blocks are there to make this work, and allow very good flexibility for how and when the resources are loaded, and be robust with respect to lost device.

Really, what an app wants to do in this case, is have a custom `LoadResourcesForLevelAsync` method, a "custom" `CreateResources`-launched task, like this:

C#

```
async Task LoadResourcesForLevelAsync(CanvasControl resourceCreator, int
level)
{
    levelBackground = await CanvasBitmap.LoadAsync(resourceCreator, ...);
    levelThingie = await CanvasBitmap.LoadAsync(resourceCreator, ...);
    // etc.
}
```

The app needs to load some of its resources after `CreateResources` is completed. In particular, the app will issue the level load after `CreateResources` has completed - e.g., from its `Draw` handler. In the code below, the app's `Draw` handler controls the progress of its level-loading Task.

To make `CreateResources` work in this situation, and be robust against lost devices, an app needs to do four things:

1. Track when `LoadResourcesForLevelAsync` is in progress.
2. Allow Win2D to handle any exceptions (in particular, device lost) that the app doesn't know how to handle.
3. If Win2D raises the `CreateResources` event to recover from a lost device while `LoadResourcesForLevelAsync` is in progress, your `CreateResources` handler should cancel that task.
4. If Win2D raises `CreateResources` to recover from a lost device after you have finished loading data using `LoadResourcesForLevelAsync`, your `CreateResources` handler must reload that custom data as well as its usual global resources.

Using a `CreateResources` handler called `CanvasControl_CreateResources`, and the `LoadResourcesForLevelAsync` method shown above, here is a complete implementation that handles all four requirements:

C#

```
int? currentLevel, wantedLevel;

// This implements requirement #1.
Task levelLoadTask;


public void LoadNewLevel(int newLevel)
{
    Debug.Assert(levelLoadTask == null);
    wantedLevel = newLevel;
    levelLoadTask = LoadResourcesForLevelAsync(canvasControl, newLevel);
}

void CanvasControl_CreateResources(CanvasControl sender,
                                   CanvasCreateResourcesEventArgs args)
{
    // Synchronous resource creation, for globally-required resources goes
here:
    x = new CanvasRenderTarget(sender, ...);
    y = new CanvasRadialGradientBrush(sender, ...);
    // etc.

    args.TrackAsyncAction(CreateResourcesAsync(sender).AsAsyncAction());
```

```
 }

 async Task CreateResourcesAsync(CanvasControl sender)
 {
     // If there is a previous load in progress, stop it, and
     // swallow any stale errors. This implements requirement #3.
     if (levelLoadTask != null)
     {
         levelLoadTask.AsAsyncAction().Cancel();
         try { await levelLoadTask; } catch { }
         levelLoadTask = null;
     }

     // Unload resources used by the previous level here.

     // Asynchronous resource loading, for globally-required resources goes
here:
     baz = await CanvasBitmap.LoadAsync(sender, ...);
     qux = await CanvasBitmap.LoadAsync(sender, ...);
     // etc.

     // If we are already in a level, reload its per-level resources.
     // This implements requirement #4.
     if (wantedLevel.HasValue)
     {
        LoadNewLevel(wantedLevel.Value);
     }
 }

// Because of how this is designed to throw an exception, this must only
// ever be called from a Win2D event handler.
bool IsLoadInProgress()
{
    // No loading task?
    if (levelLoadTask == null)
        return false;

    // Loading task is still running?
    if (!levelLoadTask.IsCompleted)
        return true;

     // Query the load task results and re-throw any exceptions
     // so Win2D can see them. This implements requirement #2.
     try
     {
         levelLoadTask.Wait();
     }
     catch (AggregateException aggregateException)
     {
         // .NET async tasks wrap all errors in an AggregateException.
         // We unpack this so Win2D can directly see any lost device errors.
         aggregateException.Handle(exception => { throw exception; });
     }
     finally
     {
```

```
            levelLoadTask = null;
        }

        currentLevel = wantedLevel;
        return false;
    }


    void CanvasControl_Draw(CanvasControl sender, CanvasDrawEventArgs args)
    {
        if (IsLoadInProgress())
        {
            DrawLoadingScreen();
        }
        else
        {
            DrawCurrentLevel(currentLevel);
        }
    }
```

Resource loading and CanvasAnimatedControl

Much of the above information for `CanvasControl` generalizes to `CanvasAnimatedControl`.

However, `CanvasAnimatedControl` has a game loop thread, which should be taken into consideration while deciding how an app should load its resources.

In particular:

- Apps should avoid blocking or waiting from within the `Draw` or `Update` handlers.
- Apps may need to synchronize data which is accessed between multiple threads.

We can adjust the above implementation to be compatible with `CanvasAnimatedControl`. In particular, we should ensure that the code checking "is the level loaded" will poll instead of wait, and no `CanvasAnimatedControl` event handlers should be marked as async.

C#

```
async Task LoadResourcesForLevelAsync(ICanvasAnimatedControl
resourceCreator, int level)
{
    return
GameLoopSynchronizationContext.RunOnGameLoopThreadAsync(resourceCreator,
async() =>
    {
        levelBackground = await CanvasBitmap.LoadAsync(resourceCreator,
```

```
...);
        levelThingie = await CanvasBitmap.LoadAsync(resourceCreator, ...);
        // etc.
    }
}


// Shared state between all threads, and a lock to control access to it.
bool needToLoad;
int? currentLevel, wantedLevel;
Task levelLoadTask; // This implements requirement #1.

Object lockable = new Object();

void LoadNewLevel(int level)
{
    lock(lockable)
    {
        wantedLevel = level;
        needToLoad = true;
    }
}

void canvasAnimatedControl_CreateResources(ICanvasAnimatedControl sender,
                                    CanvasCreateResourcesEventArgs args)
{
     // Synchronous resource creation, for globally-required resources goes
here:
     x = new CanvasRenderTarget(sender, ...);
     y = new CanvasRadialGradientBrush(sender, ...);
     // etc.

     args.TrackAsyncAction(CreateResourcesAsync(sender).AsAsyncAction());
}

async Task CreateResourcesAsync(CanvasAnimatedControl sender)
{
    // If there is a previous load in progress, stop it, and
    // swallow any stale errors. This implements requirement #3.
    lock(lockable)
    {
        if (levelLoadTask != null)
        {
            levelLoadTask.AsAsyncAction().Cancel();
            try { await levelLoadTask; } catch { }
            levelLoadTask = null;
        }
    }

    // Unload resources used by the previous level here.

    // Asynchronous resource loading, for globally-required resources goes
here:
    baz = await CanvasBitmap.LoadAsync(sender, ...);
    qux = await CanvasBitmap.LoadAsync(sender, ...);
```

```
    // etc.

    // If we are already in a level, reload its per-level resources.
    // This implements requirement #4.
    int? levelThatNeedsReloading;
    lock(lockable)
    {
        levelThatNeedsReloading = wantedLevel;
    }
    if (levelThatNeedsReloading.HasValue)
    {
        LoadNewLevel(wantedLevel.Value);
    }
}

void canvasAnimatedControl_Update(ICanvasAnimatedControl sender,
CanvasAnimatedUpdateEventArgs args)
{
    lock(lockable)
    {
        // Check if there is already an outstanding level-loading Task.
        // If so, don't try to spin up a new one.
        bool beginLoad = levelLoadTask == null && needToLoad;
        needToLoad = false;

        if (beginLoad)
        {
            levelLoadTask = LoadResourcesForLevelAsync(sender, wantedLevel);
        }

        // Indicates the loading task was run and just finished.
        if(levelLoadTask != null && levelLoadTask.IsCompleted)
        {
            AggregateException levelLoadException = levelLoadTask.Exception;
            levelLoadTask = null;

            // Query the load task results and re-throw any exceptions
            // so Win2D can see them. This implements requirement #2.
            if(levelLoadException != null)
            {
                // .NET async tasks wrap all errors in an
AggregateException.
                // We unpack this so Win2D can directly see any lost device
errors.
                levelLoadException.Handle(exception => { throw exception;
});
            }

            currentLevel = wantedLevel;
        }
    }
}

bool IsLoadInProgress()
{
```

```
    lock(lockable)
    {
        return levelLoadTask != null;
    }
}


void canvasAnimatedControl_Draw(ICanvasAnimatedControl sender,
CanvasAnimatedDrawEventArgs args)
{
    if (IsLoadInProgress())
    {
        DrawLoadingScreen(args.DrawingSession);
    }
    else
    {
        DrawCurrentLevel(args.DrawingSession, currentLevel);
    }
}
```

It's worth pointing out that there are special implications of using `await` from within application code which runs on the game loop thread. For example, when a task is run using `RunOnGameLoopThreadAsync` and contains an `await`, the first await may result in `RunOnGameLoopThreadAsync` finishing. The remainder of the task is scheduled according a continuation handler. And by default, this continuation handler is chosen from the application's thread pool, according to the default .NET `SynchronizationContext`. The remainder of the task might not run on the game loop thread at all.

To remedy this, and schedule work using `RunOnGameLoopThreadAsync` which contains `await` where the work must all be run on the game loop thread, see the `GameLoopSynchronizationContext` sample.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Avoiding memory leaks

Article • 05/29/2023

When using Win2D controls in managed XAML applications, care must be taken to avoid reference count cycles that could prevent these controls ever being reclaimed by the garbage collector.

You have a problem if...

- You are using Win2D from a .NET language such as C# (not native C++)
- You use one of the Win2D XAML controls:
  - CanvasControl
  - CanvasVirtualControl
  - CanvasAnimatedControl
  - CanvasSwapChainPanel
- You subscribe to events of the Win2D control (eg. `Draw`, `CreateResources`, `SizeChanged`...)
- Your app moves back and forth between more than one XAML page

If all these conditions are met, a reference count cycle will keep the Win2D control from ever being garbage collected. New Win2D resources are allocated each time the app moves to a different page, but the old ones are never freed so memory is leaked. To avoid this, you must add code to explicitly break the cycle.

How to fix it

To break the reference count cycle and let your page be garbage collected:

- Hook the `Unloaded` event of the XAML page which contains the Win2D control
- In the `Unloaded` handler, call `RemoveFromVisualTree` on the Win2D control
- In the `Unloaded` handler, release (by setting to `null`) any explicit references to the Win2D control

Example code:

C#

```
void page_Unloaded(object sender, RoutedEventArgs e)
{
    this.canvas.RemoveFromVisualTree();
```

```
    this.canvas = null;
}
```

For working examples, see any of the Example Gallery demo pages.

How to test for cycle leaks

To test whether your application is correctly breaking refcount cycles, add a finalizer method to any XAML pages which contain Win2D controls:

C#

```
~MyPage()
{
    System.Diagnostics.Debug.WriteLine("~" + GetType().Name);
}
```

In your `App` constructor, set up a timer that will make sure garbage collection occurs at regular intervals:

C#

```
var gcTimer = new DispatcherTimer();
gcTimer.Tick += (sender, e) => { GC.Collect(); };
gcTimer.Interval = TimeSpan.FromSeconds(1);
gcTimer.Start();
```

Navigate to the page, then away from it to some other page. If all cycles have been broken, you will see `Debug.WriteLine` output in the Visual Studio output pane within a second or two.

Note that calling `GC.Collect` is disruptive and hurts performance, so you should remove this test code as soon as you finish testing for leaks!

The gory details

A cycle occurs when an object A has a reference to B, at the same time as B also has a reference to A. Or when A references B, and B references C, while C references A, etc.

When subscribing to events of a XAML control, this sort of cycle is pretty much inevitable:

- XAML page holds references to all the controls contained within it

- Controls hold references to the handler delegates that have been subscribed to their events
- Each delegate holds a reference to its target instance
- Event handlers are typically instance methods of the XAML page class, so their target instance references point back to the XAML page, creating a cycle

If all the objects involved are implemented in .NET, such cycles are not a problem because .NET is garbage collected, and the garbage collection algorithm is able to identify and reclaim groups of objects even if they are linked in a cycle.

Unlike .NET, C++ manages memory by reference counting, which is unable to detect and reclaim cycles of objects. In spite of this limitation, C++ apps using Win2D have no problem because C++ event handlers default to holding weak rather than strong references to their target instance. Therefore the page references the control, and the control references the event handler delegate, but this delegate does not reference back to the page so there is no cycle.

The problem comes when a C++ WinRT component such as Win2D is used by a .NET application:

- The XAML page is part of the application, so uses garbage collection
- The Win2D control is implemented in C++, so uses reference counting
- The event handler delegate is part of the application, so uses garbage collection and holds a strong reference to its target instance

A cycle is present, but the Win2D objects participating in this cycle are not using .NET garbage collection. This means the garbage collector is unable to see the entire chain, so it cannot detect or reclaim the objects. When this occurs, the application must help out by explicitly breaking the cycle. This can be done either by releasing all references from the page to the control (as recommended above) or by releasing all references from the control to event handler delegates that might point back to the page (using the page Unloaded event to unsubscribe all event handlers).

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Implementing custom effects

Article • 05/30/2023

Win2D provides several APIs to represent objects that can be drawn, which are divided into two categories: images and effects. Images, represented by the `ICanvasImage` interface, have no inputs and can be directly drawn on a given surface. For example, `CanvasBitmap`, `VirtualizedCanvasBitmap` and `CanvasRenderTarget` are examples of image types. Effects, on the other hand, are represented by the `ICanvasEffect` interface. They can have inputs as well as additional resources, and can apply arbitrary logic to produce their outputs (as an effect is also an image). Win2D includes effects wrapping most D2D effects, such as `GaussianBlurEffect`, `TintEffect` and `LuminanceToAlphaEffect`.

Images and effects can also be chained together, to create arbitrary graphs which can then be displayed in your application (also refer to the D2D docs on Direct2D effects). Together, they provide an extremely flexible system to author complex graphics in an efficient manner. However, there are cases where the built-in effects are not sufficient, and you might want to build your very own Win2D effect. To support this, Win2D includes a set of powerful interop APIs that allows defining custom images and effects that can seamlessly integrate with Win2D.

> **Tip**
>
> If you are using C# and want to implement a custom effect or effect graph, it is recommended to use **ComputeSharp** rather than trying to implement an effect from scratch. **See the paragraph below** for a detailed explanation of how to use this library to implement custom effects that integrate seamlessly with Win2D.

> **Platform APIs:** ICanvasImage, CanvasBitmap, VirtualizedCanvasBitmap, CanvasRenderTarget, CanvasEffect, GaussianBlurEffect, TintEffect, ICanvasLuminanceToAlphaEffectImage, IGraphicsEffectSource, ID2D21Image, ID2D1Factory1, ID2D1Effect

Implementing a custom `ICanvasImage`

The simplest scenario to support is creating a custom `ICanvasImage`. As we mentioned, this is the WinRT interface defined by Win2D which represents all kinds of images that Win2D can interop with. This interface only exposes two `GetBounds` methods, and extends `IGraphicsEffectSource`, which is a marker interface representing "some effect source".

As you can see, there are no "functional" APIs exposed by this interface to actually perform any drawing. In order to implement your own `ICanvasImage` object, you'll need to also implement the `ICanvasImageInterop` interface, which exposes all the necessary logic for Win2D to draw the image. This is a COM interface defined in the public `Microsoft.Graphics.Canvas.native.h` header, that ships with Win2D.

The interface is defined as follows:

C++

```
[uuid("E042D1F7-F9AD-4479-A713-67627EA31863")]
class ICanvasImageInterop : IUnknown
{
    HRESULT GetDevice(
        ICanvasDevice** device,
        WIN2D_GET_DEVICE_ASSOCIATION_TYPE* type);

    HRESULT GetD2DImage(
        ICanvasDevice* device,
        ID2D1DeviceContext* deviceContext,
        WIN2D_GET_D2D_IMAGE_FLAGS flags,
        float targetDpi,
        float* realizeDpi,
        ID2D1Image** ppImage);
}
```

And it also relies on these two enumeration types, from the same header:

C++

```
enum WIN2D_GET_DEVICE_ASSOCIATION_TYPE
{
    WIN2D_GET_DEVICE_ASSOCIATION_TYPE_UNSPECIFIED,
    WIN2D_GET_DEVICE_ASSOCIATION_TYPE_REALIZATION_DEVICE,
    WIN2D_GET_DEVICE_ASSOCIATION_TYPE_CREATION_DEVICE
}

enum WIN2D_GET_D2D_IMAGE_FLAGS
{
    WIN2D_GET_D2D_IMAGE_FLAGS_NONE,
    WIN2D_GET_D2D_IMAGE_FLAGS_READ_DPI_FROM_DEVICE_CONTEXT,
    WIN2D_GET_D2D_IMAGE_FLAGS_ALWAYS_INSERT_DPI_COMPENSATION,
    WIN2D_GET_D2D_IMAGE_FLAGS_NEVER_INSERT_DPI_COMPENSATION,
    WIN2D_GET_D2D_IMAGE_FLAGS_MINIMAL_REALIZATION,
    WIN2D_GET_D2D_IMAGE_FLAGS_ALLOW_NULL_EFFECT_INPUTS,
    WIN2D_GET_D2D_IMAGE_FLAGS_UNREALIZE_ON_FAILURE
}
```

The two `GetDevice` and `GetD2DImage` methods are all that's needed to implement custom images (or effects), as they provide Win2D with the extensibility points to initialize them on a given device and retrieve the underlying D2D image to draw. Implementing these methods correctly is critical to ensure things will work properly in all supported scenarios.

Let's go over them to see how each method works.

Implementing `GetDevice`

The `GetDevice` method is the simplest of the two. What it does is it retrieves the canvas device associated with the effect, so that Win2D can inspect it if necessary (for instance, to ensure it matches the device in use). The `type` parameter indicates the "association type" for the returned device.

There are two main possible cases:

- If the image is an effect, it should support being "realized" and "unrealized" on multiple devices. What this means is: a given effect is created in an uninitialized state, then it can be realized when a device is passed while drawing, and after that it can keep being used with that device, or it can be moved to a different device. In that case, the effect will reset its internal state and then realize itself again on the new device. This means that the associated canvas device can change over time, and it can also be `null`. Because of this, `type` should be set to `WIN2D_GET_DEVICE_ASSOCIATION_TYPE_REALIZATION_DEVICE`, and the returned device should be set to the current realization device, if one is available.
- Some images have a single "owning device" which is assigned at creation time and can never change. For instance, this would be the case for an image representing a texture, as that is allocated on a specific device and cannot be moved. When `GetDevice` is called, it should return the creation device and set `type` to `WIN2D_GET_DEVICE_ASSOCIATION_TYPE_CREATION_DEVICE`. Note that when this type is specified, the returned device should not be `null`.

> ⓘ **Note**
>
> Win2D can call `GetDevice` while recursively traversing an effect graph, meaning there might be multiple active calls to `GetD2DImage` in the stack. Because of this, `GetDevice` should not take a blocking lock on the current image, as that could potentially deadlock. Rather, it should use a re-entrant lock in a non-blocking manner, and return an error if it cannot be acquired. This ensures that the same

thread recursively calling it will successfully acquire it, whereas concurrent threads doing the same will fail gracefully.

Implementing `GetD2DImage`

`GetD2DImage` is where most of the work takes place. This method is responsible for retrieving the `ID2D1Image` object that Win2D can draw, optionally realizing the current effect if needed. This also includes recursively traversing and realizing the effect graph for all sources, if any, as well as initializing any state that the image might need (eg. constant buffers and other properties, resource textures, etc.).

The exact implementation of this method is highly dependent on the image type and it can vary a lot, but generally speaking for an arbitrary effect you can expect the method to perform the following steps:

- Check whether the call was recursive on the same instance, and fail if so. This is needed to detect cycles in an effect graph (eg. effect `A` has effect `B` as source, and effect `B` has effect `A` as source).
- Acquire a lock on the image instance to protect against concurrent access.
- Handle the target DPIs according to the input flags
- Validate whether the input device matches the one in use, if any. If it does not match and the current effect supports realization, unrealize the effect.
- Realize the effect on the input device. This can include registering the D2D effect on the `ID2D1Factory1` object retrieved from the input device or device context, if needed. Additionally, all necessary state should be set on the D2D effect instance being created.
- Recursively traverse any sources and bind them to the D2D effect.

With respect to the input flags, there are several possible cases that custom effects should properly handle, to ensure compatibility with all other Win2D effects. Excluding `WIN2D_GET_D2D_IMAGE_FLAGS_NONE`, the flags to handle are the following:

- `WIN2D_GET_D2D_IMAGE_FLAGS_READ_DPI_FROM_DEVICE_CONTEXT`: in this case, `device` is guaranteed to not be `null`. The effect should check whether the device context target is an `ID2D1CommandList`, and if so, add the `WIN2D_GET_D2D_IMAGE_FLAGS_ALWAYS_INSERT_DPI_COMPENSATION` flag. Otherwise, it should set `targetDpi` (which is also guaranteed to not be `null`) to the DPIs retrieved from the input context. Then, it should remove `WIN2D_GET_D2D_IMAGE_FLAGS_READ_DPI_FROM_DEVICE_CONTEXT` from the flags.

- `WIN2D_GET_D2D_IMAGE_FLAGS_ALWAYS_INSERT_DPI_COMPENSATION` and `WIN2D_GET_D2D_IMAGE_FLAGS_NEVER_INSERT_DPI_COMPENSATION`: used when setting effect sources (see notes below).
- `WIN2D_GET_D2D_IMAGE_FLAGS_MINIMAL_REALIZATION`: if set, skips recursively realizing the sources of the effect, and just returns the realized effect with no other changes.
- `WIN2D_GET_D2D_IMAGE_FLAGS_ALLOW_NULL_EFFECT_INPUTS`: if set, effect sources being realized are allowed to be `null`, if the user has not set them to an existing source yet.
- `WIN2D_GET_D2D_IMAGE_FLAGS_UNREALIZE_ON_FAILURE`: if set, and an effect source being set is not valid, the effect should unrealize before failing. That is, if the error occurred while resolving the effect sources after realizing the effect, the effect should unrealize itself before returning the error to the caller.

With respect to the DPI-related flags, these control how effect sources are set. To ensure compatibility with Win2D, effects should automatically add DPI compensation effects to their inputs when needed. They can control whether that is the case like so:

- If `WIN2D_GET_D2D_IMAGE_FLAGS_MINIMAL_REALIZATION` is set, a DPI compensation effect is needed whenever the `inputDpi` parameter is not `0`.
- Otherwise, DPI compensation is needed if `inputDpi` is not `0`, `WIN2D_GET_D2D_IMAGE_FLAGS_NEVER_INSERT_DPI_COMPENSATION` is not set, and either `WIN2D_GET_D2D_IMAGE_FLAGS_ALWAYS_INSERT_DPI_COMPENSATION` is set, or the input DPI and the target DPI values don't match.

This logic should be applied whenever a source is being realized and bound to an input of the current effect. Note that if a DPI compensation effect is added, that should be the input set to the underlying D2D image. But, if the user tries to retrieve the WinRT wrapper for that source, the effect should take care to detect whether a DPI effect was used, and return a wrapper for the original source object instead. That is, DPI compensation effects should be transparent to users of the effect.

After all initialization logic is done, the resulting `ID2D1Image` (just like with Win2D objects, a D2D effect is also an image) should be ready to be drawn by Win2D on the target context, which is not yet known by the callee at this time.

> ⓘ **Note**
>
> Correctly implementing this method (and `ICanvasImageInterop` in general) is **extremely** complicated, and it's only meant to be done by advanced users that absolutely need the extra flexibility. A solid understanding of D2D, Win2D, COM, WinRT and C++ is recommended before attempting to write an

`ICanvasImageInterop` implementation. If your custom Win2D effect also has to wrap a custom D2D effect, you'll need to implement your own `ID2D1Effect` object as well (refer to the D2D docs on custom effects for more info on this). These docs are not an exhaustive description of all necessary logic (for instance, they don't cover how effect sources should be marshalled and managed across the D2D/Win2D boundary), so it is recommended to also use the `CanvasEffect` implementation in Win2D's codebase as a reference point for a custom effect, and modify it as needed.

Implementing `GetBounds`

The last missing component to fully implement a custom `ICanvasImage` effect is to support the two `GetBounds` overloads. To make this easy, Win2D exposes a C export which can be used to leverage the existing logic for this from Win2D on any custom image. The export is as follows:

C++

```
HRESULT GetBoundsForICanvasImageInterop(
    ICanvasResourceCreator* resourceCreator,
    ICanvasImageInterop* image,
    Numerics::Matrix3x2 const* transform,
    Rect* rect);
```

Custom images can invoke this API and pass themselves as the `image` parameter, and then simply return the result to their callers. The `transform` parameter can be `null`, if no transform is available.

Optimizing device context accesses

The `deviceContext` parameter in `ICanvasImageInterop::GetD2DImage` can sometimes be `null`, if a context is not immediately available before the invocation. This is done on purpose, so that a context is only created lazily when it's actually needed. That is, if a context is available, Win2D will pass it to the `GetD2DImage` invocation, otherwise it will let callees retrieve one on their own if necessary.

Creating a device context is relatively expensive, so to make retrieving one faster Win2D exposes APIs to access its internal device context pool. This allows custom effects to rent and return device contexts associated with a given canvas device in an efficient manner.

The device context lease APIs are defined as follows:

C++

```
[uuid("A0928F38-F7D5-44DD-A5C9-E23D94734BBB")]
interface ID2D1DeviceContextLease : IUnknown
{
    HRESULT GetD2DDeviceContext(ID2D1DeviceContext** deviceContext);
}

[uuid("454A82A1-F024-40DB-BD5B-8F527FD58AD0")]
interface ID2D1DeviceContextPool : IUnknown
{
    HRESULT GetDeviceContextLease(ID2D1DeviceContextLease** lease);
}
```

The `ID2D1DeviceContextPool` interface is implemented by `CanvasDevice`, which is the Win2D type implementing the `ICanvasDevice` interface. To use the pool, use `QueryInterface` on the device interface to obtain an `ID2D1DeviceContextPool` reference, and then call `ID2D1DeviceContextPool::GetDeviceContextLease` to obtain an `ID2D1DeviceContextLease` object to access the device context. Once that's no longer needed, release the lease. Make sure to not touch the device context after the lease has been released, as it might be used concurrently by other threads.

Enabling WinRT wrappers lookup

As seen in the Win2D interop docs, the Win2D public header also exposes a `GetOrCreate` method (accessible from the `ICanvasFactoryNative` activation factory, or through the `GetOrCreate` C++/CX helpers defined in the same header). This allows retrieving a WinRT wrapper from a given native resource. For instance, it lets you retrieve or create a `CanvasDevice` instance from an `ID2D1Device1` object, a `CanvasBitmap` from an `ID2D1Bitmap`, etc.

This method also works for all built-in Win2D effects: retrieving the native resource for a given effect and then using that to retrieve the corresponding Win2D wrapper will correctly return the owning Win2D effect for it. In order for custom effects to also benefit from the same mapping system, Win2D exposes several APIs in the interop interface for the activation factory for `CanvasDevice`, which is the `ICanvasFactoryNative` type, as well as an additional effect factory interface, `ICanvasEffectFactoryNative`:

C++

```
[uuid("29BA1A1F-1CFE-44C3-984D-426D61B51427")]
class ICanvasEffectFactoryNative : IUnknown
{
    HRESULT CreateWrapper(
```

```
        ICanvasDevice* device,
        ID2D1Effect* resource,
        float dpi,
        IInspectable** wrapper);
};

[uuid("695C440D-04B3-4EDD-BFD9-63E51E9F7202")]
class ICanvasFactoryNative : IInspectable
{
    HRESULT GetOrCreate(
        ICanvasDevice* device,
        IUnknown* resource,
        float dpi,
        IInspectable** wrapper);

    HRESULT RegisterWrapper(IUnknown* resource, IInspectable* wrapper);

    HRESULT UnregisterWrapper(IUnknown* resource);

    HRESULT RegisterEffectFactory(
        REFIID effectId,
        ICanvasEffectFactoryNative* factory);

    HRESULT UnregisterEffectFactory(REFIID effectId);
};
```

There are several APIs to consider here, as they're needed to support all the various scenarios where Win2D effects can be used, as well as how developers could do interop with the D2D layer and then try to resolve wrappers for them. Let's go over each of these APIs.

The `RegisterWrapper` and `UnregisterWrapper` methods are meant to be invoked by custom effects to add themselves into the internal Win2D cache:

- `RegisterWrapper`: registers a native resource and its owning WinRT wrapper. The `wrapper` parameter is required to also implemement IWeakReferenceSource, so that it can be cached correctly without causing reference cycles which would lead to memory leaks. The method returns `S_OK` if the native resource could be added to the cache, `S_FALSE` if there was already a registered wrapper for `resource`, and an error code if an error occurs.
- `UnregisterWrapper`: unregisters a native resource and its wrapper. Returns `S_OK` if the resource could be removed, `S_FALSE` if `resource` was not already registered, and an erro code if another error occurrs.

Custom effects should call `RegisterWrapper` and `UnregisterWrapper` whenever they are realized and unrealized, ie. when a new native resource is created and associated with them. Custom effects that do not support realization (eg. those having a fixed

associated device) can call `RegisterWrapper` and `UnregisterWrapper` when they are created and destroyed. Custom effects should make sure to correctly unregister themselves from all possible code paths that would cause the wrapper to become invalid (eg. including when the object is finalized, in case it's implemented in a managed language).

The `RegisterEffectFactory` and `UnregisterEffectFactory` methods are also meant to be used by custom effects, so that they can also register a callback to create a new wrapper in case a developer tries to resolve one for an "orphaned" D2D resource:

- `RegisterEffectFactory`: register a callback that takes in input the same parameters that a developer passed to `GetOrCreate`, and creates a new inspectable wrapper for the input effect. The effect id is used as key, so that each custom effect can register a factory for it when it's first loaded. Of course, this should only be done once per effect type, and not every time the effect is realized. The `device`, `resource` and `wrapper` parameters are checked by Win2D before invoking any registered callback, so they are guaranteed to not be `null` when `CreateWrapper` is invoked. The `dpi` is considered optional, and can be ignored in case the effect type doesn't have a specific use for it. Note that when a new wrapper is created from a registered factory, that factory should also make sure that the new wrapper is registered in the cache (Win2D will not automatically add wrappers produced by external factories to the cache).
- `UnregisterEffectFactory`: removes a previously register callback. For instance, this could be used if an effect wrapper is implemented in a managed assembly which is being unloaded.

> ⓘ **Note**
>
> `ICanvasFactoryNative` is implemented by the activation factory for `CanvasDevice`, which you can retrieve by either manually calling RoGetActivationFactory, or using helper APIs from the language extensions you're using (eg. winrt::get_activation_factory in C++/WinRT). For more info, see WinRT type system for more information on how this works.

For a practical example of where this mapping comes into play, consider how built-in Win2D effects work. If they are not realized, all state (eg. properties, sources, etc.) is stored in an internal cache in each effect instance. When they are realized, all state is transferred to the native resource (eg. properties are set on the D2D effect, all sources are resolved and mapped to effect inputs, etc.), and as long as the effect is realized it will act as the authority on the state of the wrapper. That is, if the value of any property

is fetched from the wrapper, it will retrieve the updated value for it from the native D2D resource associated with it.

This ensures that if any changes are made directly to the D2D resource, those will be visible on the outer wrapper as well, and the two will never be "out of sync". When the effect is unrealized, all state is transferred back from the native resource to the wrapper state, before the resource is released. It will be kept and updated there until the next time the effect is realized. Now, consider this sequence of events:

- You have some Win2D effect (either built-in, or custom).
- You get the `ID2D1Image` from it (which is an `ID2D1Effect`).
- You create an instance of a custom effect.
- You also get the `ID2D1Image` from that.
- You manually set this image as input for the previous effect (via ID2D1Effect::SetInput).
- You then ask that first effect for the WinRT wrapper for that input.

Since the effect is realized (it was realized when the native resource was requested), it will use the native resource as the source of truth. As such, it will get the `ID2D1Image` corresponding to the requested source, and try to retrieve the WinRT wrapper for it. If the effect this input was retrieved from has correctly added its own pair of native resource and WinRT wrapper to Win2D's cache, the wrapper will be resolved and returned to callers. If not, that property access will fail, as Win2D can't resolve WinRT wrappers for effects it does not own, as it doesn't know how to instantiate them.

This is where `RegisterWrapper` and `UnregisterWrapper` help, as they allow custom effects to seamlessly participate in Win2D's wrapper resolution logic, so that the correct wrapper can always be retrieved for any effect source, regardless of whether it was set from WinRT APIs, or directly from the underlying D2D layer.

To explain how the effect factories also come into play, consider this scenario:

- A user creates an instance of a custom wrapper and realizes it
- They then gets a reference to the underlying D2D effect and keeps it.
- Then, the effect is realized on a different device. The effect will unrealize and re-realize, and in doing so it will create a new D2D effect. The previous D2D effect no longer as an associated inspectable wrapper at this point.
- The user then calls `GetOrCreate` on the first D2D effect.

Without a callback, Win2D would just fail to resolve a wrapper, as there's no registered wrapper for it. If a factory is registered instead, a new wrapper for that D2D effect can be created and returned, so the scenario just keeps working seamlessly for the user.

Implementing a custom `ICanvasEffect`

The Win2D `ICanvasEffect` interface extends `ICanvasImage`, so all the previous points apply to custom effects as well. The only difference is the fact that `ICanvasEffect` also implements additional methods specific to effects, such as invalidating a source rectangle, getting the required rectangles and so on.

To support this, Win2D exposes C exports that authors of custom effects can use, so that they won't have to reimplement all this extra logic from scratch. This works in the same way as the C export for `GetBounds`. Here are the available exports for effects:

C++

```
HRESULT InvalidateSourceRectangleForICanvasImageInterop(
    ICanvasResourceCreatorWithDpi* resourceCreator,
    ICanvasImageInterop* image,
    uint32_t sourceIndex,
    Rect const* invalidRectangle);

HRESULT GetInvalidRectanglesForICanvasImageInterop(
    ICanvasResourceCreatorWithDpi* resourceCreator,
    ICanvasImageInterop* image,
    uint32_t* valueCount,
    Rect** valueElements);

HRESULT GetRequiredSourceRectanglesForICanvasImageInterop(
    ICanvasResourceCreatorWithDpi* resourceCreator,
    ICanvasImageInterop* image,
    Rect const* outputRectangle,
    uint32_t sourceEffectCount,
    ICanvasEffect* const* sourceEffects,
    uint32_t sourceIndexCount,
    uint32_t const* sourceIndices,
    uint32_t sourceBoundsCount,
    Rect const* sourceBounds,
    uint32_t valueCount,
    Rect* valueElements);
```

Let's go over how they can be used:

- `InvalidateSourceRectangleForICanvasImageInterop` is meant to support `InvalidateSourceRectangle`. Simply marshal the input parameters and invoke it directly, and it'll take care of all the necessary work. Note that the `image` parameter is the current effect instance being implemented.
- `GetInvalidRectanglesForICanvasImageInterop` supports `GetInvalidRectangles`. This also requires no special consideration, other than needing to dispose the returned COM array once it's no longer needed.

- `GetRequiredSourceRectanglesForICanvasImageInterop` is a shared method that can support both `GetRequiredSourceRectangle` and `GetRequiredSourceRectangles`. That is, it takes a pointer to an existing array of values to populate, so callers can either pass a pointer to a single value (which can also be on the stack, to avoid one allocation), or to an array of values. The implementation is the same in both cases, so a single C export is enough to power both of them.

Custom effects in C# using ComputeSharp

As we mentioned, if you're using C# and want to implement a custom effect, the recommended approach is to use the ComputeSharp library. It enables you to both implement custom D2D1 pixel shaders entirely in C#, as well as to easily define custom effects graphs that are compatible with Win2D. The same library is also used in the Microsoft Store to power several graphics components in the application.

You can add a reference to ComputeSharp in your project through NuGet:

- On UWP, select the **ComputeSharp.D2D1.Uwp** package.
- On WinAppSDK, select the **ComputeSharp.D2D1.WinUI** package.

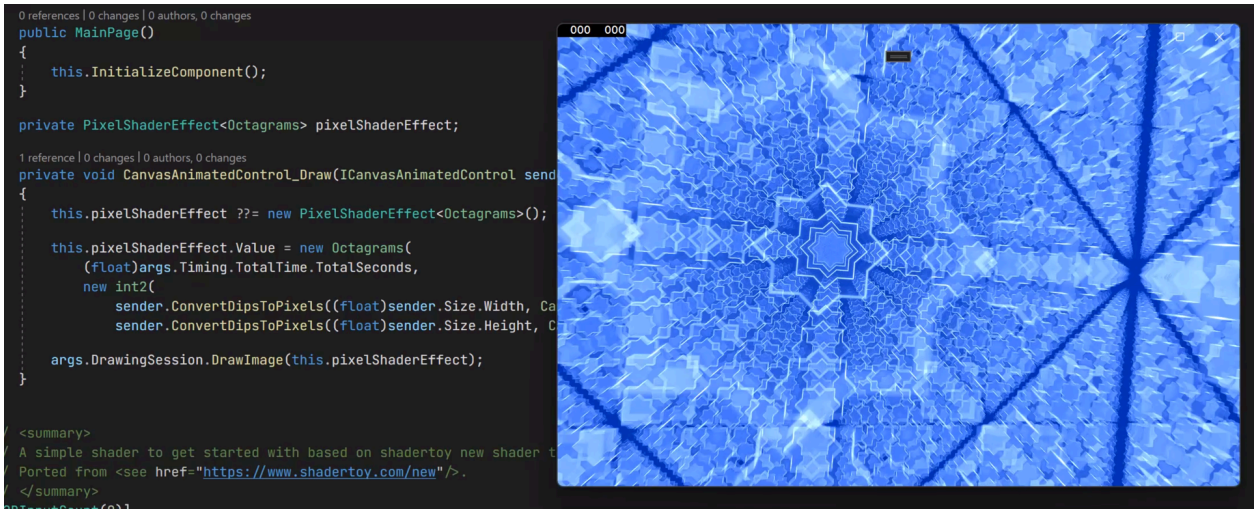
> **Note**
>
> Many APIs in ComputeSharp.D2D1.* are identical across the UWP and WinAppSDK targets, the only difference being the namespace (ending in either `.Uwp` or `.WinUI`). However, the UWP target is in sustained maintenance and not receiving new features. As such, some code changes might be needed compared to the samples shown here for WinUI. The snippets in this document reflect the API surface as of ComputeSharp.D2D1.WinUI 3.0.0 (the last release for the UWP target is instead 2.1.0).

There are two main components in ComputeSharp to interop with Win2D:

- `PixelShaderEffect<T>`: a Win2D effect that is powered by a D2D1 pixel shader. The shader itself is written in C# using the APIs provided by ComputeSharp. This class also provides properties to set effect sources, constant values, and more.
- `CanvasEffect`: a base class for custom Win2D effects that wraps an arbitrary effect graph. It can be used to "package" complex effects into an easy to use object that can be reused in several parts of an application.

Here is an example of a custom pixel shader (ported from this shadertoy shader), used with `PixelShaderEffect<T>` and then draw onto a Win2D `CanvasControl` (note that

`PixelShaderEffect<T>` implements `ICanvasImage`):



You can see how in just two lines of code you can create an effect and draw it via Win2D. ComputeSharp takes care of all the work necessary to compile the shader, register it, and manage the complex lifetime of a Win2D-compatible effect.

Next, let's see a step by step guide on how to create a custom Win2D effect that also uses a custom D2D1 pixel shader. We'll go over how to author a shader with ComputeSharp and setup its properties, and then how to create a custom effect graph packaged into a `CanvasEffect` type that can easily be reused in your application.

Designing the effect

For this demo, we want to create a simple frosted glass effect.

This will include the following components:

- Gaussian blur
- Tint effect
- Noise (which we can procedurally generate with a shader)

We'll also want to expose properties to control the blur and noise amount. The final effect will contain a "packaged" version of this effect graph and be easy to use by just creating an instance, setting those properties, connecting a source image, and then drawing it. Let's get started!

Creating a custom D2D1 pixel shader

For the noise on top of the effect, we can use a simple D2D1 pixel shader. The shader will compute a random value based on its coordinates (which will act as a "seed" for the

random number), and then it will use that noise value to compute the RGB amount for that pixel. We can then blend this noise on top of the resulting image.

To write the shader with ComputeSharp, we just need to define a `partial struct` type implementing the `ID2D1PixelShader` interface, and then write our logic in the `Execute` method. For this noise shader, we can write something like this:

C#

```
using ComputeSharp;
using ComputeSharp.D2D1;

[D2DInputCount(0)]
[D2DRequiresScenePosition]
[D2DShaderProfile(D2D1ShaderProfile.PixelShader40)]
[D2DGeneratedPixelShaderDescriptor]
public readonly partial struct NoiseShader(float amount) : ID2D1PixelShader
{
    /// <inheritdoc/>
    public float4 Execute()
    {
        // Get the current pixel coordinate (in pixels)
        int2 position = (int2)D2D.GetScenePosition().XY;

        // Compute a random value in the [0, 1] range for each target pixel.
This line just
        // calculates a hash from the current position and maps it into the
[0, 1] range.
        // This effectively provides a "random looking" value for each
pixel.
        float hash = Hlsl.Frac(Hlsl.Sin(Hlsl.Dot(position, new float2(41,
289))) * 45758.5453f);

        // Map the random value in the [0, amount] range, to control the
strength of the noise
        float alpha = Hlsl.Lerp(0, amount, hash);

        // Return a white pixel with the random value modulating the opacity
        return new(1, 1, 1, alpha);
    }
}
```

> **Note**
>
> While the shader is written entirely in C#, basic knowledge of HLSL (the programming language for DirectX shaders, which ComputeSharp transpiles C# to) is recommended.

Let's go over this shader in detail:

- The shader has no inputs, it just produces an infinite image with random grayscale noise.
- The shader requires access to the current pixel coordinate.
- The shader is precompiled at build time (using the `PixelShader40` profile, which is guaranteed to be available on any GPU where the application could be running).
- The `[D2DGeneratedPixelShaderDescriptor]` attribute is needed to trigger the source generator bundled with ComputeSharp, which will analyze the C# code, transpile it to HLSL, compile the shader to bytecode, etc.
- The shader captures a `float amount` parameter, via its primary constructor. The source generator in ComputeSharp will automatically take care of extracting all captured values in a shader and preparing the constant buffer that D2D needs to initialize the shader state.

And this part is done! This shader will generate our custom noise texture whenever needed. Next, we need to create our packaged effect with the effect graph connecting all our effects together.

Creating a custom effect

For our easy to use, packaged effect, we can use the `CanvasEffect` type from ComputeSharp. This type provides a straightforward way to setup all the necessary logic to create an effect graph and update it via public properties that users of the effect can interact with. There are two main methods we'll need to implement:

- `BuildEffectGraph`: this method is responsible for building the effect graph that we want to draw. That is, it needs to create all effects we need, and register the output node for the graph. For effects that can be updated at a later time, the registration is done with an associated `CanvasEffectNode<T>` value, which acts as lookup key to retrieve the effects from the graph when needed.
- `ConfigureEffectGraph`: this method refreshes the effect graph by applying the settings that the user has configured. This method is automatically invoked when needed, right before drawing the effect, and only if at least one effect property has been modified since the last time the effect was used.

Our custom effect can be defined as follows:

C#

```
using ComputeSharp.D2D1.WinUI;
using Microsoft.Graphics.Canvas;
using Microsoft.Graphics.Canvas.Effects;

public sealed class FrostedGlassEffect : CanvasEffect
```

```
{
    private static readonly CanvasEffectNode<GaussianBlurEffect> BlurNode =
new();
    private static readonly CanvasEffectNode<PixelShaderEffect<NoiseShader>>
NoiseNode = new();

    private ICanvasImage? _source;
    private double _blurAmount;
    private double _noiseAmount;

    public ICanvasImage? Source
    {
        get => _source;
        set => SetAndInvalidateEffectGraph(ref _source, value);
    }

    public double BlurAmount
    {
        get => _blurAmount;
        set => SetAndInvalidateEffectGraph(ref _blurAmount, value);
    }

    public double NoiseAmount
    {
        get => _noiseAmount;
        set => SetAndInvalidateEffectGraph(ref _noiseAmount, value);
    }

    /// <inheritdoc/>
    protected override void BuildEffectGraph(CanvasEffectGraph effectGraph)
    {
        // Create the effect graph as follows:
        //
        // ┌────────┐   ┌──────┐
        // │ source ├──►│ blur ├──────┐
        // └────────┘   └──────┘      ▼
        //                         ┌─────┐   ┌────────┐
        //                         │ blend ├──►│ output │
        //                         └─────┘   └────────┘
        //                            ▲
        //    ┌───────┐               │
        //    │ noise ├───────────────┘
        //    └───────┘
        //
        GaussianBlurEffect gaussianBlurEffect = new();
        BlendEffect blendEffect = new() { Mode = BlendEffectMode.Overlay };
        PixelShaderEffect<NoiseShader> noiseEffect = new();
        PremultiplyEffect premultiplyEffect = new();

        // Connect the effect graph
        premultiplyEffect.Source = noiseEffect;
        blendEffect.Background = gaussianBlurEffect;
        blendEffect.Foreground = premultiplyEffect;

        // Register all effects. For those that need to be referenced later
(ie. the ones with
```

```
        // properties that can change), we use a node as a key, so we can
perform lookup on
        // them later. For others, we register them anonymously. This allows
the effect
        // to autommatically and correctly handle disposal for all effects
in the graph.
        effectGraph.RegisterNode(BlurNode, gaussianBlurEffect);
        effectGraph.RegisterNode(NoiseNode, noiseEffect);
        effectGraph.RegisterNode(premultiplyEffect);
        effectGraph.RegisterOutputNode(blendEffect);
    }

    /// <inheritdoc/>
    protected override void ConfigureEffectGraph(CanvasEffectGraph
effectGraph)
    {
        // Set the effect source
        effectGraph.GetNode(BlurNode).Source = Source;

        // Configure the blur amount
        effectGraph.GetNode(BlurNode).BlurAmount = (float)BlurAmount;

        // Set the constant buffer of the shader
        effectGraph.GetNode(NoiseNode).ConstantBuffer = new
NoiseShader((float)NoiseAmount);
    }
}
```

You can see there are four sections in this class:

- First, we have fields to track all mutable state, such as the effects that can be updated as well as the backing fields for all the effect properties that we want to expose to users of the effect.
- Next, we have properties to configure the effect. The setter of each property uses the `SetAndInvalidateEffectGraph` method exposed by `CanvasEffect`, which will automatically invalidate the effect if the value being set is different than the current one. This ensures the effect is only configured again when really necessary.
- Lastly, we have the `BuildEffectGraph` and `ConfigureEffectGraph` methods we mentioned above.

> ⓘ **Note**
>
> The `PremultiplyEffect` node after the noise effect is very important: this is because Win2D effects assume that the output is premultiplied, whereas pixel shaders generally work with unpremultiplied pixels. As such, remember to manually insert premultiply/unpremultiply nodes before and after custom shaders, to ensure colors are correctly preserved.

> ⓘ **Note**
>
> This sample effect is using WinUI 3 namespaces, but the same code can be used on UWP as well. In that case, the namespace for ComputeSharp will be `ComputeSharp.Uwp`, matching the package name.

Ready to draw!

And with this, our custom frosted glass effect is ready! We can easily draw it as follows:

C#

```
private void CanvasControl_Draw(CanvasControl sender, CanvasDrawEventArgs
args)
{
    FrostedGlassEffect effect = new()
    {
        Source = _canvasBitmap,
        BlurAmount = 12,
        NoiseAmount = 0.1
    };

    args.DrawingSession.DrawImage(effect);
}
```

In this example, we're drawing the effect from the `Draw` handler of a `CanvasControl`, using a `CanvasBitmap` which we previously loaded as source. This is the input image we'll use to test the effect:

And here is the result:

> ⓘ **Note**
>
> Credits to Dominic Lange for the picture.

Additional resources

- Check out the Win2D source code for more information.
- For more information on ComputeSharp, check out the sample apps and the unit tests.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Push notifications overview

Article • 03/16/2023

Push notifications in the Windows App SDK use Windows Push Notifications Service (WNS) to send rich notifications to Windows apps using Azure App Registration identities.

Push notification types and usage scenarios

Push notifications can be used to enable several distinct features. The content and effect of a push notification will vary based on the way it is being used.

Raw notifications

Raw notifications are consumed by the app itself and are not communicated to the user. They can be used to control application behavior or notify applications of state changes remotely.

Scenario	Description	Example
Application Wake Up	Raw notifications can be used by app developers to wake up their application instead of it constantly running, which frees up user resources.	*Without raw notifications*: The Contoso Chat app runs in the background while waiting for a VOIP call. *With raw notifications*: The Contoso app process can be in a terminated state until a raw notification signals it and brings the process up, indicating a VOIP call has been initiated.
Real Time Sync	Raw notifications can replace polling scenarios by allowing the app developer to send payloads from their App Service to the App Client on the local device. These payloads notify the App Client to sync with the App Service.	*Without raw notifications*: The Contoso Chat app polls the Contoso cloud service every 30 mins to check for content updates, and initiates a sync if updates are available. *With raw notifications*: The Contoso Chat app is notified immediately when new content is available, and syncs that content right away.

App notifications from the cloud

App notifications are used to communicate with the user. The notification content is displayed in a transient window in the bottom right corner of the screen and in the Notification Center (called Action Center in Windows 10). App notifications can be used to inform the user of application status or state changes, or to prompt the user to take an action. App notifications can be either push (sent from the cloud) or sent locally. Sending a cloud-sourced app notification is similar to sending a raw notification, except the *X-WNS-Type* header is `toast`, *Content-Type* is `text/xml`, and the content contains the app notification XML payload, which you can learn more about here.

Limitations

The push notifications support in the Windows App SDK currently has these limitations:

- If your app is published as *self-contained* or is running with the elevated (admin) privilege, this feature may not be supported. In your app, use the `winrt::PushNotificationManager::IsSupported()` check as demonstrated in Quickstart: Push notifications in the Windows App SDK and implement a custom socket if the feature is unsupported.
- Microsoft reserves the right to disable or revoke apps from using push notifications.

Next steps

See Quickstart: Push notifications in the Windows App SDK to get started.

Related topics

- Windows Push Notification Service (WNS)
- Quickstart: Push notifications in the Windows App SDK
- Troubleshooting Windows App SDK
- Notifications UX Guidance

Quickstart: Push notifications in the Windows App SDK

Article • 05/21/2024

In this quickstart you will create a desktop Windows application that sends and receives push notifications using the Windows App SDK.

Prerequisites

- Start developing Windows apps
- Either Create a new project that uses the Windows App SDK OR Use the Windows App SDK in an existing project
- An Azure Account is required in order to use Windows App SDK push notifications.
- Read Push notifications overview

Sample app

This quickstart walks through adding push notifications support to your app. See the example code from this quickstart in context in the sample apps found on GitHub.

Sample App Code

API reference

For API reference documentation for push notifications, see Microsoft.Windows.PushNotifications Namespace.

Configure your app's identity in Azure Active Directory (AAD)

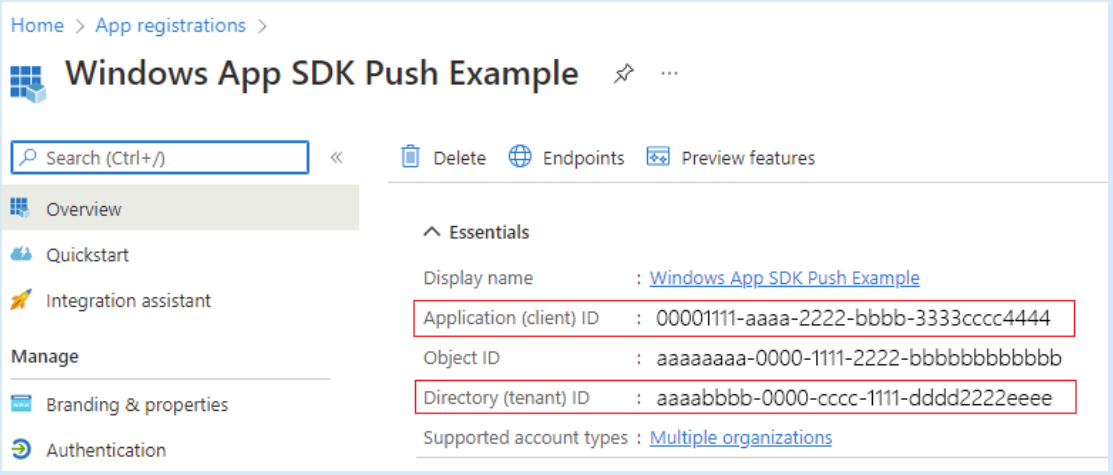
Push notifications in Windows App SDK use identities from Azure Active Directory (AAD). Azure credentials are required when requesting a WNS Channel URI and when requesting access tokens in order to send push notifications. **Note**: We do **NOT** support using Windows App SDK push notifications with Microsoft Partner Center.

Step 1: Create an AAD app registration

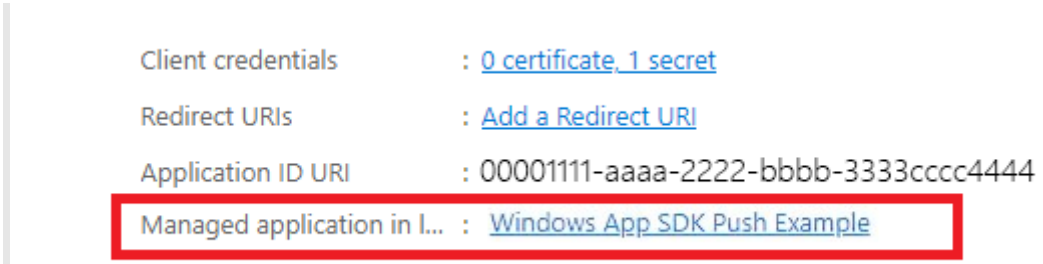
Login to your Azure account and create a new **AAD App Registration** resource. Select *New registration*.

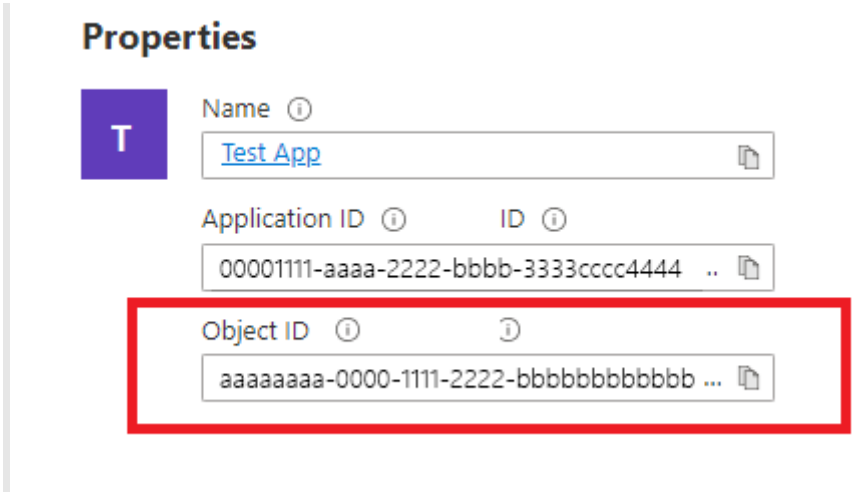
Step 2: Provide a name and select a multi-tenant option

1. Provide an app name.
2. Push notifications require the multi-tenant option, so select that.
   a. For more information about tenants, see Who can sign in to your app?.
3. Select *Register*
4. Take note of your **Application (client) ID**, as this is your **Azure AppId** that you will be using during activation registration and access token request.
5. Take note of your **Directory (tenant) ID**, as this is your **Azure TenantId** that you will be using when requesting an access token.

> **Important**
>
> 
>
>
> Take note of your **Application (client) ID** and **Directory (tenant) ID**.

6. Take note of your **Object ID**, as this is your **Azure ObjectId** that you will be using when requesting a channel request. Note that this is NOT the object ID listed on the **Essentials** page. Instead, to find the correct **Object ID**, click on the app name in the **Managed application in local directory** field on the **Essentials** page:

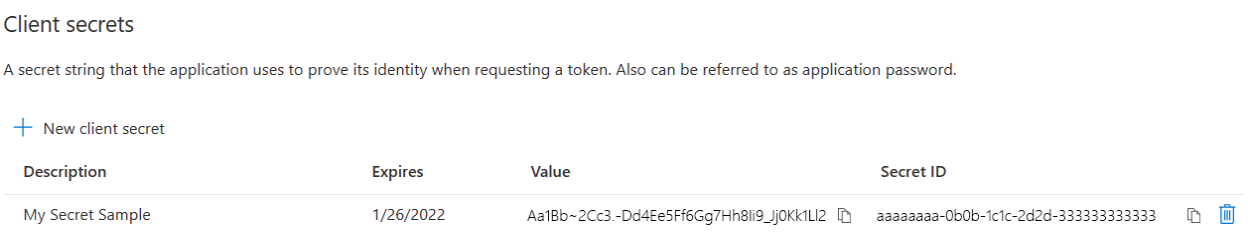




> **Note**
>
> A **service principal** is required to get an Object ID, if there is not one associated with your app, follow the steps in one of the following articles to create one in the Azure portal or using the command line:
>
> Use the portal to create an Azure AD application and service principal that can access resources
>
> Use Azure PowerShell to create a service principal with a certificate

Step 3: Create a secret for your app registration

Your secret will be used along with your Azure AppId/ClientId when requesting an access token to send push notifications.



Navigate to **Certificates & secrets** and select **New client secret**.

> **Important**
>
> Ensure you copy your secret once created and store it in a safe location, like Azure Key Vault. It will only be viewable once right after creation.

Step 4: Map your app's Package Family Name to its Azure AppId

If your app is packaged (including packaged with an external location), you can use this flow to map your app's Package Family Name (PFN) and its Azure AppId.

If your app is a packaged Win32 app, then create a Package Family Name (PFN) mapping request by emailing Win_App_SDK_Push@microsoft.com with subject line "Windows App SDK Push Notifications Mapping Request" and body "PFN: [your PFN]", AppId: [your APPId], ObjectId: [your ObjectId]. Mapping requests are completed on a weekly basis. You will be notified once your mapping request has been completed.

Configure your app to receive push notifications

Step 1: Add namespace declarations

Add the namespace for Windows App SDK push notifications `Microsoft.Windows.PushNotifications`.

C++

```
#include <winrt/Microsoft.Windows.PushNotifications.h>

using namespace winrt::Microsoft::Windows::PushNotifications;
```

Step 2: Add your COM activator to your app's manifest

> **Important**
>
> If your app is unpackaged (that is, it lacks package identity at runtime), then skip to **Step 3: Register for and respond to push notifications on app startup**.

If your app is packaged (including packaged with external location): Open your **Package.appxmanifest**. Add the following inside the `<Application>` element. Replace the `Id`, `Executable`, and `DisplayName` values with those specific to your app.

XAML

```
<!--Packaged apps only-->
<!--package.appxmanifest-->

<Package
  ...
  xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"
  ...
  <Applications>
    <Application>
      ...
      <Extensions>

        <!--Register COM activator-->
        <com:Extension Category="windows.comServer">
          <com:ComServer>
              <com:ExeServer Executable="SampleApp\SampleApp.exe"
DisplayName="SampleApp" Arguments="----WindowsAppRuntimePushServer:">
                <com:Class Id="[Your app's Azure AppId]"
DisplayName="Windows App SDK Push" />
            </com:ExeServer>
          </com:ComServer>
        </com:Extension>

      </Extensions>
    </Application>
  </Applications>
 </Package>
```

Step 3: Register for and respond to push notifications on app startup

Update your app's `main()` method to add the following:

1. Register your app to receive push notifications by calling PushNotificationManager::Default().Register().
2. Check the source of the activation request by calling AppInstance::GetCurrent().GetActivatedEventArgs(). If the activation was triggered from a push notification, respond based on the notification's payload.

> **ⓘ Important**

You must call **PushNotificationManager::Default().Register** before calling [AppInstance.GetCurrent.GetActivatedEventArgs](#).

The following sample is from the sample packaged app found on GitHub.

C++

```
// cpp-console.cpp
#include "pch.h"
#include <iostream>
#include <winrt/Microsoft.Windows.PushNotifications.h>
#include <winrt/Microsoft.Windows.AppLifecycle.h>
#include <winrt/Windows.Foundation.h>
#include <wil/result.h>
#include <wil/cppwinrt.h>


using namespace winrt;
using namespace Windows::Foundation;

using namespace winrt::Microsoft::Windows::PushNotifications;
using namespace winrt::Microsoft::Windows::AppLifecycle;

winrt::guid remoteId{ "7edfab6c-25ae-4678-b406-d1848f97919a" }; // Replace
this with your own Azure ObjectId



void SubscribeForegroundEventHandler()
{
    winrt::event_token token{
PushNotificationManager::Default().PushReceived([](auto const&,
PushNotificationReceivedEventArgs const& args)
    {
        auto payload{ args.Payload() };

        std::string payloadString(payload.begin(), payload.end());
        std::cout << "\nPush notification content received in the
FOREGROUND: " << payloadString << std::endl;
    }) };
}

int main()
{
    // Setup an event handler, so we can receive notifications in the
foreground while the app is running.
    SubscribeForegroundEventHandler();

    PushNotificationManager::Default().Register();

    auto args{ AppInstance::GetCurrent().GetActivatedEventArgs() };
    switch (args.Kind())
    {
```

```
        // When it is launched normally (by the users, or from the
debugger), the sample requests a WNS Channel URI and
        // displays it, then waits for notifications. This user can take a
copy of the WNS Channel URI and use it to send
        // notifications to the sample
        case ExtendedActivationKind::Launch:
        {
            // Checks to see if push notifications are supported. Certain
self-contained apps may not support push notifications by design
            if (PushNotificationManager::IsSupported())
            {
                // Request a WNS Channel URI which can be passed off to an
external app to send notifications to.
                // The WNS Channel URI uniquely identifies this app for this
user and device.
                PushNotificationChannel channel{ RequestChannel() };
                if (!channel)
                {
                    std::cout << "\nThere was an error obtaining the WNS
Channel URI" << std::endl;

                    if (remoteId == winrt::guid { "00000000-0000-0000-0000-
000000000000" })
                    {
                        std::cout << "\nThe ObjectID has not been set. Refer
to the readme file accompanying this sample\nfor the instructions on how to
obtain and setup an ObjectID" << std::endl;
                    }
                }

                std::cout << "\nPress 'Enter' at any time to exit App." <<
std::endl;
                std::cin.ignore();
            }
            else
            {
                // App implements its own custom socket here to receive
messages from the cloud since Push APIs are unsupported.
            }
        }
        break;

        // When it is activated from a push notification, the sample only
displays the notification.
        // It doesn't register for foreground activation of perform any
other actions
        // because background activation is meant to let app perform only
small tasks in order to preserve battery life.
        case ExtendedActivationKind::Push:
        {
            PushNotificationReceivedEventArgs pushArgs{
args.Data().as<PushNotificationReceivedEventArgs>() };

            // Call GetDeferral to ensure that code runs in low power
            auto deferral{ pushArgs.GetDeferral() };
```

```
            auto payload{ pushArgs.Payload() } ;

            // Do stuff to process the raw notification payload
            std::string payloadString(payload.begin(), payload.end());
            std::cout << "\nPush notification content received in the
BACKGROUND: " << payloadString.c_str() << std::endl;
            std::cout << "\nPress 'Enter' to exit the App." << std::endl;

            // Call Complete on the deferral when finished processing the
payload.
            // This removes the override that kept the app running even when
the system was in a low power mode.
            deferral.Complete();
            std::cin.ignore();
        }
        break;

        default:
            std::cout << "\nUnexpected activation type" << std::endl;
            std::cout << "\nPress 'Enter' to exit the App." << std::endl;
            std::cin.ignore();
            break;
    }

    // We do not call PushNotificationManager::UnregisterActivator
    // because then we wouldn't be able to receive background activations,
once the app has closed.
    // Call UnregisterActivator once you don't want to receive push
notifications anymore.
}
```

Step 4: Request a WNS Channel URI and register it with the WNS server

WNS Channel URIs are the HTTP endpoints for sending push notifications. Each client must request a Channel URI and register it with the WNS server to receive push notifications.

> ⓘ **Note**
>
> WNS Channel URIs expire after 30 days.

C++

```
auto channelOperation{
PushNotificationManager::Default().CreateChannelAsync(winrt::guid("[Your
```

```
app's Azure ObjectID]")) };
```

The **PushNotificationManager** will attempt to create a Channel URI, retrying automatically for no more than 15 minutes. Create an event handler to wait for the call to complete. Once the call is complete, if it was successful, register the URI with the WNS server.

C++

```
// cpp-console.cpp

winrt::Windows::Foundation::IAsyncOperation<PushNotificationChannel>
RequestChannelAsync()
{
    // To obtain an AAD RemoteIdentifier for your app,
    // follow the instructions on https://learn.microsoft.com/azure/active-
directory/develop/quickstart-register-app
    auto channelOperation =
PushNotificationManager::Default().CreateChannelAsync(remoteId);

    // Setup the inprogress event handler
    channelOperation.Progress(
        [](auto&& sender, auto&& args)
        {
            if (args.status == PushNotificationChannelStatus::InProgress)
            {
                // This is basically a noop since it isn't really an error
state
                std::cout << "Channel request is in progress." << std::endl
<< std::endl;
            }
            else if (args.status ==
PushNotificationChannelStatus::InProgressRetry)
            {
                LOG_HR_MSG(
                    args.extendedError,
                    "The channel request is in back-off retry mode because
of a retryable error! Expect delays in acquiring it. RetryCount = %d",
                    args.retryCount);
            }
        });

    auto result = co_await channelOperation;

    if (result.Status() == PushNotificationChannelStatus::CompletedSuccess)
    {
        auto channelUri = result.Channel().Uri();

        std::cout << "channelUri: " <<
winrt::to_string(channelUri.ToString()) << std::endl << std::endl;

        auto channelExpiry = result.Channel().ExpirationTime();
```

```
        // Caller's responsibility to keep the channel alive
        co_return result.Channel();
    }
    else if (result.Status() ==
PushNotificationChannelStatus::CompletedFailure)
    {
        LOG_HR_MSG(result.ExtendedError(), "We hit a critical non-retryable
error with channel request!");
        co_return nullptr;
    }
    else
    {
        LOG_HR_MSG(result.ExtendedError(), "Some other failure occurred.");
        co_return nullptr;
    }

};

PushNotificationChannel RequestChannel()
{
    auto task = RequestChannelAsync();
    if (task.wait_for(std::chrono::seconds(300)) != AsyncStatus::Completed)
    {
        task.Cancel();
        return nullptr;
    }

    auto result = task.GetResults();
    return result;
}
```

Step 5: Build and install your app

Use Visual Studio to build and install your app. Right click on the solution file in the Solution Explorer and select **Deploy**. Visual Studio will build your app and install it on your machine. You can run the app by launching it via the Start Menu or the Visual Studio debugger.

Send a push notification to your app

At this point, all configuration is complete and the WNS server can send push notifications to client apps. In the following steps, refer to the Push notification server request and response headers for more detail.

Step 1: Request an access token

To send a push notification, the WNS server first needs to request an access token. Send an HTTP POST request with your Azure TenantId, Azure AppId, and secret. For information on retrieving the Azure TenantId and Azure AppId, see Get tenant and app ID values for signing in.

HTTP Sample Request:

```HTTP
POST /{tenantID}/oauth2/v2.0/token Http/1.1
Host: login.microsoftonline.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 160

grant_type=client_credentials&client_id=
<Azure_App_Registration_AppId_Here>&client_secret=
<Azure_App_Registration_Secret_Here>&scope=https://wns.windows.com/.default/
```

C# Sample Request:

```C#
//Sample C# Access token request
var client = new
RestClient("https://login.microsoftonline.com/{tenantID}/oauth2/v2.0");
var request = new RestRequest("/token", Method.Post);
request.AddHeader("Content-Type", "application/x-www-form-urlencoded");
request.AddParameter("grant_type", "client_credentials");
request.AddParameter("client_id", "[Your app's Azure AppId]");
request.AddParameter("client_secret", "[Your app's secret]");
request.AddParameter("scope", "https://wns.windows.com/.default");
RestResponse response = await client.ExecutePostAsync(request);
Console.WriteLine(response.Content);
```

If your request is successful, you will receive a response that contains your token in the **access_token** field.

```JSON
{
    "token_type":"Bearer",
    "expires_in":"86399",
    "ext_expires_in":"86399",
    "expires_on":"1653771789",
    "not_before":"1653685089",
    "access_token":"[your access token]"
}
```

Step 2. Send a raw notification

Create an HTTP POST request that contains the access token you obtained in the previous step and the content of the push notification you want to send. The content of the push notification will be delivered to the app.

HTTP

```
POST /?token=[The token query string parameter from your channel URL. E.g. AwYAAABa5cJ3...] HTTP/1.1
Host: dm3p.notify.windows.com
Content-Type: application/octet-stream
X-WNS-Type: wns/raw
Authorization: Bearer [your access token]
Content-Length: 46

{ Sync: "Hello from the Contoso App Service" }
```

C#

```
var client = new RestClient("[Your channel URL. E.g. https://wns2-by3p.notify.windows.com/?token=AwYAAABa5cJ3...]");
var request = new RestRequest();
request.Method = Method.Post;
request.AddHeader("Content-Type", "application/octet-stream");
request.AddHeader("X-WNS-Type", "wns/raw");
request.AddHeader("Authorization", "Bearer [your access token]");
request.AddBody("Notification body");
RestResponse response = await client.ExecutePostAsync(request);");
```

Step 3: Send a cloud-sourced app notification

If you are only interested in sending raw notifications, disregard this step. To send a cloud-sourced app notification, also known a push toast notification, first follow Quickstart: App notifications in the Windows App SDK. App notifications can either be push (sent from the cloud) or sent locally. Sending a cloud-sourced app notification is similar to sending a raw notification in **Step 2**, except the *X-WNS-Type* header is `toast`, *Content-Type* is `text/xml`, and the content contains the app notification XML payload. See Notifications XML schema for more on how to construct your XML payload.

Create an HTTP POST request that contains your access token and the content of the cloud-sourced app notification you want to send. The content of the push notification will be delivered to the app.

HTTP

```
POST /?
token=AwYAAAB%2fQAhYEiAESPobjHzQcwGCTjHu%2f%2fP3CCNDcyfyvgbK5xD3kztniW%2bjba
1b3aSSun58SA326GMxuzZooJYwtpgzL9AusPDES2alyQ8CHvW94cO5VuxxLDVzrSzdO1ZVgm%2bN
SB9BAzOASvHqkMHQhsDy HTTP/1.1
Host: dm3p.notify.windows.com
Content-Type: text/xml
X-WNS-Type: wns/toast
Authorization: Bearer [your access token]
Content-Length: 180

<toast><visual><binding template="ToastGeneric"><text>Example cloud toast
notification</text><text>This is an example cloud notification using
XML</text></binding></visual></toast>
```

C#

```
var client = new RestClient("https://dm3p.notify.windows.com/?
token=AwYAAAB%2fQAhYEiAESPobjHzQcwGCTjHu%2f%2fP3CCNDcyfyvgbK5xD3kztniW%2bjba
1b3aSSun58SA326GMxuzZooJYwtpgzL9AusPDES2alyQ8CHvW94cO5VuxxLDVzrSzdO1ZVgm%2bN
SB9BAzOASvHqkMHQhsDy");
client.Timeout = -1;

var request = new RestRequest(Method.POST);
request.AddHeader("Content-Type", "text/xml");
request.AddHeader("X-WNS-Type", "wns/toast");
request.AddHeader("Authorization", "Bearer <AccessToken>");
request.AddParameter("text/xml", "<toast><visual><binding
template=\"ToastGeneric\"><text>Example cloud toast notification</text>
<text>This is an example cloud notification using XML</text></binding>
</visual></toast>",  ParameterType.RequestBody);
Console.WriteLine(response.Content);
```

Resources

- Windows Push Notification Service (WNS)
- Push notifications sample code on GitHub
- Microsoft.Windows.PushNotifications API details
- Push notifications spec on GitHub
- Toast content
- Notifications XML schema

Troubleshooting push notifications

Article • 06/03/2022

WNS channel request errors

In case the **CreateChannelAsync** call fails, these are the common HRESULTS and our recommended actions.

HRESULT	Definition	Description
0x880403E8L	WNP_E_NOT_CONNECTED	The app is not connected to the WNS Server after retries.
0x880403E9L	WNP_E_RECONNECTING	The app is in the process of reconnecting to the WNS Server. Try requesting a channelURI again after several minutes.
0x880403FEL	WNP_E_BIND_USER_BUSY	The WNS client is having connectivity issues with the WNS server. Try requesting a channelURI again after several minutes.

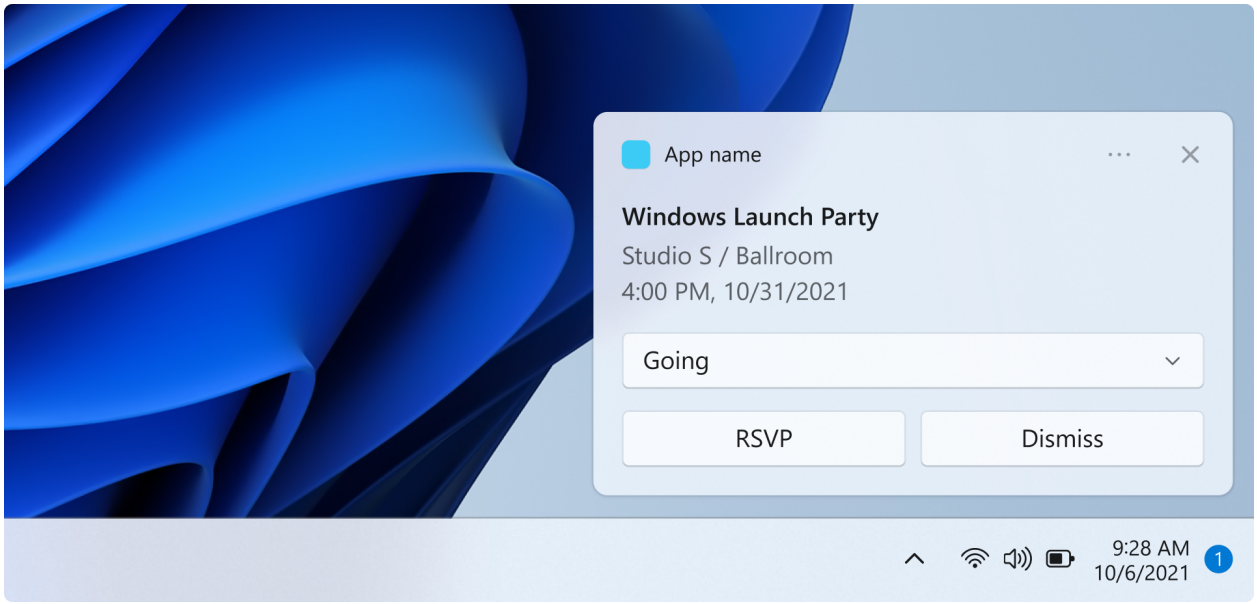
WNS HTTP response codes

HTTP response code	Description	Recommended action
200 Ok	The notification was accepted by WNS.	None required.
400 Bad Request	One or more headers were specified incorrectly or conflict with another header.	Log the details of your request. Inspect your request and compare against this documentation.
401 Unauthorized	The cloud service did not present a valid authentication ticket. The OAuth ticket may be invalid.	Request a valid access token by authenticating your cloud service using the access token request.
403 Forbidden	The cloud service is not authorized to send a notification to this URI even though they are authenticated.	The access token provided in the request does not match the credentials of the app that requested the channel URI. Ensure that your package name in your app's manifest matches

HTTP response code	Description	Recommended action
		the cloud service credentials given to your app in the Dashboard.
404 Not Found	The channel URI is not valid or is not recognized by WNS.	Log the details of your request. Do not send further notifications to this channel; notifications to this address will fail.
405 Method Not Allowed	Invalid method (GET, CREATE); only POST	Log the details of your request. Switch to using HTTP POST.
406 Not Acceptable	The cloud service exceeded its throttle limit.	Log the details of your request. Reduce the rate at which you are sending notifications.
410 Gone	The channel expired.	Log the details of your request. Do not send further notifications to this channel. Have your app request a new channel URI.
413 Request Entity Too Large	The notification payload exceeds the 5000 byte size limit.	Log the details of your request. Inspect the payload to ensure it is within the size limitations.
429 Monthly Quota Exceeded	The app is over the monthly quota limit.	Wait until monthly quota limit is reset or move to a higher WNS tier.
500 Internal Server Error	An internal failure caused notification delivery to fail.	Log the details of your request. Report this issue in the Windows App SDK Issues with the *area-Notifications* label.
503 Service Unavailable	The server is currently unavailable.	Log the details of your request. Report this issue in the Windows App SDK Issues with the *area-Notifications* label.

App notifications overview

Article • 07/12/2022



App notifications in the Windows App SDK are messages that your app can construct and deliver to your user while they are not currently inside your app. The notification content is displayed in a transient window in the bottom right corner of the screen and in the Notification Center (called Action Center in Windows 10). App notifications can be used to inform the user of application status or state changes, or to prompt the user to take an action. App notifications can either be sent locally or from a cloud service using push notifications.

Limitations

The app notifications support in the Windows App SDK currently has these limitations:

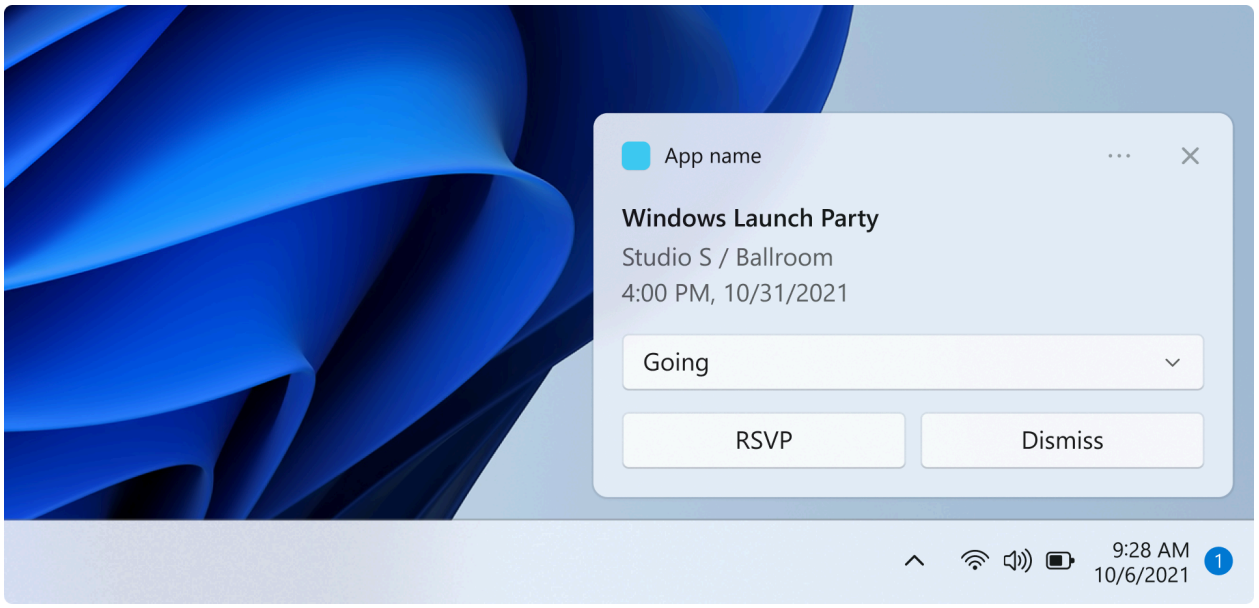
- Notifications for an elevated (admin) app are currently not supported.

Next steps

See Quickstart: App notifications in the Windows App SDK to get started.

Quickstart: App notifications in the Windows App SDK

Article • 03/10/2023



In this quickstart, you will create a desktop Windows application that sends and receives local app notifications, also known as toast notifications, using the Windows App SDK.

> **Important**
>
> Notifications for an elevated (admin) app is currently not supported.

Prerequisites

- Get started with WinUI
- Either Create a new project that uses the Windows App SDK OR Use the Windows App SDK in an existing project

Sample app

This quickstart covers code from the notifications sample apps found on GitHub.

Sample App Code

API reference

For API reference documentation for app notifications, see Microsoft.Windows.AppNotifications Namespace.

Step 1: Add namespace declarations

Add the namespace for Windows App SDK app notifications `Microsoft.Windows.AppNotifications`.

```
C#

using Microsoft.Windows.AppNotifications;
```

Step 2: Update your app's manifest

If your app is unpackaged (that is, it lacks package identity at runtime), then skip to **Step 3: Register to handle an app notification**.

If your app is packaged (including packaged with external location):

1. Open your **Package.appxmanifest**.
2. Add `xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"` and `xmlns:desktop="http://schemas.microsoft.com/appx/manifest/desktop/windows10"` namespaces to `<Package>`
3. Add `<desktop:Extension>` for `windows.toastNotificationActivation` to declare your COM activator CLSID. You can obtain a CLSID by navigating to **Create GUID** under **Tools** in Visual Studio.
4. Add `<com:Extension>` for the COM activator using the same CLSID.
   a. Specify your .exe file in the `Executable` attribute. The .exe file must be the same process calling `Register()` when registering your app for notifications, which is described more in **Step 3**. In the example below, we use `Executable="SampleApp\SampleApp.exe"`.
   b. Specify `Arguments="----AppNotificationActivated:"` to ensure that Windows App SDK can process your notification's payload as an AppNotification kind.
   c. Specify a `DisplayName`.

> **ⓘ Important**

Warning: If you define a Windows.Protocol app extensibility type in your appx manifest with `<uap:Protocol>`, then clicking on notifications will launch new processes of the same app, even if your app is already running.

XAML

```
<!--Packaged apps only-->
<!--package.appxmanifest-->

<Package
  xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"

xmlns:desktop="http://schemas.microsoft.com/appx/manifest/desktop/windows10"
  ...
  <Applications>
    <Application>
      ...
      <Extensions>

        <!--Specify which CLSID to activate when notification is clicked-->
        <desktop:Extension Category="windows.toastNotificationActivation">
          <desktop:ToastNotificationActivation
ToastActivatorCLSID="replaced-with-your-guid-C173E6ADF0C3" />
        </desktop:Extension>

        <!--Register COM CLSID-->
        <com:Extension Category="windows.comServer">
          <com:ComServer>
            <com:ExeServer Executable="SampleApp\SampleApp.exe"
DisplayName="SampleApp" Arguments="----AppNotificationActivated:">
              <com:Class Id="replaced-with-your-guid-C173E6ADF0C3" />
            </com:ExeServer>
          </com:ComServer>
        </com:Extension>

      </Extensions>
    </Application>
  </Applications>
 </Package>
```

Step 3: Register to handle an app notification

Register your app to handle notifications, then unregister when your app terminates.

In your `App.xaml` file, register for AppNotificationManager::Default().NotificationInvoked, then call AppNotificationManager::Default().Register. The order of these calls matters.

ⓘ Important

You must call **AppNotificationManager::Default().Register** before calling AppInstance.GetCurrent.GetActivatedEventArgs.

When your app is terminating, call AppNotificationManager::Default().Unregister() to free up the COM server and allow for subsequent invokes to launch a new process.

C#

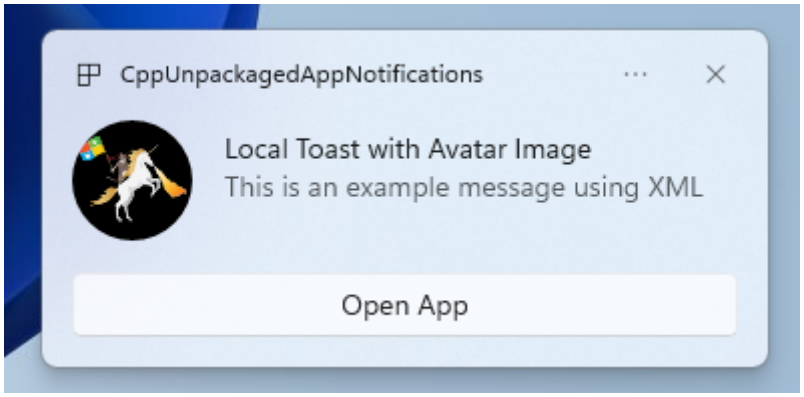
cs

```
// App.xaml.cs
namespace CsUnpackagedAppNotifications
{

    public partial class App : Application
    {
        private Window mainWindow;
        private NotificationManager notificationManager;

        public App()
        {
            this.InitializeComponent();
            notificationManager = new NotificationManager();
            AppDomain.CurrentDomain.ProcessExit += new
EventHandler(OnProcessExit);
        }

        protected override void OnLaunched(LaunchActivatedEventArgs
args)
        {
            mainWindow = new MainWindow();

            notificationManager.Init();

            // Complete in Step 5

            mainWindow.Activate();
        }

        void OnProcessExit(object sender, EventArgs e)
        {
            notificationManager.Unregister();
        }
    }
}


// NotificationManager.cs
namespace CsUnpackagedAppNotifications
{
    internal class NotificationManager
```

```
    {
        private bool m_isRegistered;

        private Dictionary<int,
Action<AppNotificationActivatedEventArgs>> c_map;

        public NotificationManager()
        {
            m_isRegistered = false;

            // When adding new a scenario, be sure to add its
notification handler here.
            c_map = new Dictionary<int,
Action<AppNotificationActivatedEventArgs>>();
            c_map.Add(ToastWithAvatar.ScenarioId,
ToastWithAvatar.NotificationReceived);
            c_map.Add(ToastWithTextBox.ScenarioId,
ToastWithTextBox.NotificationReceived);
        }

        ~NotificationManager()
        {
            Unregister();
        }

        public void Init()
        {
            // To ensure all Notification handling happens in this
process instance, register for
            // NotificationInvoked before calling Register(). Without
this a new process will
            // be launched to handle the notification.
            AppNotificationManager notificationManager =
AppNotificationManager.Default;

            notificationManager.NotificationInvoked +=
OnNotificationInvoked;

            notificationManager.Register();
            m_isRegistered = true;
        }

        public void Unregister()
        {
            if (m_isRegistered)
            {
                AppNotificationManager.Default.Unregister();
                m_isRegistered = false;
            }
        }

        public void
ProcessLaunchActivationArgs(AppNotificationActivatedEventArgs
notificationActivatedEventArgs)
        {
```

```
            // Complete in Step 5
        }

    }
}
```

Step 4: Display an app notification



You **MUST** complete **Step 3: Register to handle an app notification** before proceeding.

Now you will display a simple app notification with an `appLogoOverride` image and a button.

Construct your app notification using the **AppNotificationBuilder** class and then call `Show`. For more information on how to construct your app notification using XML, please refer to the examples at Toast content and the Notifications XML schema.

> **Note**
>
> If your app is packaged (including packaged with external location), then your app's icon in the notification's upper left corner is sourced from the `package.manifest`. If your app is unpackaged, then the icon is sourced by first looking into the shortcut, then looking at the resource file in the app process. If all attempts fail, then the Windows default app icon is used. The supported icon file types are `.jpg`, `.png`, `.bmp`, and `.ico`.

C#

C#

```
// ToastWithAvatar.cs
class ToastWithAvatar
```

```
{
    public const int ScenarioId = 1;
    public const string ScenarioName = "Local Toast with Avatar Image";

    public static bool SendToast()
    {
        var appNotification = new AppNotificationBuilder()
            .AddArgument("action", "ToastClick")
            .AddArgument(Common.scenarioTag, ScenarioId.ToString())
            .SetAppLogoOverride(new System.Uri("file://" +
App.GetFullPathToAsset("Square150x150Logo.png")),
AppNotificationImageCrop.Circle)
            .AddText(ScenarioName)
            .AddText("This is an example message using XML")
            .AddButton(new AppNotificationButton("Open App")
                .AddArgument("action", "OpenApp")
                .AddArgument(Common.scenarioTag, ScenarioId.ToString()))
            .BuildNotification();

        AppNotificationManager.Default.Show(appNotification);

        return appNotification.Id != 0; // return true (indicating
success) if the toast was sent (if it has an Id)
    }

    public static void
NotificationReceived(AppNotificationActivatedEventArgs
notificationActivatedEventArgs)
    {
        // Complete in Step 5
    }
}

// Call SendToast() to send a notification.
```

Step 5: Process a user selecting a notification

Users can select your notification's body or button. Your app needs to process the invocation in response to a user interacting with your notification.

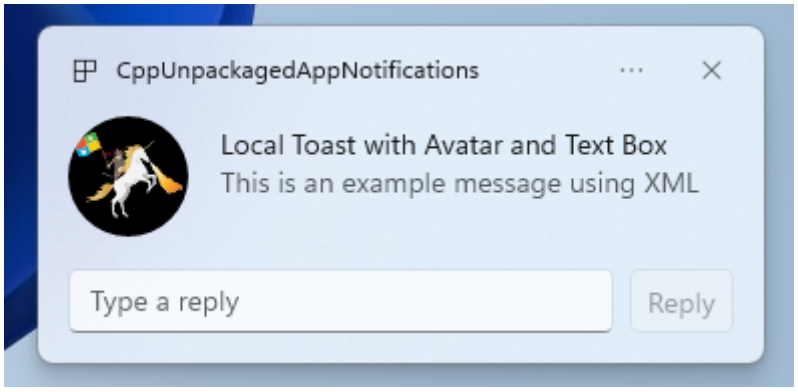
There are 2 common ways to process this:

1. You choose to have your app launch in a specific UI context OR
2. You choose to have your app evaluate an action-specific behavior (like a button press in the notification body) without rendering any UI. Also known as a background action.

The code example below, which is not from the sample app, illustrates both ways of processing a user-generated action. Add a `launch` value (corresponds to user clicking

the notification body), an `input` element (quick reply text box), and a button with an `arguments` value (corresponds to user clicking the button) to your notification's XML payload. In your `ProcessLaunchActivationArgs`, case on each argument.

> **ⓘ Important**
>
> Setting `activationType="background"` in the notification XML payload is ignored for desktop apps. You must instead process the activation arguments and decide whether to display a window or not, as stated in this step.



C++

```
// Example of how to process a user either selecting the notification body
or inputting a quick reply in the text box.

// Notification XML payload
//<toast launch="action=openThread&amp;threadId=92187">
//  <visual>
//      <binding template="ToastGeneric">
//          <image placement="appLogoOverride" hint-crop="circle" src="C:\
<fullpath>\Logo.png"/>
//          <text>Local Toast with Avatar and Text box</text>
//          <text>This is an example message using</text>
//      </binding>
//  </visual>
//  <actions>
//      <input id="replyBox" type="text" placeHolderContent="Reply" />
//      <action
//          content="Send"
//          hint-inputId="replyBox"
//          arguments="action=reply&amp;threadId=92187" />
//  </actions>
//</toast>

void ProcessLaunchActivationArgs(const
winrt::AppNotificationActivatedEventArgs& notificationActivatedEventArgs)
{
    // If the user clicks on the notification body, your app needs to launch
the chat thread window
```

```
    if
(std::wstring(notificationActivatedEventArgs.Argument().c_str()).find(L"open
Thread") != std::wstring::npos)
    {
        GenerateChatThreadWindow();
    }
    else // If the user responds to a message by clicking a button in the
notification, your app needs to reply back to the other user with no window
launched
    if
(std::wstring(notificationActivatedEventArgs.Argument().c_str()).find(L"repl
y") != std::wstring::npos)
    {
        auto input = notificationActivatedEventArgs.UserInput();
        auto replyBoxText = input.Lookup(L"replyBox");

        // Process the reply text
        SendReplyToUser(replyBoxText);
    }
}
```

Follow the below guidelines:

1. If a notification is selected by the user and your app is not running, it is expected that your app is launched and the user can see the foreground window in the notification's context.
2. If a notification is selected by the user and your app is minimized, it is expected that your app is brought to the foreground and a new window is rendered in the notification's context.
3. If a notification background action is invoked by the user (e.g. the user responds to a notification by typing in the notification text box and hitting reply), your app processes the payload without rendering a foreground window.

See the sample app code found on GitHub for a more detailed example.

Step 6: Remove notifications

Remove notifications when they are no longer relevant to the user.

In this example, the user has seen all messages from a group chat in your app, so you clear all notifications from the group chat. Then, the user mutes a friend, so you clear all notifications from the friend. You first added the **Group** and **Tag** properties to the notifications before displaying in order to identify them now.

```
C++
```

```
void SendNotification(winrt::hstring const& payload, winrt::hstring const&
friendId, winrt::hstring const& groupChatId)
{
    winrt::AppNotification notification(payload);

    // Setting Group Id here allows clearing notifications from a specific
chat group later
    notification.Group(groupChatId);

    // Setting Tag Id here allows clearing notifications from a specific
friend later
    notification.Tag(friendId);

    winrt::AppNotificationManager::Default().Show(notification);
}

winrt::Windows::Foundation::IAsyncAction
RemoveAllNotificationsFromGroupChat(const std::wstring groupChatId)
{
    winrt::AppNotificationManager manager =
winrt::AppNotificationManager::Default();
    co_await manager.RemoveByGroupAsync(groupChatId);
}

winrt::Windows::Foundation::IAsyncAction
RemoveAllNotificationsFromFriend(const std::wstring friendId)
{
    winrt::AppNotificationManager manager =
winrt::AppNotificationManager::Default();
    co_await manager.RemoveByTagAsync(friendId);
}
```

Additional features

Send a cloud-sourced app notification

To send an app notification from the cloud, follow **Send a cloud-sourced app notification** at Quickstart: Push notifications in the Windows App SDK.

Set an expiration time

Set an expiration time on your app notification using the `Expiration` property if the message in your notification is only relevant for a certain period of time. For example, if you send a calendar event reminder, set the expiration time to the end of the calendar event.

> **Note**
>
> The default and maximum expiration time is 3 days.

C#

```
class ToastWithAvatar
{
    public static bool SendToast()
    {

        var appNotification = new AppNotificationBuilder()
            .SetAppLogoOverride(new System.Uri("ms-
appx:///images/logo.png"), AppNotificationImageCrop.Circle)
            .AddText("Example expiring notification")
            .AddText("This is an example message")
            .BuildNotification();


        appNotification.Expiration = DateTime.Now.AddDays(1);
        AppNotificationManager.Default.Show(appNotification);

        return appNotification.Id != 0; // return true (indicating success)
if the toast was sent (if it has an Id)
    }
}
```

Ensure notifications expire on reboot

Set the `ExpiresOnReboot` property to **True** if you'd like notifications to delete on reboot.

C#

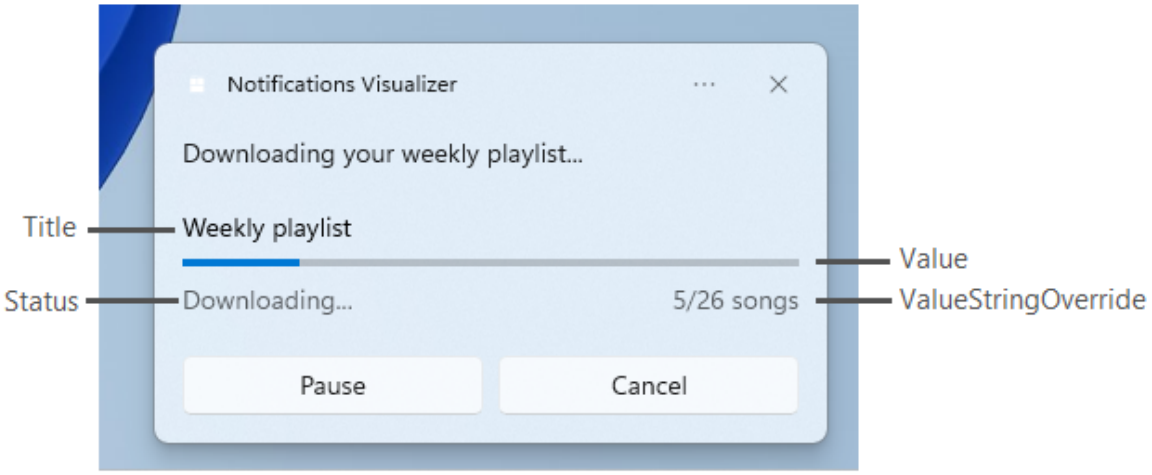
C#

```
class ToastWithAvatar
{
    public static bool SendToast()
    {

        var appNotification = new AppNotificationBuilder()
            .SetAppLogoOverride(new System.Uri("ms-
appx:///images/logo.png"), AppNotificationImageCrop.Circle)
            .AddText("Example ExpiresOnReboot notification")
            .AddText("This is an example message")
            .BuildNotification();
```

```
            appNotification.ExpiresOnReboot = true;
            AppNotificationManager.Default.Show(appNotification);

            return appNotification.Id != 0; // return true (indicating
success) if the toast was sent (if it has an Id)
        }
    }
```

Send and update a progress bar notification

You can display progress bar related updates in a notification:



Use the `AppNotificationProgressData` construct to update the progress bar notification.

C++

```
const winrt::hstring c_tag = L"weekly-playlist";
const winrt::hstring c_group = L"downloads";

// Send first Notification Progress Update
void SendUpdatableNotificationWithProgress()
{
    auto notification{ winrt::AppNotificationBuilder()
            .AddText(L"Downloading this week's new music...")
            .AddProgressBar(winrt::AppNotificationProgressBar()
                .BindTitle()
                .BindValue()
                .BindValueStringOverride()
                .BindStatus())
            .BuildNotification() }

    notification.Tag(c_tag);
    notification.Group(c_group);

    // Assign initial values for first notification progress UI
    winrt::AppNotificationProgressData data(1); // Sequence number
    data.Title(L"Weekly playlist"); // Binds to {progressTitle} in xml
```

```
payload
    data.Value(0.6); // Binds to {progressValue} in xml payload
    data.ValueStringOverride(L"15/26 songs"); // Binds to
{progressValueString} in xml payload
    data.Status(L"Downloading..."); // Binds to {progressStatus} in xml
payload

    notification.Progress(data);
    winrt::AppNotificationManager::Default().Show(notification);
}

// Send subsequent progress updates
winrt::Windows::Foundation::IAsyncAction UpdateProgressAsync()
{
    // Assign new values
    winrt::AppNotificationProgressData data(2 /* Sequence number */ );
    data.Title(L"Weekly playlist"); // Binds to {progressTitle} in xml
payload
    data.Value(0.7); // Binds to {progressValue} in xml payload
    data.ValueStringOverride(L"18/26 songs"); // Binds to
{progressValueString} in xml payload
    data.Status(L"Downloading..."); // Binds to {progressStatus} in xml
payload

    auto result = co_await
winrt::AppNotificationManager::Default().UpdateAsync(data, c_tag, c_group);
    if (result ==
winrt::AppNotificationProgressResult::AppNotificationNotFound)
    {
        // Progress Update failed since the previous notification update was
dismissed by the user! So account for this in your logic by stopping updates
or starting a new Progress Update flow.
    }
}
```

Resources

- Microsoft.Windows.AppNotifications API details
- Notifications code sample on GitHub
- App notifications spec on GitHub
- Toast content
- Notifications XML schema

Collaborate with us on GitHub

Windows developer feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Integrate with Windows

Article • 11/22/2024

This section of the documentation provides developer guidance for integrating with Windows.

Windows system components that support 3rd party integration

The following table lists the Windows system components that support integration for 3rd party developers.

Expand table

Feature	Description
Feed providers	Learn how to integrate into the Windows feeds experience.
Microsoft Copilot hardware key providers	Learn how to integrate into the Windows feeds experience.
Search providers	Learn how to integrate into the Windows Search experience.
Widget providers	Learn how to implement a Windows widget service provider to support your app.

Other Windows integration features

Expand table

Feature	Description
Smart App Control	Smart App Control is a new app execution control feature that combines Microsoft's app intelligence services and Windows' code integrity features to protect users from untrusted or potentially dangerous code.
Windows Share integration	The Windows Share Sheet is a system-provided UI that enables users to share content from your app with other apps.
Windows People experiences	Learn how third party apps can integrate with Windows People experiences using APIs to store their contacts.

Microsoft Copilot hardware key providers

Article • 03/31/2025

Starting with Windows Build 22621, apps can register to be included in the picker UI that allows users to select the app that is launched when the Microsoft Copilot hardware key or the Windows key + C is pressed.

> **Note**
>
> It is recommended that apps that register to be a Microsoft Copilot hardware key provider be implemented as single-window apps.

Microsoft Copilot hardware key app extension

An app must be packaged in order to register as a Microsoft Copilot hardware key provider. For information on app packaging, see An overview of Package Identity in Windows app. The app package manifest file, `Package.appxmanifest`, supports many different extensions and features for Windows apps. The app package manifest format is defined by a set of schemas that are documented in the Package manifest schema reference. Microsoft Copilot hardware key providers declare their registration information within the uap3:AppExtension. The **Name** attribute of the extension must be set to "com.microsoft.windows.copilotkeyprovider".

XML

```
<Package
...

  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
...>
    <Applications>
        <Application...>
            ...
            <Extensions>
                <uap3:Extension Category="windows.appExtension">
                    <uap3:AppExtension
Name="com.microsoft.windows.copilotkeyprovider"
                        Id="MyAppId"
                        DisplayName="App display name"
                        Description="App description"
                        PublicFolder="Public" />
                </uap3:Extension>
```

```
            </Extensions>
          ...
    </Application>
    </Applications>
    ...
</Package>
```

The following table *uap3:AppExtension* describes the attributes of the **uap3:AppExtension** element.

Expand table

Attribute	Description	Required
Id	The app-defined identifier for the app.	Yes
DisplayName	The app name displayed in the Windows Copilot hardware button picker UI.	Yes
Description	The app description displayed in the Windows Copilot hardware button picker UI.	Yes
PublicFolder	The folder that the instance declares as the location where a host can have read access to files through a broker.	Yes

Sign your Windows Copilot hardware key provider

Provider apps must be signed in order to be enabled as a target of the Microsoft Copilot hardware key. For information on packaging and signing your app, see Package a desktop or UWP app in Visual Studio.

Handle Microsoft Copilot hardware key state changes

Article • 03/28/2025

This article describes how apps can register to be activated and receive notifications when the Microsoft Copilot hardware key or Windows key + C is pressed, pressed and held, and released. This feature enables apps to perform different actions depending on which key state change is detected. For example, an app may perform normal activation when the key is single-pressed, but take a screenshot when the key is pressed and held. Or, an app may begin recording audio and show a status indicator that audio is being recorded when the key is pressed and held, and then stop recording audio when the key is released. The key must be pressed and held for at least 300 ms to move into the held state.

This feature extends the features of a basic Microsoft Copilot hardware key provider, which simply registers to be launched when the hardware key is pressed. For more information, see Microsoft Copilot hardware key providers.

The rest of this article will walk through creating a simple C# WinUI 3 app that responds to activation initiated by a single press or a press and hold and release of the Microsoft Copilot button.

Create a new project

In Visual Studio, create a new project. For this example, in the **Create a new project** dialog, set the language filter to C# and the project type to WinUI 3 and then select the "Blank App, Packaged (WinUI 3 in Desktop).

Add a property to track the Microsoft Copilot key pressed state

In this example, we will create a property called **State** that we will use to display the current activation state in the UI. In `MainWindow.xaml.cs`, inside the definition of **MainWindow** add the following code to create a string property that we can bind to in our XAML file.

```
C#
```

```
// MainWindow.xaml.cs
public event PropertyChangedEventHandler? PropertyChanged;

private void OnPropertyChanged([CallerMemberName] string propertyName =
"State")
{
    PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
}

public void SetState(string state)
{
    State = state;
}

private string _state;
public string State
{
    get => _state;
    set
    {
        if (_state != value)
        {
            _state = value;
            OnPropertyChanged();
        }
    }
}
```

Add a **TextBox** control to the UI to show the current activation state of the app. Replace the default **StackPanel** element in MainPage.xaml with the following code.

XAML

```
<!-- MainWindow.xaml -->
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
VerticalAlignment="Center">
    <TextBlock Name="KeyStateText" Text="{x:Bind State, Mode=OneWay}" />
</StackPanel>
```

Finally, update the **MainWindow** constructor to take an argument that will set the **State** property when the window is created.

C#

```
// MainWindow.xaml.cs
public MainWindow(string state)
{
    this.InitializeComponent();
```

```
    _state = state;
}
```

Register for URI activation

The system launches Microsoft Copilot hardware key providers using URI activation. Register a launch protocol by adding the uap:Protocol element to your app manifest. For more information about how to register as the default handler for a URI scheme, see Handle URI activation.

The following example shows the **uap:Extension** registering the URI scheme "myapp-copilothotkey".

XML

```
<!-- Package.appxmanifest -->
...
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
...

<Extensions>
  ...
  <uap:Extension Category="windows.protocol">
    <uap:Protocol Name="myapp-copilothotkey"> <!-- app-defined protocol name
-->
      <uap:DisplayName>SDK Sample URI Scheme</uap:DisplayName>
    </uap:Protocol>
  </uap:Extension>
  ...
```

Microsoft Copilot hardware key app extension

An app must be packaged in order to register as a Microsoft Copilot hardware key provider. For information on app packaging, see An overview of Package Identity in Windows app. Microsoft Copilot hardware key providers declare their registration information within the uap3:AppExtension. The **Name** attribute of the extension must be set to "com.microsoft.windows.copilotkeyprovider". To support the key state changes, apps must provide some additional entries to their **uap3:AppExtension** declaration.

Inside of the **uap3:AppExtension** element, add a uap3:Properties element with child elements **PressAndHoldStart** and **PressAndHoldStop**. The contents of these elements should be the URI of the protocol scheme registered in the manifest in the previous step. The query string arguments specify whether the URI is being launched because the user pressed and held the hot key or because the user released the hot key. The app

uses these query string values during app activation to determine the correct action to take.

XML

```
<!-- Package.appxmanifest -->

<Extensions>
  ...
  <uap3:Extension Category="windows.appExtension">
    <uap3:AppExtension Name="com.microsoft.windows.copilotkeyprovider"
      Id="MyAppId"
      DisplayName="App display name"
      Description="App description"
      PublicFolder="Public">
      <uap3:Properties>
        <PressAndHoldStart>myapp-copilothotkey:?
state=Down</PressAndHoldStart>
        <PressAndHoldStop>myapp-copilothotkey:?state=Up</PressAndHoldStop>
      </uap3:Properties>
    </ uap3:AppExtension>
  </uap3:Extension>
  ...
```

Handle URI activation

To detect whether the app was activated via URI activation, call AppInstance.GetActivatedEventArgs and check to see if the value of the AppActivationArguments.Kind property is Protocol. If the app was launched via protocol activation, check to see if the URI scheme is the same as the protocol name you specified in your app manifest. If all of these tests pass, then you know that your app was activated by the user pressing the Copilot hardware key. At this point you can parse the URI query string and get the *state* parameter, which will have the values you specified in the **PressAndHoldStart** and **PressAndHoldStop** elements in the app manifest.

C#

```
// App.xaml.cs

protected override void
OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
{
    var eventargs = AppInstance.GetCurrent().GetActivatedEventArgs();
    string state = "";
    if ((eventargs != null) && (eventargs.Kind ==
ExtendedActivationKind.Protocol))
    {
```

```
        var protocolArgs =
(Windows.ApplicationModel.Activation.ProtocolActivatedEventArgs)eventargs.Da
ta;
        WwwFormUrlDecoder decoderEntries = new
WwwFormUrlDecoder(protocolArgs.Uri.Query);
        state =
Uri.UnescapeDataString(decoderEntries.GetFirstValueByName("state"));
    }
    state = (state == "") ? "Launched" : state;

    m_window = new MainWindow(state);
    m_window.Activate();
}
```

> **Important**
>
> Note that, by default, WinUI 3 apps are multi-instanced, which means that a new instance will be launched whenever the Microsoft Copilot hot key is pressed or released. This may be the desired behavior for many providers, but if you would prefer, you can update you app to use a single instance. For more information, see Create a single-instanced WinUI app with C#.

Handle fast path invocation

In addition to URI activation, apps can register to support fast path invocation in which a running app receives messages about Copilot hardware app through window messages. For a currently-running app, this invocation method is faster than URI activation and will provide a better user experience, since the app can begin listening for speech more quickly after the key is pressed and held.

Update the app manifest file to support fast path invocation

To add support for fast path invocation, update the "com.microsoft.windows.copilotkeyprovider" extension to add the *MessageWParam* attribute to the **SingleTap**, **PressAndHoldStart**, and **PressAndHoldStop** elements. Each *MessageWParam* value must be a unique 32-bit integer, but the values used are chosen by the app. This example uses values of 0, 1, and 2, respectively. These values will be used later in the example when they are passed in the *wParam* parameter of a Windows message to determine the current pressed state of the Windows Copilot hardware key.

```
XML
```

```
<!-- Package.appxmanifest -->

<uap3:Extension Category="windows.appExtension">
  <uap3:AppExtension Name="com.microsoft.windows.copilotkeyprovider"
    Id="MyAppId"
    DisplayName="App display name"
    Description="App description"
    PublicFolder="Public">
    <uap3:Properties>
      <SingleTap FastPathValue="0"/>
      <PressAndHoldStart MessageWParam="1">myapp-copilothotkey://?
state=Down</PressAndHoldStart>
      <PressAndHoldStop MessageWParam="2">myapp-copilothotkey://?
state=Up</PressAndHoldStop>
    </uap3:Properties>
  </uap3:AppExtension>
</uap3:Extension>
```

Access win32 APIs for window registration

Fast path activation is enabled by setting a property on the IPropertyStore associated with one of the app's windows. To do this requires access to some native Win32 APIs. This walkthrough will use the CsWin32 library, which automates the generation of C# bindings and is available as a NuGet package.

In Visual Studio, in **Solution Explorer**, right-click on your project file and select **Manage NuGet packages...**. On the **Browse** tab of the NuGet package manager, search for "cswin32" and select the "Microsoft.Windows.CsWin32" package and click *Install*.

After the package is installed, add a new text file in your project directory and name it "NativeMethods.txt". The CsWin32 tool will look in this file for a list of the Win32 APIs that it will generate bindings for. Put the following API names in "NativeMethods.txt".

`SUBCLASSPROC`

`SHGetPropertyStoreForWindow`

`IPropertyStore`

`SetWindowSubclass`

`DefSubclassProc`

Register the window for Microsoft Copilot fastpath invocation

Next we will update the **MainWindow** class to register the window to receive fastpath invocations from the Copilot hardware key.

First, call GetWindowHandle to get an HWND handle to the **MainWindow**. Call SHGetPropertyStoreForWindow to get the **IPropertyStore** for the window. Create a new PROPERTYKEY and set the *fmtid* member to the GUID for Windows Copilot fastpath activation. Set the value of the property to an app-defined value, that will be passed back to the app from the system when the hardware key state changes. The app-defined value is the windows message ID which must be in the WM_APP range. For more information, see WM_APP. Call SetValue and then call Commit to commit the change to the property store.

Finally, create a SUBCLASSPROC callback that will be called when the hardware key state changes. **WindowSubClass** is the callback implementation that will be shown in the next step. Call SetWindowSubclass to register the callback.

C#

```
private HWND hWndMain;
private Windows.Win32.UI.Shell.SUBCLASSPROC SubClassDelegate;
public const int WM_COPILOT = 0x8000 + 0x0001;

public MainWindow(string state)
{
    this.InitializeComponent();

    hWndMain = (HWND)WinRT.Interop.WindowNative.GetWindowHandle(this);
    Microsoft.UI.Windowing.AppWindow appWindow = AppWindow;


    var propertyStoreGUID = new Guid("886D8EEB-8CF2-4446-8D02-
CDBA1DBDCF99");
    var hr =
PInvoke.SHGetPropertyStoreForWindow((HWND)this.AppWindow.Id.Value, in
propertyStoreGUID, out var propertyStore);
    var key = new PROPERTYKEY();
    var copilotFastpathGUID = new Guid("38652BCA-4329-4E74-86F9-
39CF29345EEA");
    key.fmtid = copilotFastpathGUID;
    key.pid = 0x00000002;
    var value = new PROPVARIANT();
    value.Anonymous.Anonymous.vt = VARENUM.VT_UINT;
    value.Anonymous.decVal = WM_COPILOT;
    ((IPropertyStore)propertyStore).SetValue(in key, in value);
    ((IPropertyStore)propertyStore).Commit();

    SubClassDelegate = new
Windows.Win32.UI.Shell.SUBCLASSPROC(WindowSubClass);
    bool bRet = PInvoke.SetWindowSubclass((HWND)appWindow.Id.Value,
SubClassDelegate, 0, 0);
```

```
    _state = state;
}
```

Implement the window subclass callback

The last step in this example is implementing the window subclass callback that will be called whenever the app is running and the state of the Windows Copilot hardware key changes. In this example, we check that the window message is the **WM_COPILOT** value that we specified when setting the property store value in the previous step. Then we check the value of the *wParam* argument to see which of the values we specified with the **MessageWParam** attributes in the app manifest has been passed in. **SetState** is called to update the UI with the current state.

C#

```
private LRESULT WindowSubClass(HWND hWnd, uint uMsg, WPARAM wParam, LPARAM
lParam, nuint uIdSubclass, nuint dwRefData)
{
    switch (uMsg)
    {
        case WM_COPILOT:
        {
            switch (wParam.Value)
            {
                case 0:
                    SetState("SingleTap");
                    break;
                case 1:
                    SetState("PressAndHold START");
                    break;
                case 2:
                    SetState("PressAndHold END");
                    break;
            }
        }
        break;

    }
    return PInvoke.DefSubclassProc((HWND)hWnd, uMsg, wParam, lParam);

}
```

Sign your Windows Copilot hardware key provider

Provider apps must be signed in order to be enabled as a target of the Microsoft Copilot hardware key. For information on packaging and signing your app, see Package a desktop or UWP app in Visual Studio.

Cross Device People API

Article • 06/18/2024

Windows is an ideal platform for third party apps to integrate their top people contacts. This integration enables users to interact with the personas for various people experiences. Windows now provides third party WinUI 3, UWP, and other apps with package identity with APIs to store all their contacts.

> **ⓘ Important**
>
> Some information relates to a prerelease product which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.

> **ⓘ Note**
>
> Apps can try this feature as soon as it is released to the Windows Insider Program (Beta Channel) in Windows Update settings (see Get started with the Windows Insider Program for more information).

Once your apps store their contacts in Windows, users will be able to see these contact suggestions on the **Share** panel in Windows to seamlessly share with their top contacts. See How to share files in File Explorer on Windows for more information about the **Share** panel.

Creating a UserDataAccount for People Contract

Start by creating a user data account. Third party apps are required to create a UserDataAccount with `UserDisplayName` as `"com.microsoft.peoplecontract"`.

C#

```
UserDataAccountStore udas =
    await
UserDataAccountManager.RequestStoreAsync(UserDataAccountStoreAccessType.AppA
ccountsReadWrite);
UserDataAccount uda = await
udas.CreateAccountAsync("com.microsoft.peoplecontract");
```

Next, add `"com.microsoft.windows.system"` to the list of ExplictReadAccessPackageFamilyNames for the account. This will provide restricted access of third party contacts to Windows experiences.

C#

```
uda.ExplictReadAccessPackageFamilyNames.Add("com.microsoft.windows.system");
await uda.SaveAsync();
```

Storing contacts

The first step in storing contacts is to create a contact list. To do this, third party apps must create the new contact list for a `UserDataAccount` in the Windows ContactStore. Apps can choose to keep the default `OtherAppReadAccess` access type for the contact list, while setting it to `None` will prevent other apps from having any access to these contacts. See the ContactListOtherAppReadAccess enum for the full list of available access types.

C#

```
ContactStore store = await
ContactManager.RequestStoreAsync(ContactStoreAccessType.AppContactsReadWrite
);
this.contactList = await store.CreateContactListAsync(contactListsName,
uda.Id);
contactList.OtherAppReadAccess = ContactListOtherAppReadAccess.None;
await contactList.SaveAsync();
```

While storing a contact, third party apps must include all relevant information required for Windows experiences to power a Contact.

The following fields are required when storing a contact:

- `FirstName`
- `RemoteId`
- `DisplayPicture`

The following fields are optional:

- `LastName`
- `Phones`
- `Emails`

This code snippet demonstrates how to store a contact:

```
C#
```

```
foreach (var appContact in AppContacts)
{
  var cont = new Contact
  {
    FirstName = appContact.FirstName,
    LastName = appContact.LastName,
    RemoteId = appContact.Id,
    SourceDisplayPicture = RandomAccessStreamReference.CreateFromUri(new
Uri(appContact.ProfilePicPath)),
    Phones = { new ContactPhone { Number = appContact.Phone } }
  };

  await this.contactList.SaveContactAsync(cont);
}
```

> **Note**
>
> The `DisplayName` for the **Contact** is constructed using `FirstName` and `LastName`. If the last name is not provided, `DisplayName` will be identical to the string provided for the first name.

Storing ranks for contacts

You can create an annotation list for the `UserDataAccount` to store ranks for your contacts. Apps can store ranks for their top contacts by adding annotations to the contacts. These annotations are stored as part of an annotation list in the contact store.

```
C#
```

```
ContactAnnotationStore annotationStore = await

ContactManager.RequestAnnotationStoreAsync(ContactAnnotationStoreAccessType.
AppAnnotationsReadWrite);
this.contactAnnotationList = await
annotationStore.CreateAnnotationListAsync(uda.Id);
```

You can store ranks for your top contacts by using the annotations on contacts. Ranks are stored as part of ProviderProperties on a contact annotation. Along with rank, apps must set the SupportedOperations on a contact annotation as `Share`.

```
C#
```

```
foreach (var appContact in topAppContacts)
{
  Contact contact = await
list.GetContactFromRemoteIdAsync(topAppContact.RemoteID);
  var annotation = new ContactAnnotation
  {
    ContactId = contact.Id,
    SupportedOperations = ContactAnnotationOperations.Share
  };
  annotation.ProviderProperties.Add("Rank", rank);
  await annotationsLst.TrySaveAnnotationAsync(annotation);
}
```

Updating contact ranks

It's at the discretion of apps when to update the ranks of the contacts stored in Windows. Windows recommends that ranked lists be updated regularly to provide the best user experience. Whenever you need to update a ranked list, you'll need to follow several steps.

1. Delete the ContactAnnotationList.

   Once the app has an updated list of top contacts, the annotation list can be deleted and a new annotation list with updated annotations for their top contacts can be created.

   C#

   ```
   await this.contactAnnotationList.DeleteAsync();
   ```

2. Create a new `ContactAnnotationList`. Follow the steps in the Storing ranks for contacts section to create a new annotation list and store ranks for your top contacts.

See also

- UserDataAccount
- ContactStore
- Windows.ApplicationModel.Contacts Namespace

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Smart App Control

Article • 12/15/2022

Smart App Control is a new app execution control feature that combines Microsoft's app intelligence services and Windows' code integrity features to protect users from untrusted or potentially dangerous code. Smart App Control selectively allows apps and binaries to run only if they're likely to be safe. Microsoft's app intelligence services provide safety predictions for many popular apps. If the app intelligence service is unable to make a prediction, then Smart App Control will still allow an app to run if it is signed with a certificate issued by a certificate authority (CA) within the Trusted Root Program.

Malware, Potentially Unwanted Apps (PUA), and unknown, unsigned code are blocked by default.

Smart App Control requirements

Smart App Control is designed to protect a device for its entire lifetime. As such, it can only be enabled on a clean install of a version of Windows that contains the Smart App Control feature. Additionally, Smart App Control is only enabled in certain regions. We hope to roll out additional regions soon.

- Windows 11, version 22572 or higher,
- A clean Windows install,

  > **Note**
  >
  > **Resetting your device** counts as a clean Windows install.

- North America or Europe

Smart App Control stages

Smart App Control can be running in either *evaluation mode* or *enforcement mode*.

In *evaluation mode*, Smart App Control runs in the background, observing activity on the device. During this time, Smart App Control is evaluating whether the device is a good fit for the protection it offers based the variety of apps installed and used on the device.

In *enforcement mode*, Smart App Control is actively protecting your device. Apps cannot be run unless they are recognized by Microsoft's app intelligence services, or signed

with a trusted certificate.

Frequently Asked Questions

How can I tell if Smart App Control is installed my device?

Go to **Settings** > **Windows Security** > **App and Browser Control**. If Smart App Control is installed on your system, you will see a section called **Smart App Control**

How can I tell if Smart App Control running in evaluation or enforcement mode?

Go to **Settings** > **Windows Security** > **App and Browser Control**.

- If **On** is selected, Smart App Control is running in enforcement mode.
- If **Evaluation** is selected, Smart App Control is running in evaluation mode.
- If **Off** is selected, Smart App Control is not running on this device.

Will I be notified when Smart App Control enters enforcement mode?

Yes. You will receive a Toast Notification when Smart App Control enters enforcement mode.

What files will Smart App Control block while running in enforcement mode?

Smart App Control allows apps and binaries to run only if they're likely to be safe. Smart App Control will block apps and binary files identified as unsafe by Microsoft's app intelligence services unless those files are code signed with a certificate issued by a certificate authority (CA) within the Trusted Root Program.

Note that some older Microsoft binaries are considered unsafe because attackers can potentially use them to gain unauthorized access. For a complete list of these files, please see Application Control for Windows.

Further information

- What is Smart App Control?

Sign your app for Smart App Control compliance

Article • 05/09/2024

Code signing is a cryptographic operation that can be performed on an app in order to verify its contents and publisher. Smart App Control considers apps signed with a trusted digital certificate to be safe, and allows them to run on a protected computer.

There are several ways to sign your app.

Obtain a code signing certificate from a trusted provider

Code can be signed with any certificate, but Smart App Control only considers certificates issued by trusted providers. For information about how to obtain a code signing certificate from a trusted provider, see Manage code signing certificates

Sign your app with Trusted Signing

Trusted Signing (formerly Azure Code Signing) is the preferred way to sign your app. Trusted Signing is currently in public preview.

Sign your app with signtool.exe

Signtool.exe is an app included with Visual Studio that can sign apps with a digital certificate. For instructions on how to sign your app with signtool.exe, see How to sign an app package using SignTool

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Test your app's signature with Smart App Control

Article • 12/15/2022

Before distributing your signed app to users, you should test your app's signature against Smart App Control. Because Smart App Control evaluates binaries as they're loaded, be sure to test all code paths and features of your app. This includes testing all of your app's install and uninstall binaries, all of your app's features, and all integrations with other apps that might load your binaries (for example, Office add-ins). You can test Smart App Control using audit policies, which will create log entries without actually blocking your app from executing, or test directly against Smart App Control's enforcement mode.

Configure Smart App Control for Testing

You can configure Smart App Control in the Windows Settings app, or by manually editing the Windows Registry.

Configure Smart App Control using Windows Settings

Go to **Settings** > **Privacy & Security** > **Windows Security** > **App and Browser Control** > **Smart App Control settings**.

> **Note**
>
> Configuring Smart App Control to **Off** or **On** (enforcement) is a one-way operation. This means you cannot change modes using Windows Settings unless the current setting is **Evaluation**. For testing purposes, you can force Smart App Control into another setting **using the registry**.

If Smart App Control is in Evaluation mode, Smart App Control will evaluate your app's signature, but will not block your app if its signature is invalid. In this mode, you can use Audit Policies to view Smart App Control's output, including errors encountered while checking your app's signature.

Select **On** to put Smart App Control in enforcement mode. In this mode, Smart App Control will prevent your app from running if its signature is invalid.

Configure Smart App Control using the Registry

> **ⓘ Important**
>
> Smart App Control can be manually configured via the Registry **for testing purposes only.** Editing Smart App Control settings in this way could compromise the protection it provides.

Configuring Smart App Control using the Windows Registry allows you to force any desired enforcement mode, even if you cannot select that mode using Windows Settings. To configure Smart App Control:

1. Open a command prompt with administrator privileges and execute the following commands:

   ```PowerShell
   manage-bde -protectors c: -disable -rebootcount 2
   "C:\Program Files\Windows Defender\MpCmdRun.exe" -RemoveDefinitions -DynamicSignatures
   ```

   > **ⓘ Note**
   >
   > You may have to update the second command if your system drive is not C:.

2. Reboot into the boot menu by launching Settings and selecting **Recovery** > **Recovery Options** > **Advanced Startup** > **Restart now**.

3. From the advanced boot menu, select **Troubleshoot** > **Advanced** > **Command Prompt**. A recovery command prompt will open.

   > **ⓘ Note**
   >
   > The recovery command prompt opens the recovery drive X: by default. This does not indicate your system drive has changed. Your system drive is still associated with its usual drive letter (usually C:).

4. Execute the following commands:

   > **ⓘ Note**

In the following commands, replace {VALUE} with the value of the mode you want to set.

Value	Mode
0	Off
1	On (Enforcement)
2	Evaluation

ps

```
reg load HKLM\sac c:\windows\system32\config\system
reg add hklm\sac\controlset001\control\ci\policy /v
VerifiedAndReputablePolicyState /t REG_DWORD /d {VALUE} /f
reg add hklm\sac\controlset001\control\ci\protected /v
VerifiedAndReputablePolicyStateMinValueSeen /t REG_DWORD /d {VALUE} /f
reg unload hklm\sac

reg load HKLM\sac2 C:\windows\system32\config\SOFTWARE
reg add "hklm\sac2\Microsoft\Windows Defender" /v SacLearningModeSwitch
/t REG_DWORD /d 0
reg unload hklm\sac2
```

5. Restart the computer.

Verify Smart App Control's current mode

You can verify Smart App Control's current mode by opening a command prompt and executing the following command:

`citool.exe -lp`

Smart App Control is in evaluation mode if the value of Friendly Name is `VerifiedAndReputableDesktopEvaluation` and the value of Is Currently Enforced is `true`.

Smart App Control is in enforcement mode if the value of Friendly Name is `VerifiedAndReputableDesktop` and the value of Is Currently Enforced is `true`.

Configure Smart App Control's audit policy

The default Windows Defender Application Control (WDAC) policy used by Smart App Control in evaluation mode does not log audit events in the CodeIntegrity Operational log. This is to reduce the size of the log on typical consumer devices shipping with Smart App Control in evaluation mode.

For the purposes of evaluating applications against Smart App Control, a developer or system administrator may want to enable audit logs in evaluation mode to see what files would be blocked if the system were in enforcement mode.

> **Note**
>
> Audit policies only apply when Smart App Control is running in Evaluation mode. In Enforcement mode, Smart App Control will log events by default.

A zip file containing two sample policies below can be downloaded here.

> **Note**
>
> You can also create your own policies. See **Windows Defender Application Control (WDAC) example base policies** and **Create WDAC policy for lightly managed devices** for more information.

Smart App Control audit policy (SmartAppControlAudit.bin)

This is the standard Smart App Control policy, with audit logs enabled in evaluation mode. All binaries and scripts allowed by signature and cloud reputation will pass the policy, just as they would if enforcement mode was enabled. Applications and binaries that would be blocked would log an audit event.

> **Note**
>
> This policy only works with Smart App Control in evaluation mode. It is still possible for the Smart App Control evaluation model to turn evaluation mode off with this policy is applied, so we recommend testing with one of the other methods below.

When this policy is applied, the output for `citool.exe -lp` will show `VerifiedAndReputableDesktopEvaluationAudit` as the policy name.

Apply the Smart App Control audit policy

First, ensure SAC is in evaluation mode.

Take ownership of the evaluation mode policy file `C:\WINDOWS\System32\CodeIntegrity\CiPolicies\Active\{1283AC0F-FFF1-49AE-ADA1-8A933130CAD6}.cip` using takeown.exe. If you are unable to use takeown, then you can manually take ownership using the following steps:

> **ⓘ Important**
>
> We strongly recommend using takeown, if possible.

1. Right click the file in explorer and select "Properties."
2. Go to the Security tab, and choose Advanced at the bottom.
3. Click "Change" in the dialog.
4. In the popup dialog, enter your user information (e.g. `<PC name>\<username>`) and click OK.
5. Click OK in the Advanced Security Settings dialog and confirm.
6. Reopen the file properties Security tab and click "Edit."
7. Under Administrators, choose all the checkboxes and click OK, and confirm again in the popup dialog.

Now that you have ownership of the policy file, rename it to `{1283AC0F-FFF1-49AE-ADA1-8A933130CAD6}.cip.old`. Rename the audit policy file you want to apply to `{1283AC0F-FFF1-49AE-ADA1-8A933130CAD6}.cip`, and copy it to the policy directory.

Run `citool.exe -r` from an admin command prompt to refresh the policy.

Smart App Control audit policy without ISG (SmartAppControlAuditNoISG.bin)

This is the recommended policy for testing your own apps as a developer.

This policy checks binaries and scripts against Smart App Control in evaluation mode, without checking the Intelligent Security Graph, meaning that only apps that are properly signed by a trusted certificate will be allowed without audit events. Because reputation may not be available for newly published binaries, and can change over time, ensuring that all your binaries are correctly signed is the best way to make sure users don't encounter issues using your app. This is also the requirement when publishing

through the Windows Store, where a signature from a cert obtained from a trusted Certificate Authority is required.

This policy can be applied even when Smart App Control is set to Off. When this policy is applied, the output for `citool.exe -lp` will show `VerifiedAndReputableDesktopEvaluationAuditNoISG` as the policy name.

Apply the Smart App Control Audit policy without ISG

This policy is for testing applications in evaluation mode against the signing requirement of Smart App Control exclusively, and will not allow any app binaries based on cloud intelligence from the Intelligent Security Graph.

Ensure that Smart App Control is in evaluation mode or off

Run `mountvol S: /S` from an admin command prompt

copy SmartAppControlAuditNoISG.bin to `S:\efi\microsoft\boot\cipolicies\active\{5283AC0F-FFF1-49AE-ADA1-8A933130CAD6}.cip` .

Run `citool.exe -r` from admin command line to refresh the policy

Checking Event Logs

Smart App Control logs any executable that was (or would have been) blocked into the Code Integrity Event Logs. You can find those logs by opening the Event Viewer, and then browsing to **Application and Services Logs** > **Microsoft** > **Windows** > **CodeIntegrity** > **Operational**.

Smart App Control logs evaluation mode events with event ID 3076, and enforcement mode events with event ID 3077. For more information about Smart App Control and Microsoft Defender event logging, please see Review event logs and error codes to troubleshoot issues with Microsoft Defender Antivirus.

Integrate with Windows Search

Article • 05/10/2025

This section of the documentation provides developer guidance for integrating with Windows Search.

Windows system components that support 3rd party integration

The following table lists the Windows Search integration features for 3rd party developers.

Expand table

Feature	Description
Web search providers	Learn how to integrate a web search provider with Windows. Only available the European Economic Area (EEA).
Cloud files provider integration with Windows Search	Learn about cloud file provider integration with Windows Search.

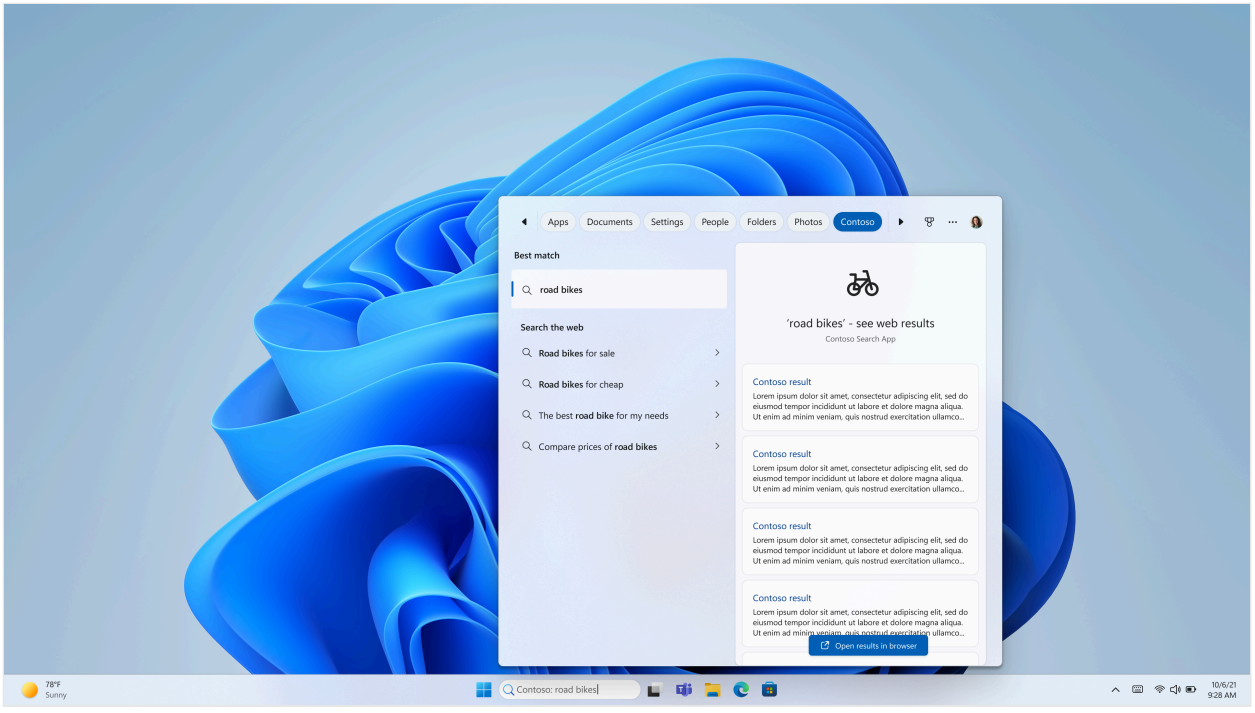
Windows Search providers

Article • 05/07/2024

> **Note**
>
> **Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.**

> **Important**
>
> The feature described in this topic is available in preview builds of Windows starting with build 22631.2787 for Windows 11 and build 19045.3758 for Windows 10, although newer builds are recommended because they contain stability improvements. For information on preview builds of Windows, see Windows 10 Insider Preview.

Windows Search currently uses the Web Search from Microsoft Bing app to return web content and search results. In the European Economic Area (EEA), you can enable installed apps that implement a web search provider to return web content and search results in Windows Search through Settings.



Search providers integrate with the Search experience by creating an MSIX package with a package manifest file that provides the required information for the OS to register the

search provider. Users can add a search provider to Windows by installing the associated app package and can remove the search provider through the **Add or remove programs** page in Windows Settings app.

For development and testing, when Developer Mode is enabled and the search provider app has been sideloaded on the device, it will appear in the list of available search providers. For more information, see Developer Mode features and debugging.

Once the search provider is registered with the OS, user queries are passed to the HTTP endpoint specified by the provider in their package manifest using a standardized query string. The endpoint returns suggested results in a JSON document. With each suggested URL in the response document, the search provider includes the preview endpoint URL, which returns an HTML document that is displayed in the preview pane in the search results UI.

This article provides guidance for creating a search provider app package and details about the protocols for implementing search provider HTTP endpoints.

Create a search extensibility app package

Search providers register with the OS by providing an MSIX package that contains required information about the provider, such as the name of the search provider and the HTTP endpoints for suggestions and previews.

Search provider app extension

The app package manifest file supports many different extensions and features for Windows apps. The app package manifest format is defined by a set of schemas that are documented in the Package manifest schema reference. Search providers declare their registration information within the uap3:AppExtension. The **Name** attribute of the extension must be set to "com.microsoft.windows.websearchprovider".

Search providers should include the uap3:Properties as the child of **uap3:AppExtension**. The package manifest schema does not enforce the structure of the **uap3:Properties** element other than requiring well-formed XML. The rest of this section describes the XML format that the OS expects in order to successfully register a search provider.

XML

```
<uap3:Extension Category="windows.appExtension">
  <uap3:AppExtension Name="com.microsoft.windows.websearchprovider"
DisplayName="SearchExampleApp" Id="ContosoSearchApp" PublicFolder="Public">
    <uap3:Properties>
```

```
    <!-- Search provider registration content goes here -->
    </uap3:Properties>
  </uap3:AppExtension>
</uap3:Extension>
```

Element hierarchy

uap3:Properties

EndPoint

Protocol

Endpoint

The URL of the HTTPS endpoint to which the OS will send search query requests.

Protocol

The protocol schema that will be used when launching the provided web search results. If the specified protocol is not registered by an app on the OS, then the default browser will be launched for search results. For more information on registering protocol schemas, see uap:Protocol.

DynamicContentEndpoint

The URL of the HTTPS endpoint to which the OS will send a request for the gleam icon to be displayed in the search box. For more information see Implement a gleam icon endpoint. This feature is supported starting with Windows 10 build 19045.4233 and Windows 11 build 22621.3371.

Example package manifest file

The following is an example `appmanifest.xml` package manifest file for registering a Windows Search provider.

XML

```
<!-- appxmanifest.xml -->

  <uap3:Extension Category="windows.appExtension">
      <uap3:AppExtension Name="com.microsoft.windows.websearchprovider"
DisplayName="CustomSearch" Id="CustomSearchApp" PublicFolder="Public">
```

```
            <uap3:Properties>
                <Endpoint>https://customsearchendpoint</Endpoint>
                <Protocol>customsearch</Protocol>

<DynamicContentEndpoint>https://sub.contoso.com/dynamic</DynamicContentEndpo
int>
            </uap3:Properties>
        </uap3:AppExtension>
    </uap3:Extension>
    <uap:Extension Category="windows.protocol">
        <uap:Protocol Name="customsearch"/>
    </uap:Extension>
```

Implement a Windows Search provider suggestion endpoint

Search providers must expose and register an HTTPS endpoint that is called by the OS when a user types into the Windows Search box. This endpoint should return a JSON-formatted string containing the search suggestions for the provided user query. Content must be delivered over HTTPS. Search integration does not support content delivered over HTTP.

Suggestion HTTPS request format

The HTTPS request to the suggestion endpoint uses the following format.

`https://contoso.com?setlang=en-US&cc=US&qry=`

The query string parameters passed to the suggestion endpoint are the following.

Expand table

Parameter	Description
setlang	The locale associated with the query.
cc	The country code associated with query.
qry	The query provided by the user. If the parameter has no value, i.e. appears in the query string as `qry=`, then the user query is empty. Search providers can still provide suggestions and preview pages in response to an empty query. **NOTE** The OS does not perform any sanitization of query strings. Search providers can implement their own sanitization when the query is received.

Suggestion HTTPS response headers

Search provider must include the following headers in the response from the suggestion HTTPS endpoint.

- Access-Control-Allow-Origin: https://www.bing.com
- Access-Control-Allow-Credentials: true
- Access-Control-Allow-Methods: GET
- Content-Type: application/json; charset=utf-8
- Content-Length: [Must be the exact length of response]

Suggestion response JSON format

The search provider HTTPS endpoint for suggestions must return a JSON document with the following format. The key names must match the format exactly.

Expand table

Key	Description
Suggestions	Contains a list of JSON objects with key `Attributes` representing the suggestions associated with the user query.
Attributes	Contains the attributes of a suggestion.
url	The URL for the search suggestion on the provider web site.
query	The user query associated with the search suggestion.
previewPaneUrl	The URL of the preview endpoint from which an HTML preview of the suggestion can be retrieved.
Text	The text description of the suggestion.

JSON

```
{"Suggestions":
   [{"Attributes":
     {"url":"https://www.contoso.com/search?
q=projection+matrix","query":"projection
matrix","previewPaneUrl":"http://www.contoso.com/preview"}
,"Text":"projection matrix"},
    {"Attributes":
     {"url":"https://www.contoso.com/search?
q=rotation+matrix","query":"rotation
matrix","previewPaneUrl":"http://www.contoso.com/preview"} ,"Text":"rotation
matrix"}
```

```
    ]
}
```

Implement a Windows Search provider preview endpoint

Search providers return the URL of an HTTPS endpoint that provides an HTML preview of the page associated with each suggestion in the search results. The preview endpoint response must return the HTML code for a functioning page.

Preview HTTPS request format

The HTTPS request to the preview endpoint uses the following format.

`https://contoso.com?Darkschemeovr=1`

The query string parameters passed to the suggestion endpoint are the following.

Expand table

Parameter	Description
Darkschemeovr	Specifies if the calling Windows system has dark theme enabled. The value is 1 if dark theme is enabled and 0 if the dark theme is disabled.

Preview HTTPS response headers

- Access-Control-Allow-Origin: https://www.bing.com
- Access-Control-Allow-Credentials: true
- Access-Control-Allow-Methods: GET
- Content-Type: text/html; charset=utf-8
- Content-Length: [Must be the exact length of preview html]

OPTIONS request and Cross-Origin Resource Sharing (CORS)

Search providers must support the OPTIONS request method and respond to this request with HTTP OK. If the search provider endpoint is using CORS, the Windows search client will send out a HTTP OPTIONS request before each GET request.

Implement a gleam icon endpoint

Search providers can optionally provide light and dark mode gleam icons that are displayed in the search bar when the search provider is currently enabled. When the **DynamicContentEndpoint** element is provided in the app manifest, a request will be sent to the specified URL and the search provider responds with a json file in the format defined below that includes the URLs of the icon image files and other metadata. The gleam icon request will be sent periodically while the search provider is the most recent provider active in Windows Search. The cadence for this request is every 6 hours. A request will also be sent upon each Search launch and on device unlock.

Gleam icon HTTPS request format

The HTTPS request to the gleam icon endpoint uses the following format.

```
https://www.contoso.com/Gleam?cc=FR&setlang=en-us&dateTime=3%2F29%2F2024%2C%208%3A33%3A56%20PM&deviceOs=windows10&schemaversion=1.0.0
```

The query string parameters passed to the suggestion endpoint are the following.

Expand table

Parameter	Description
setlang	The locale associated with the query.
cc	The country code associated with query.
dateTime	The current date and time from client device, url-encoded.
deviceOs	The OS of the client device. The value of this parameter can be "Windows10" or "Windows11". On Windows 10, the gleam icon size is 30x60. On Windows 11, the gleam icon size is 20x36
schemaversion	The gleam schema version.

Gleam icon response JSON format

The search provider HTTPS endpoint for gleam icons must return a JSON document with the following format. The key names must match the format exactly. The current schema version is 1.0.0.

Expand table

Key	Description
schemaVersion	The gleam schema version. This should match the *schemaVersion* query string parameter in the request.
telemetryId	A unique identifier for the gleam icon. If the value in the response is the same as the value for the current gleam icon, the OS will not update the icon.
expirationTime	The expiration time for the gleam icon. Must be a time in the future.
content	The content section of the response.
taskbarSearchBox	Contains settings for the search box.
gleam	Contains settings for the gleam icon.
altText	Alternate text for the gleam icon.
dimensionEnum	The value "30x60" if the request was sent from a Windows 10 device. The value "20x36" if the request was sent from a Windows 11 device.
iconUrl	Contains the URLs for the light and dark gleam icon image files.
light	The URL for the light gleam icon image file.
dark	The URL for the dark gleam icon image file.

JSON

```
{
  "schemaVersion":"1.0.0",
  "telemetryId":"<unique gleam Id>",
  "expirationTime":"2025-12-09T20:37:13Z",
  "content": {
    "taskbarSearchBox": {
      "gleam":{
        "altText": "<alt text of the gleam>",
        "dimensionEnum": "(30x60 for Windows 10, 20x36 for Windows 11)",
        "iconUrl": {
          "light":"<3p's light gleam url>",
          "dark": "<3p's dark gleam url>"
        }
      }
    }
  }
}
```

Gleam icon response validation

The response must specify both the light asset URL and the dark asset URL. The domains for the icon image URLs must use HTTPS and the subdomain must match the subdomain specified in the **DynamicContentEndpoint** element in the app manifest file.

The image files must be in SVG format and the maximum file size is 300 kB. The gleam needs to be within a 240x120px frame inside the SVG.

If an empty payload is received, that will clear the active gleam icon and no gleam will be displayed.

Related articles

Cloud files provider integration with Windows Search

Article • 05/10/2025

Windows Search integrates with Cloud Sync Engines. When searching for files, Windows Search will call into all registered cloud files providers. Cloud files providers must implement IStorageProviderSearchHandlerFactory in order to be integrated into the Windows Search experience. For more information about implementing a Cloud Sync Engine, see Cloud Sync Engines.

Integrate Share options in your Windows app: A comprehensive guide

Article • 11/11/2024

The Windows Share Sheet is a system-provided UI that enables users to share content from your app with other Windows apps. The Share Sheet is available in the Windows shell and is accessible from any app that supports sharing. It provides a consistent and familiar experience for users, and it's a great way to increase the discoverability of your app.

Integrating share options in your Windows app can significantly enhance user experience by allowing seamless content sharing across different applications. Whether you're developing a new app or updating an existing one, this guide will provide you with the necessary steps and code samples to implement the Windows Share feature effectively.

In this comprehensive guide, you'll learn how to add the share feature to your packaged or unpackaged Windows app, implement file sharing methods, and show your app on the share sheet. You can leverage share options in a .NET, Windows C++, or PWA app, and this guide will help you get started. By following these steps, you can ensure that your app leverages the full potential of the Windows Share feature, making it more interactive and user-friendly.

In this section

Expand table

Topic	Description
Integrate packaged apps with Windows Share	Discover how to integrate packaged apps with the Windows Share Sheet.
Integrate Progressive Web Apps (PWAs) with Windows Share	Discover how to integrate a Progressive Web App (PWA) with the Windows Share Sheet.
Integrate unpackaged apps with Windows Share	Discover how to integrate unpackaged apps with the Windows Share Sheet.
Share data - UWP apps	Learn how to share data between UWP apps.

See also

- Communication - Windows apps
- Windows App SDK deployment overview
- Create your first WinUI 3 project
- Migrate from UWP to the Windows App SDK
- Advantages and Disadvantages of packaging an application - Deployment overview
- Identity, Registration and Activation of Non-packaged Win32 Apps
- Share Contract Implementation for WinAppSDK App
- Share Contract Implementation for Apps Packaged with External Location

Integrate packaged apps with Windows Share

Article • 05/08/2024

This article explains how to integrate packaged apps with the **Windows Share** feature. Apps that are packaged with MSIX have package identity and are ready to register as a **Share Target**. The Share feature allows users to share content from one app to another. A packaged app will register as a Share Target in order to receive and handle shared files within the app.

What is Share Target?

Share Target is a feature that was introduced in Windows 8, and it allows an app to receive data from another app. Share Target works like a Clipboard but with dynamic content.

For the default share target registration to work with Win32 apps, the app needs to have a package identity and also handle the share arguments as ShareTargetActivatedEventArgs, which is a live object from the source app. It isn't a static memory content that is sent to the target app.

> ⓘ **Note**
>
> In a C++ app, use the GetCurrentPackageFullName API to check if the running app has package identity. The API returns the `APPMODEL_ERROR_NO_PACKAGE` error code if it isn't running with package identity.

Prerequisites

To support `ShareTargetActivatedEventArgs`, the app must target Windows 10, version 2004 (build 10.0.19041.0) or later. This is the minimum target version for the feature.

Register as a Share Target

There are two steps required to implement the Share contract in your app.

Add a share target extension to appxmanifest

In Visual Studio's Solution Explorer, open the `package.appxmanifest` file of the Packaging project in your solution and add the share target extension.

XML

```
<Extensions>
      <uap:Extension
          Category="windows.shareTarget">
        <uap:ShareTarget>
          <uap:SupportedFileTypes>
            <uap:SupportsAnyFileType />
          </uap:SupportedFileTypes>
          <uap:DataFormat>Bitmap</uap:DataFormat>
        </uap:ShareTarget>
      </uap:Extension>
</Extensions>
```

Add the supported data format that is supported by your application to the `DataFormat` configuration. In this case, the app supports sharing images, so the `DataFormat` is set to `Bitmap`.

Fetch Share Event arguments

Starting in Windows 10, version 1809, packaged apps can call the AppInstance.GetActivatedEventArgs method to retrieve certain kinds of app activation info during startup. For example, you can call this method to get information about app activation; whether it was triggered by opening a file, clicking an interactive toast, or using a registered protocol.

However, ShareTargetActivatedEventArgs activation info is supported only on Windows 10, version 2004, and later. So, the application should target to devices with this specific minimum version.

To see a Windows App SDK implementation, see the `OnLaunched` method in the Share Target sample app.

For other packaged apps, in the `Main` method of the application, check for `AppInstance.GetActivatedEventArgs`.

C#

```
public static void Main(string[] cmdArgs)
{
    ...
    if (isRunningWithIdentity())
    {
```

```
        var activationArgs = AppInstance.GetActivatedEventArgs();
        if (activationArgs != null)
        {
            switch (activationArgs.Kind)
            {
                case ActivationKind.Launch:
                    HandleLaunch(activationArgs as
LaunchActivatedEventArgs);
                    break;
                case ActivationKind.ToastNotification:
                    HandleToastNotification(activationArgs as
ToastNotificationActivatedEventArgs);
                    break;
                case ActivationKind.ShareTarget:
                    HandleShareAsync(activationArgs as
ShareTargetActivatedEventArgs);
                    break;
                default:
                    HandleLaunch(null);
                    break;
            }
        }
    }
}
```

See the Photo Store Demo app for a complete implementation.

Handle shared files

The following code snippet shows how to handle shared files in a packaged app. The code snippet is part of the `HandleShareAsync` method that is called when the app is activated as a Share Target in the previous example.

C#

```
static async void HandleShareAsync(ShareTargetActivatedEventArgs args)
{
    ShareOperation shareOperation = args.ShareOperation;
    shareOperation.ReportStarted();

    if (shareOperation.Data.Contains(

Windows.ApplicationModel.DataTransfer.StandardDataFormats.StorageItems))
    {
        try
        {
            IReadOnlyList<IStorageItem> items = await
shareOperation.Data.GetStorageItemsAsync();
            var file = (IStorageFile)items[0];
            string path = file.Path;
            var image = new ImageFile(path);
```

```
                image.AddToCache();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
        shareOperation.ReportCompleted();

        // app launch code
    }
```

See also

- Windows App SDK deployment overview
- Create your first WinUI 3 project
- Migrate from UWP to the Windows App SDK
- Advantages and Disadvantages of packaging an application - Deployment overview
- Identity, Registration and Activation of Non-packaged Win32 Apps
- Share Contract Implementation for WinAppSDK App
- Share Contract Implementation for Apps Packaged with External Location
- Get activation info for packaged apps
- ShareTarget schema reference

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Integrate Progressive Web Apps with Windows Share

Article • 05/08/2024

This article explains how to integrate a Progressive Web App (PWA) with the **Windows Share** feature. This feature allows users to share content from one Windows app to another. The PWA registers as a **Share Target** to receive and handle shared files within the app.

What is Share Target?

Share Target is a feature that was introduced in Windows 8, and it allows an app to receive data from another app. Share Target works like a Clipboard but with dynamic content.

Share content from your PWA

To share content, a PWA generates content (such as text, links, or files) and hands off the shared content to the operating system. The operating system lets the user decide which app they want to use to receive that content. PWAs can use the Web Share API to trigger displaying the Share Sheet in Windows. The Web Share API is supported in Microsoft Edge and other Chromium-based browsers.

See Sharing content in the Microsoft Edge documentation for a complete example of how to share content from a PWA.

Receive shared files in your PWA

To receive content, a PWA acts as a content target. The PWA must be registered with the operating system as a content-sharing target.

The `share_target` member must contain the necessary information for the system to pass the shared content to your app. Consider the following sample manifest `share_target` configuration:

JSON

```
"share_target": {
  "action": "./share_target_path/?custom_param=foo",
  "method": "POST",
```

```
  "enctype": "multipart/form-data",
  "params": {
    "files": [
      {
        "name": "mapped_files",
        "accept": ["image/jpeg"]
      }
    ]
  }
},
```

When your app is selected by the user as the target for shared content, the PWA is launched. A `GET` HTTP request is made to the URL specified by the `action` property. The shared data is passed as the `title`, `text`, and `url` query parameters. The following request is made: `/handle-shared-content/?title=shared title&text=shared text&url=shared url`.

The following example illustrates how to register the scoped service worker:

JavaScript

```
navigator.serviceWorker.register('scoped-service-worker.js',
                { scope: "./share_target_path/" })
```

The service worker handles the share data as desired, and then either fulfills the request, or it can redirect the request back out of the custom path. The following example illustrates how to redirect the request back out of the custom path:

JavaScript

```
self.addEventListener('fetch', (event) => {
    event.respondWith((async () => {
        // Read the shared data here, then
        // Redirect back out of the share_target_path to the actual site
        return
Response.redirect(event.request.url.replace("share_target_path/", ""));
    })());
    return;
});
```

See the Receiving shared content example in the Microsoft Edge documentation for more information.

Performance considerations

If the fetch handler is added solely for share support, potential latency issues may arise as the requests get interrupted by the service worker. To address this issue, consider setting the `share_target` as a pseudo sub-path and registering a service worker specifically scoped to that path. This approach enables the use of a *fetch* handler for the share target without registering the same *fetch* handler for other calls.

Sample PWA app

The PWA logo printer sample app on GitHub demonstrates how to integrate a PWA with the Windows Share Sheet. The app allows users to print the PWA logo to a printer. The app uses the Windows Share Sheet to share the logo with other apps.

See also

- How to receive shared files
- PWA logo printer sample app
- Share content with other apps
- Web Share API

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Share data

Article • 10/18/2023

This article explains how to support the Share contract in a desktop or a Universal Windows Platform (UWP) app. The Share contract is an easy way to quickly share data—such as text, links, photos, and videos—between apps. For example, a user might want to share a webpage with their friends using a social networking app, or save a link in a notes app to refer to later.

> **Note**
>
> The code examples in this article are from UWP apps. Desktop apps should use the **IDataTransferManagerInterop**. For more info, and code examples, see **Display WinRT UI objects that depend on CoreWindow**. Also see the **WPF Sharing content source app sample**.

Set up an event handler

Add a **DataRequested** event handler to be called whenever a user invokes share. This can occur either when the user taps a control in your app (such as a button or app bar command) or automatically in a specific scenario (if the user finishes a level and gets a high score, for example).

C#

```
DataTransferManager dataTransferManager =
DataTransferManager.GetForCurrentView();
dataTransferManager.DataRequested += DataTransferManager_DataRequested;
```

When a **DataRequested** event occurs, your app receives a **DataRequest** object. This contains a **DataPackage** that you can use to provide the content that the user wants to share. You must provide a title and data to share. A description is optional, but recommended.

C#

```
DataRequest request = args.Request;
```

Choose data

You can share various types of data, including:

- Plain text
- Uniform Resource Identifiers (URIs)
- HTML
- Formatted text
- Bitmaps
- Files
- Custom developer-defined data

The DataPackage object can contain one or more of these formats, in any combination. The following example demonstrates sharing text.

C#

```
request.Data.SetText("Hello world!");
```

Set properties

When you package data for sharing, you can supply a variety of properties that provide additional information about the content being shared. These properties help target apps improve the user experience. For example, a description helps when the user is sharing content with more than one app. Adding a thumbnail when sharing an image or a link to a web page provides a visual reference to the user. For more information, see DataPackagePropertySet.

> ⚠ **Warning**
>
> All properties except the title are optional. The title property is mandatory and must be set.

C#

```
request.Data.Properties.Title = "Share Example";
request.Data.Properties.Description = "A demonstration on how to share";
```

Launch the share UI

A UI for sharing is provided by the system. To launch it, call the ShowShareUI method.

C#

```
DataTransferManager.ShowShareUI();
```

Handle errors

In most cases, sharing content is a straightforward process. However, there's always a chance that something unexpected could happen. For example, the app might require the user to select content for sharing but the user didn't select any. To handle these situations, use the **FailWithDisplayText** method, which will display a message to the user if something goes wrong.

Delay share with delegates

Sometimes, it might not make sense to prepare the data that the user wants to share right away. For example, if your app supports sending a large image file in several different possible formats, it's inefficient to create all those images before the user makes their selection.

To solve this problem, a **DataPackage** can contain a delegate — a function that is called when the receiving app requests data. We recommend using a delegate any time that the data a user wants to share is resource-intensive.

C#

```
async void OnDeferredImageRequestedHandler(DataProviderRequest request)
{
    // Provide updated bitmap data using delayed rendering
    if (this.imageStream != null)
    {
        DataProviderDeferral deferral = request.GetDeferral();
        InMemoryRandomAccessStream inMemoryStream = new
InMemoryRandomAccessStream();

        // Decode the image.
        BitmapDecoder imageDecoder = await
BitmapDecoder.CreateAsync(this.imageStream);

        // Re-encode the image at 50% width and height.
        BitmapEncoder imageEncoder = await
BitmapEncoder.CreateForTranscodingAsync(inMemoryStream, imageDecoder);
        imageEncoder.BitmapTransform.ScaledWidth = (uint)
(imageDecoder.OrientedPixelWidth * 0.5);
        imageEncoder.BitmapTransform.ScaledHeight = (uint)
(imageDecoder.OrientedPixelHeight * 0.5);
        await imageEncoder.FlushAsync();
```

```
request.SetData(RandomAccessStreamReference.CreateFromStream(inMemoryStream)
);
        deferral.Complete();
    }
}
```

See also

- Display WinRT UI objects that depend on CoreWindow
- App-to-app communication
- Receive data
- DataPackage
- DataPackagePropertySet
- DataRequest
- DataRequested
- FailWithDisplayText
- ShowShareUi

Integrate unpackaged apps with Windows Share

Article • 05/08/2024

This article explains how to integrate unpackaged apps with the **Windows Share** feature. The Share feature allows users to share content from one Windows app to another. An unpackaged app must be provided with package identity so it can register as a **Share Target**. Once registered, the app can receive and process shared files.

How to onboard an unpackaged app as a Share Target:

- Provide the app with package identity
- Implement the Share contract

Provide unpackaged apps with package identity

An app can get package identity in two ways:

- Make a new MSIX installation package (preferred method) **OR**
- Make apps packaged with external location compatible with the current installer. This is only recommended for apps that have an existing installer and which can't switch to MSIX installation.

Make a new MSIX installation package

It's recommended to package the app with MSIX using the **Windows Application Packaging Project** template in Visual Studio. This will include all the binaries in the MSIX package and provide a clean and trusted install experience.

Things to note before packaging desktop apps: Prepare to package a desktop application (MSIX).

Follow the steps in Set up your desktop application for MSIX packaging in Visual Studio to generate a package for your existing app's project.

> **Note**

When creating the packaging project, select **Windows 10, version 2004 (10.0; Build 19041)** or later as the **Minimum version**.

When that is completed, you will create the package by following Package a desktop or UWP app in Visual Studio.

Make packaging with external location compatible with the current installer

The second way to give your app package identity is to add a package with external location to your application and register it with your existing installer. The package with external location is an empty MSIX package that contains the .appxmanifest having identity, share target registration, and visual assets. The app's binaries are still managed by app's existing installer. When registering the package, you need to provide the app's install location in the API. It is important to keep the identity in the MSIX package manifest and the Win32 app manifest in sync with the certificate used for signing the app.

Steps to grant package identity to an unpackaged app

Documentation on how to create a package with external location is available here, including information on templates to use: Grant package identity by packaging with external location.

The full sample app is available on GitHub: SparsePackages (Packaged with External Location) .

Register as a Share Target

Once the app has package identity, the next step is to implement the Share contract. The Share contract allows your app to receive data from another app.

You can follow the same steps in the Register as a Share Target section of the documentation for packaged apps to integrate with Share Sheet.

Walkthrough of the sample PhotoStore app

In this walkthrough on package identity, registration & share activation for unpackaged Win32 Applications, you will learn how to grant package identity to an unpackaged Win32 application by creating a package with external location. Once the app has

package identity, it can register and handle activation as a Share Target. You will take the following steps using the PhotoStoreDemo sample:

- Generate the `AppxManifest.xml` file
- Create a package
- Sign the package
- Register the package
- Handle app activation

Start by creating the `AppxManifest.xml` file, which includes necessary properties like `<AllowExternalContent>`, identity, and capabilities & share target extension. Make sure the `Publisher`, `PackageName` & `ApplicationId` values in the `AppxManifest.xml` file match the values in the `PhotoStoreDemo.exe.manifest` file. The `Publisher` value should also match the value in the certificate used to sign the package. Add visual assets as required and referenced in `AppxManifest.xml`. In Visual Studio, you can use the **Visual Assets** node when editing `package.manifest` in the Application Packaging project to generate required the visual assets.

This is a sample `AppxManifest.xml` snippet with external content allowed:

XML

```
<Identity Name="PhotoStoreDemo" ProcessorArchitecture="neutral"
Publisher="CN=YourPubNameHere" Version="1.0.0.0" />
  <Properties>
    <DisplayName>PhotoStoreDemo</DisplayName>
    <PublisherDisplayName>Sparse Package</PublisherDisplayName>
    <Logo>Assets\storelogo.png</Logo>
    <uap10:AllowExternalContent>true</uap10:AllowExternalContent>
  </Properties>
  <Resources>
    <Resource Language="en-us" />
  </Resources>
  <Dependencies>
    <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.19041.0"
MaxVersionTested="10.0.19041.0" />
  </Dependencies>
  <Capabilities>
    <rescap:Capability Name="runFullTrust" />
    <rescap:Capability Name="unvirtualizedResources"/>
  </Capabilities>
  <Applications>
    <Application Id="PhotoStoreDemo" Executable="PhotoStoreDemo.exe"
uap10:TrustLevel="mediumIL" uap10:RuntimeBehavior="win32App">
      <uap:VisualElements AppListEntry="none" DisplayName="PhotoStoreDemo"
Description="PhotoStoreDemo" BackgroundColor="transparent"
Square150x150Logo="Assets\Square150x150Logo.png"
Square44x44Logo="Assets\Square44x44Logo.png">
```

```
        <uap:DefaultTile Wide310x150Logo="Assets\Wide310x150Logo.png"
Square310x310Logo="Assets\LargeTile.png"
Square71x71Logo="Assets\SmallTile.png"></uap:DefaultTile>
        <uap:SplashScreen Image="Assets\SplashScreen.png" />
      </uap:VisualElements>
      <Extensions>
        <uap:Extension Category="windows.shareTarget">
          <uap:ShareTarget Description="Send to PhotoStoreDemo">
            <uap:SupportedFileTypes>
              <uap:FileType>.jpg</uap:FileType>
              <uap:FileType>.png</uap:FileType>
              <uap:FileType>.gif</uap:FileType>
            </uap:SupportedFileTypes>
            <uap:DataFormat>StorageItems</uap:DataFormat>
            <uap:DataFormat>Bitmap</uap:DataFormat>
          </uap:ShareTarget>
        </uap:Extension>
        ...
```

This is a sample Application.exe.manifest file:

XML

```
<?xml version="1.0" encoding="utf-8"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
  <assemblyIdentity version="1.0.0.0" name="PhotoStoreDemo.app"/>
  <msix xmlns="urn:schemas-microsoft-com:msix.v1"
          publisher="CN=YourPubNameHere"
          packageName="PhotoStoreDemo"
          applicationId="PhotoStoreDemo"
        />
</assembly>
```

Next, to create the package with external location, use the `MakeAppx.exe` tool with the `/nv` command to create a package containing the `AppxManifest.xml` file.

Example:

Console

```
MakeAppx.exe pack /d <Path to directory with AppxManifest.xml> /p <Output
Path>\mypackage.msix /nv
```

Note

A package with external location contains a package manifest, but no other app binaries and content. The manifest of a package with external location can reference files outside the package in a predetermined external location.

Sign your package with a trusted certificate using `SignTool.exe`.

Example:

Console

```
SignTool.exe sign /fd SHA256 /a /f <path to cert>  /p <cert key> <Path to
Package>
```

The cert used for signing the package should be installed in a trusted location on the machine.

On first run of the application, register the package with Windows. When an app has its own installer, it should also contain the signed MSIX as the payload and should place it to a specified location (for example, the app's install location). This location must be known to the app at runtime because the app will need the absolute path of MSIX to register it. Place the assets and `resources.pri` in the app's install location as well.

The following code is an example of unpackaged execution of the app's Main method:

C#

```
[STAThread]
public static void Main(string[] cmdArgs)
{
    //if app isn't running with identity, register its package with external
identity
    if (!ExecutionMode.IsRunningWithIdentity())
    {
        //TODO - update the value of externalLocation to match the output
location of your VS Build binaries and the value of
        //externalPkgPath to match the path to your signed package with
external identity (.msix).
        //Note that these values cannot be relative paths and must be
complete paths
        string externalLocation = Environment.CurrentDirectory;
        string externalPkgPath = externalLocation +
@"\PhotoStoreDemo.package.msix";

        //Attempt registration
        bool bPackageRegistered = false;
        //bPackageRegistered =
registerPackageWithExternalLocation(externalLocation, externalPkgPath);
        if (bPackageRegistered)
```

```
        {
            //Registration succeeded, restart the app to run with identity

System.Diagnostics.Process.Start(Application.ResourceAssembly.Location,
arguments: cmdArgs?.ToString());
        }
        else //Registration failed, run without identity
        {
            Debug.WriteLine("Package Registration failed, running WITHOUT
Identity");
            SingleInstanceManager wrapper = new SingleInstanceManager();
            wrapper.Run(cmdArgs);
        }
    }
    ...
```

This example shows how to register the MSIX on first run of the app:

C#

```
[STAThread]
public static void Main(string[] cmdArgs)
{
    //If app isn't running with identity, register its package with external
identity
    if (!ExecutionMode.IsRunningWithIdentity())
    {
        //TODO - update the value of externalLocation to match the output
location of your VS Build binaries and the value of
        //externalPkgPath to match the path to your signed package with
external identity (.msix).
        //Note that these values cannot be relative paths and must be
complete paths
        string externalLocation = Environment.CurrentDirectory;
        string externalPkgPath = externalLocation +
@"\PhotoStoreDemo.package.msix";

        //Attempt registration
        if (registerPackageWithExternalLocation(externalLocation,
externalPkgPath))
        {
            //Registration succeeded, restart the app to run with identity

System.Diagnostics.Process.Start(Application.ResourceAssembly.Location,
arguments: cmdArgs?.ToString());
        }
        else //Registration failed, run without identity
        {
            Debug.WriteLine("Package Registration failed, running WITHOUT
Identity");
            SingleInstanceManager wrapper = new SingleInstanceManager();
            wrapper.Run(cmdArgs);
        }
```

```
    }
    ...
```

Finally, handle the app's activation:

C#

```
[STAThread]
public static void Main(string[] cmdArgs)
{
    //if app isn't running with identity, register its sparse package
    if (!ExecutionMode.IsRunningWithIdentity())
    {
        ...
    }
    else //App is registered and running with identity, handle launch and
activation
    {
        //Handle Sparse Package based activation e.g Share target activation
or clicking on a Tile
        // Launching the .exe directly will have activationArgs == null
        var activationArgs = AppInstance.GetActivatedEventArgs();
        if (activationArgs != null)
        {
            switch (activationArgs.Kind)
            {
                case ActivationKind.Launch:
                    HandleLaunch(activationArgs as
LaunchActivatedEventArgs);
                    break;
                case ActivationKind.ToastNotification:
                    HandleToastNotification(activationArgs as
ToastNotificationActivatedEventArgs);
                    break;
                case ActivationKind.ShareTarget: // Handle the activation as
a share target
                    HandleShareAsync(activationArgs as
ShareTargetActivatedEventArgs);
                    break;
                default:
                    HandleLaunch(null);
                    break;
            }

        }
        //This is a direct exe based launch e.g. double click app .exe or
desktop shortcut pointing to .exe
        else
        {
            SingleInstanceManager singleInstanceManager = new
SingleInstanceManager();
            singleInstanceManager.Run(cmdArgs);
```

```
        }
    }
```

Windows Share with package identity demo

The following video demonstrates how an unpackaged app can be a share target after being granted package identity and registering as a share target: https://learn-video.azurefd.net/vod/player?id=49784732-8213-43b2-b1fe-003aa218432e&locale=en-us&embedUrl=%2Fwindows%2Fapps%2Fdevelop%2Fwindows-integration%2Fintegrate-sharesheet-unpackaged

See also

- Windows App SDK deployment overview
- Create your first WinUI 3 project
- Migrate from UWP to the Windows App SDK
- Advantages and Disadvantages of packaging an application - Deployment overview
- Identity, Registration and Activation of Non-packaged Win32 Apps
- Share Contract Implementation for WinAppSDK App
- Share Contract Implementation for Apps Packaged with External Location

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Feed providers

Article • 01/14/2025

> **Note**
>
> **Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.**

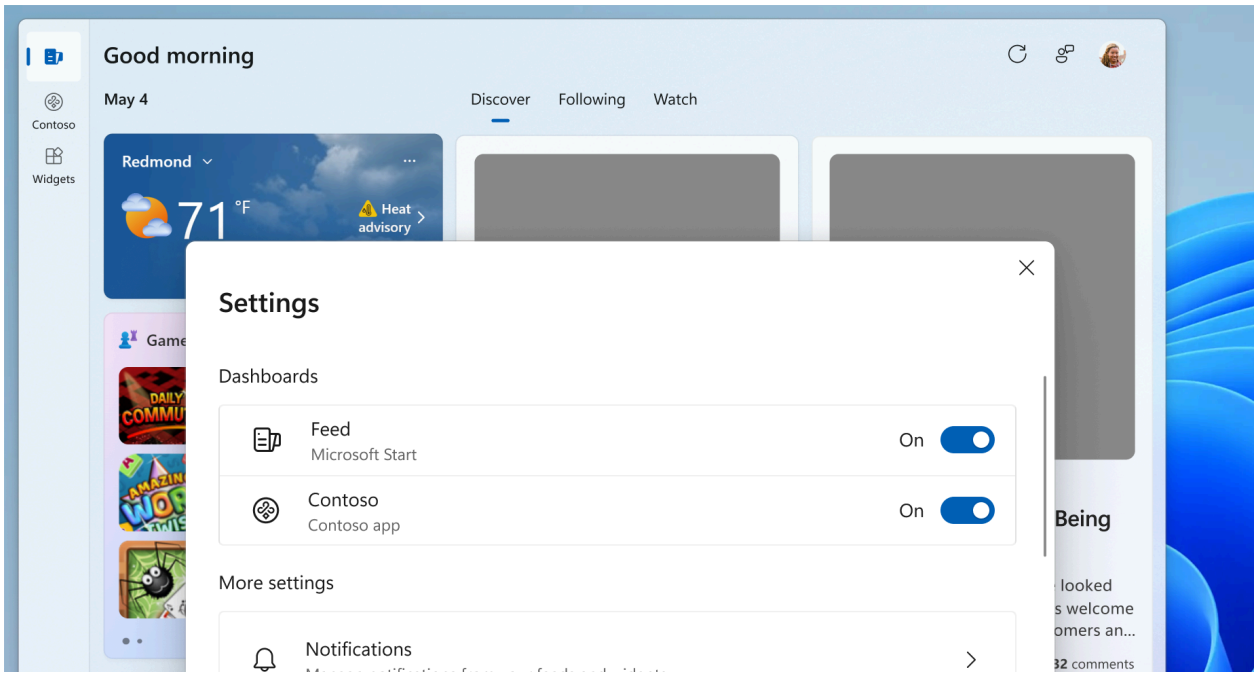
The feed providers feature in the Windows App SDK is a new integration point for third-party applications. It enables these applications to register their content feeds to be directly available within the Windows Widgets Board, enhancing the user experience by providing quick access to a variety of content directly from the desktop.

This article introduces the concept of feed providers and provides a high-level explanation the feature.

For detailed guidance on how to implement a feed provider, see these articles below:

- Implement a feed provider in a C# Windows App
- Implement a feed provider in a win32 app (C++/WinRT)
- Feed provider package manifest XML format

Feeds in the Widgets Board helps users stay on top of what matters, enabling them to easily discover useful information and empowering them to act on it. Feed providers enable users to see content from multiple apps and services at the same time. Users can access content from various apps directly on their Widgets Board without the need to open individual apps, ensuring they have the latest information at their fingertips. Users also have the control to enable or disable feeds from the Widgets Board settings, tailoring the content to their preferences.



Getting started with feed providers

The following lists the high-level steps for developing a feed provider:

1. **Register feeds** - Register your app's feeds in the app manifest. Once registered and detected, these feeds become directly available in the Widgets Board.
2. **Implement feed experience** - Develop the feed experience as a web component that will be rendered within an i-frame on the Widgets Board. Feeds will appear as pivots above the **Feeds** section of the Widgets Board.
3. **Provide personalization controls (optional)** - Each feed provider can define a personalization control dialog that allows users to customize their feed experience according to their preferences.

Limitations and considerations

- The feed providers feature is in preview.
- This feature is available only to users in the European Economic Area (EEA). In the EEA, installed apps that implement a feed provider can provide content feed in the Widgets Board.
- The feature requires using the latest Windows App SDK for app development.
- Specific technical and design guidelines must be adhered to for proper feed integration.

Next steps

For detailed guidance on how to implement a feed provider, see these articles below:

- Implement a feed provider in a C# Windows App
- Implement a feed provider in a win32 app (C++/WinRT)
- Feed provider package manifest XML format

Implement a feed provider in a C# Windows App

Article • 12/15/2023

> ⓘ Note
>
> **Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.**

This article walks you through creating a simple feed provider that registers a feed content URI and implements the **IFeedProvider** interface. The methods of this interface are invoked by the Widgets Board to request custom query string parameters, typically to support authentication scenarios. Feed providers can support a single feed or multiple feeds.

To implement a feed provider using C++/WinRT, see Implement a feed provider in a win32 app (C++/WinRT).

Prerequisites

- Your device must have developer mode enabled. For more information see Enable your device for development.
- Visual Studio 2022 or later with the **Universal Windows Platform development** workload.

Create a new C# console app

In Visual Studio, create a new project. In the **Create a new project** dialog, set the language filter to "C#" and the platform filter to Windows, then select the Console App project template. Name the new project "ExampleFeedProvider". For this walkthrough, make sure that **Place solution and project in the same directory** is unchecked. When prompted, set the target .NET version to 6.0.

When the project loads, in **Solution Explorer** right-click the project name and select **Properties**. On the **General** page, scroll down to **Target OS** and select "Windows". Under **Target OS Version**, select version 10.022631.2787 or later.

Note that this walkthrough uses a console app that displays the console window when the feed is activated to enable easy debugging. When you are ready to publish your feed provider app, you can convert the console application to a Windows application by following the steps in Convert your console app to a Windows app.

Add references to the Windows App SDK NuGet package

This sample uses the latest stable Windows App SDK NuGet package. In **Solution Explorer**, right-click **Dependencies** and select **Manage NuGet packages...**. In the NuGet package manager, select the **Browse** tab and search for "Microsoft.WindowsAppSDK". Select the latest stable version in the **Version** drop-down and then click **Install**.

Add a FeedProvider class to handle feed operations

In Visual Studio, right-click the `ExampleFeedProvider` project in **Solution Explorer** and select **Add->Class**. In the **Add class** dialog, name the class "FeedProvider" and click **Add**. In the generated FeedProvider.cs file, update the class definition to indicate that it implements the **IFeedProvider** interface.

Create a CLSID that will be used to identify your feed provider for COM activation. Generate a GUID in Visual Studio by going to **Tools->Create GUID**. Save this GUID in a text file to be used later when packaging the feed provider app. Replace the GUID in the annotations for the **FeedProvider** class shown in the following example.

C#

```
// FeedProvider.cs
using Microsoft.Windows.Widgets.Feeds.Providers;
...
[ComVisible(true)]
[ComDefaultInterface(typeof(IFeedProvider))]
[Guid("xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")]
public sealed class FeedProvider : IFeedProvider
```

Implement the IFeedProvider methods

In the next few sections, we'll implement the methods of the **IFeedProvider** interface.

> **Note**
>
> Objects passed into the callback methods of the **IFeedProvider** interface are only guaranteed to be valid within the callback. You should not store references to these objects because their behavior outside of the context of the callback is undefined.

OnFeedProviderEnabled

The **OnFeedProviderEnabled** method is invoked when a feed associated with the provider is created by the Widgets Board host. In the implementation of this method, generate a query string with the parameters that will be passed to the URL that provides the feed content, including any necessary authentication tokens. Create an instance of **CustomQueryParametersUpdateOptions**, passing in the **FeedProviderDefinitionId** from the event args that identifies the feed that has been enabled and the query string. Get the default **FeedManager** and call **SetCustomQueryParameters** to register the query string parameters with the Widgets Board.

C#

```
// FeedProvider.cs

public void OnFeedProviderEnabled(FeedProviderEnabledArgs args)
{
    Console.WriteLine($"{args.FeedProviderDefinitionId} feed provider was
enabled.");
    var updateOptions = new
CustomQueryParametersUpdateOptions(args.FeedProviderDefinitionId,
"param1&param2");
    FeedManager.GetDefault().SetCustomQueryParameters(updateOptions);
}
```

OnFeedProviderDisabled

OnFeedProviderDisabled is called when the Widgets Board when all of the feeds for this provider have been disabled. Feed providers are not required to perform any actions in response to these this method call. The method invocation can be used for telemetry purposes or to update the query string parameters or revoke authentication tokens, if needed. If the app only supports a single feed provider or if all feed providers supported by the app have been disabled, then the app can exit in response to this callback.

C#

```
// FeedProvider.cs
public void OnFeedProviderDisabled(FeedProviderDisabledArgs args)
{
    Console.WriteLine($"{args.FeedProviderDefinitionId} feed provider was
disabled.");
}
```

OnFeedEnabled, OnFeedDisabled

OnFeedEnabled and **OnFeedDisabled** are invoked by the Widgets Board when a feed is enabled or disabled. Feed providers are not required to perform any actions in response to these method calls. The method invocation can be used for telemetry purposes or to update the query string parameters or revoke authentication tokens, if needed.

C#

```
// FeedProvider.cs
public void OnFeedEnabled(FeedEnabledArgs args)
{
    Console.WriteLine($"{args.FeedDefinitionId} feed was enabled.");
}
```

C#

```
// FeedProvider.cs
public void OnFeedDisabled(FeedDisabledArgs args)
{
    Console.WriteLine($"{args.FeedDefinitionId} feed was disabled.");
}
```

OnCustomQueryParametersRequested

OnCustomQueryParametersRequested is raised when the Widgets Board determines that the custom query parameters associated with the feed provider need to be refreshed. For example, this method may be raised if the operation to fetch feed content from the remote web service fails. The **FeedProviderDefinitionId** property of the **CustomQueryParametersRequestedArgs** passed into this method specifies the feed for which query string params are being requested. The provider should regenerate the query string and pass it back to the Widgets Board by calling **SetCustomQueryParameters**.

C#

```
// FeedProvider.cs

public void
OnCustomQueryParametersRequested(CustomQueryParametersRequestedArgs args)
{
    Console.WriteLine($"CustomQueryParamaters were requested for
{args.FeedProviderDefinitionId}.");
    var updateOptions = new
CustomQueryParametersUpdateOptions(args.FeedProviderDefinitionId,
"param1&param2");
    FeedManager.GetDefault().SetCustomQueryParameters(updateOptions);
}
```

Implement a class factory that will instantiate FeedProvider on request

In order for the feed host to communicate with our feed provider, we must call CoRegisterClassObject. This function requires us to create an implementation of the IClassFactory that will create a class object for our **FeedProvider** class. We will implement our class factory in a self-contained helper class.

In Visual Studio, right-click the `ExampleFeedProvider` project in **Solution Explorer** and select **Add->Class**. In the **Add class** dialog, name the class "FactoryHelper" and click **Add**.

Replace the contents of the FactoryHelper.cs file with the following code. This code defines the **IClassFactory** interface and implements its two methods, CreateInstance and LockServer. This code is typical boilerplate for implementing a class factory and is not specific to the functionality of a feed provider except that we indicate that the class object being created implements the **IFeedProvider** interface.

C#

```
// FactoryHelper.cs
using Microsoft.Windows.Widgets.Feeds.Providers;
using System.Runtime.InteropServices;
using WinRT;

namespace ExampleFeedProvider
{
    namespace Com
    {
        static class Guids
        {
            public const string IClassFactory = "00000001-0000-0000-C000-
000000000046";
```

```
            public const string IUnknown = "00000000-0000-0000-C000-
000000000046";
        }

        [ComImport(), InterfaceType(ComInterfaceType.InterfaceIsIUnknown),
Guid(Guids.IClassFactory)]
        internal interface IClassFactory
        {
            [PreserveSig]
            int CreateInstance(IntPtr pUnkOuter, ref Guid riid, out IntPtr
ppvObject);
            [PreserveSig]
            int LockServer(bool fLock);
        }

        static class ClassObject
        {
            public static void Register(Guid clsid, object pUnk, out uint
cookie)
            {
                [DllImport("ole32.dll")]
                static extern int CoRegisterClassObject(
                    [MarshalAs(UnmanagedType.LPStruct)] Guid rclsid,
                    [MarshalAs(UnmanagedType.IUnknown)] object pUnk,
                    uint dwClsContext,
                    uint flags,
                    out uint lpdwRegister);

                int result = CoRegisterClassObject(clsid, pUnk, 0x4, 0x1,
out cookie);
                if (result != 0)
                {
                    Marshal.ThrowExceptionForHR(result);
                }
            }

            public static int Revoke(uint cookie)
            {
                [DllImport("ole32.dll")]
                static extern int CoRevokeClassObject(uint dwRegister);

                return CoRevokeClassObject(cookie);
            }
        }
    }

    internal class FeedProviderFactory<T> : Com.IClassFactory
            where T : IFeedProvider, new()
    {
        public int CreateInstance(IntPtr pUnkOuter, ref Guid riid, out
IntPtr ppvObject)
        {
            ppvObject = IntPtr.Zero;

            if (pUnkOuter != IntPtr.Zero)
```

```
            {
                Marshal.ThrowExceptionForHR(CLASS_E_NOAGGREGATION);
            }

            if (riid == typeof(T).GUID || riid ==
Guid.Parse(Com.Guids.IUnknown))
            {
                // Create the instance of the .NET object
                ppvObject =
MarshalInspectable<IFeedProvider>.FromManaged(new T());
            }
            else
            {
                // The object that ppvObject points to does not support the
                // interface identified by riid.
                Marshal.ThrowExceptionForHR(E_NOINTERFACE);
            }

            return 0;
        }

        int Com.IClassFactory.LockServer(bool fLock)
        {
            return 0;
        }

        private const int CLASS_E_NOAGGREGATION = -2147221232;
        private const int E_NOINTERFACE = -2147467262;
    }
}
```

Register the feed provider class object with OLE

In the Program.cs file for our executable, we will call **CoRegisterClassObject** to register our feed provider with OLE, so that the Widgets Board can interact with it. Replace the contents of Program.cs with the following code. This uses the **FeedProviderFactory** interface we defined in a previous step to register the **FeedProvider** helper class. For debugging purposes, this example calls **GetEnabledFeedProviders** on the default **FeedManager** instance to get a list of **FeedProviderInfo** objects representing the enabled feed providers. The it loops through the enabled feed providers, using the **EnabledFeedDefinitionIds** property to list all enabled feed IDs.

C#

```
// Program.cs

using Microsoft.Windows.Widgets.Feeds.Providers;
using Microsoft.Windows.Widgets.Providers;
```

```
using System;
using System.Runtime.InteropServices;

namespace ExampleFeedProvider
{

    public static class Program
    {
        [DllImport("kernel32.dll")]
        static extern IntPtr GetConsoleWindow();

        [MTAThread]
        static void Main(string[] args)
        {
            Console.WriteLine("FeedProvider Starting...");
            if (args.Length > 0 && args[0] == "-RegisterProcessAsComServer")
            {
                WinRT.ComWrappersSupport.InitializeComWrappers();

                uint registrationHandle;
                var factory = new FeedProviderFactory<FeedProvider>();
                Com.ClassObject.Register(typeof(FeedProvider).GUID, factory,
out registrationHandle);

                Console.WriteLine("Feed Provider registered.");

                var existingFeedProviders =
FeedManager.GetDefault().GetEnabledFeedProviders();
                if (existingFeedProviders != null)
                {
                    Console.WriteLine($"There are
{existingFeedProviders.Length} FeedProviders currently outstanding:");
                    foreach (var feedProvider in existingFeedProviders)
                    {
                        Console.WriteLine($"  ProviderId:
{feedProvider.FeedProviderDefinitionId}, DefinitionIds: ");
                        var m = WidgetManager.GetDefault().GetWidgetIds();
                        if (feedProvider.EnabledFeedDefinitionIds != null)
                        {
                            foreach (var enabledFeedId in
feedProvider.EnabledFeedDefinitionIds)
                            {
                                Console.WriteLine($" {enabledFeedId} ");
                            }
                        }
                    }
                }
                if (GetConsoleWindow() != IntPtr.Zero)
                {
                    Console.WriteLine("Press ENTER to exit.");
                    Console.ReadLine();
                }
                else
                {
                    while (true)
```

```
                    {
                        // You should fire an event when all the outstanding
                        // FeedProviders have been disabled and exit the
app.
                    }
                }
            }
            else
            {
                Console.WriteLine("Not being launched to service Feed
Provider... exiting.");
            }
        }
    }
}
```

Note that this code example imports the GetConsoleWindow function to determine if the app is running as a console application, the default behavior for this walkthrough. If function returns a valid pointer, we write debug information to the console. Otherwise, the app is running as a Windows app. In that case, we wait for the event that we set in **OnFeedProviderDisabled** method when the list of enabled feed providers is empty, and the we exit the app. For information on converting the example console app to a Windows app, see Convert your console app to a Windows app.

Package your feed provider app

In the current release, only packaged apps can be registered as feed providers. The following steps will take you through the process of packaging your app and updating the app manifest to register your app with the OS as a feed provider.

Create an MSIX packaging project

In **Solution Explorer**, right-click your solution and select **Add->New Project...**. In the **Add a new project** dialog, select the "Windows Application Packaging Project" template and click **Next**. Set the project name to "ExampleFeedProviderPackage" and click **Create**. When prompted, set the target version to build 22621 or later and click **OK**. Next, right-click the ExampleFeedProviderPackage project and select **Add->Project reference**. Select the **ExampleFeedProvider** project and click OK.

Add Windows App SDK package reference to the packaging project

You need to add a reference to the Windows App SDK nuget package to the MSIX packaging project. In **Solution Explorer**, double-click the ExampleFeedProviderPackage project to open the ExampleFeedProviderPackage.wapproj file. Add the following xml inside the **Project** element.

XML

```
<!--ExampleWidgetProviderPackage.wapproj-->
<ItemGroup>
    <PackageReference Include="Microsoft.WindowsAppSDK"
Version="1.5.231116003-experimentalpr">
        <IncludeAssets>build</IncludeAssets>
    </PackageReference>
</ItemGroup>
```

> **Note**
>
> Make sure the **Version** specified in the **PackageReference** element matches the latest stable version you referenced in the previous step.

If the correct version of the Windows App SDK is already installed on the computer and you don't want to bundle the SDK runtime in your package, you can specify the package dependency in the Package.appxmanifest file for the ExampleFeedProviderPackage project.

XML

```
<!--Package.appxmanifest-->
...
<Dependencies>
...
    <PackageDependency Name="Microsoft.WindowsAppRuntime.1.5.233430000-
experimental1" MinVersion="2000.638.7.0" Publisher="CN=Microsoft
Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US" />
...
</Dependencies>
...
```

Update the package manifest

In **Solution Explorer** right-click the `Package.appxmanifest` file and select **View Code** to open the manifest xml file. Next you need to add some namespace declarations for the app package extensions we will be using. Add the following namespace definitions to the top-level Package element.

XML

```
<!-- Package.appmanifest -->
<Package
  ...
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
  xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"
```

Inside the **Application** element, create a new empty element named **Extensions**. Make sure this comes after the closing tag for **uap:VisualElements**.

XML

```
<!-- Package.appxmanifest -->
<Application>
...
    <Extensions>

    </Extensions>
</Application>
```

The first extension we need to add is the ComServer extension. This registers the entry point of the executable with the OS. This extension is the packaged app equivalent of registering a COM server by setting a registry key, and is not specific to widget providers.Add the following **com:Extension** element as a child of the **Extensions** element. Change the GUID in the **Id** attribute of the **com:Class** element to the GUID that you generated in a previous step when defining the **FeedProvider** class.

XML

```
<!-- Package.appxmanifest -->
<Extensions>
    <com:Extension Category="windows.comServer">
        <com:ComServer>
            <com:ExeServer
Executable="ExampleFeedProvider\ExampleFeedProvider.exe" Arguments="-
RegisterProcessAsComServer" DisplayName="C# Feed Provider App">
                <com:Class Id="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
DisplayName="FeedProvider" />
            </com:ExeServer>
        </com:ComServer>
    </com:Extension>
</Extensions>
```

Next, add the extension that registers the app as a feed provider. Paste the uap3:Extension element in the following code snippet, as a child of the **Extensions**

element. Be sure to replace the **ClassId** attribute of the **COM** element with the GUID you used in previous steps.

XML

```
<!-- Package.appxmanifest -->
<Extensions>
    ...
    <uap3:Extension Category="windows.appExtension">
        <uap3:AppExtension Name="com.microsoft.windows.widgets.feeds"
DisplayName="ContosoFeed" Id="com.examplewidgets.examplefeed"
PublicFolder="Public">
            <uap3:Properties>
                <FeedProvider Icon="ms-appx:Assets\StoreLogo.png"
Description="FeedDescription">
                    <Activation>
                        <!-- Apps exports COM interface which implements
IFeedProvider -->
                        <CreateInstance ClassId="xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx" />
                    </Activation>
                    <Definitions>
                        <Definition Id="Contoso_Feed"
                            DisplayName="Contoso_Feed Feed"
                            Description="Feed representing Contoso"
                            ContentUri="https://www.contoso.com/"
                            Icon="ms-appx:Images\StoreLogo.png">
                        </Definition>
                        <Definition Id="Fabrikam_Feed"
                            DisplayName="Fabrikam Feed"
                            Description="Feed representing Example"
                            ContentUri="https://www.fabrikam.com/"
                            Icon="ms-appx:Images\StoreLogo.png">
                        </Definition>
                    </Definitions>
                </FeedProvider>
            </uap3:Properties>
        </uap3:AppExtension>
    </uap3:Extension>
</Extensions>
```

For detailed descriptions and format information for all of these elements, see Feed provider package manifest XML format.

Testing your feed provider

Make sure you have selected the architecture that matches your development machine from the **Solution Platforms** drop-down, for example "x64". In **Solution Explorer**, right-click your solution and select **Build Solution**. Once this is done, right-click your

ExampleWidgetProviderPackage and select **Deploy**. The console app should launch on deploy and you will see the feeds get enabled in the console output. Open the Widgets Board and you should see the new feeds in the tabs along the top of the feeds section.

Debugging your feed provider

After you have pinned your feeds, the Widget Platform will start your feed provider application in order to receive and send relevant information about the feed. To debug the running feed you can either attach a debugger to the running feed provider application or you can set up Visual Studio to automatically start debugging the feed provider process once it's started.

In order to attach to the running process:

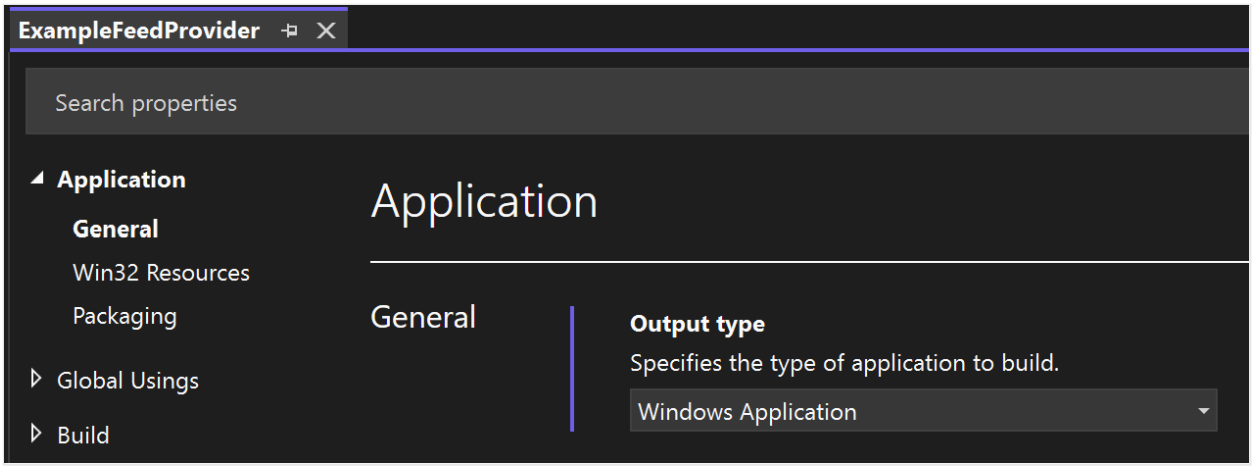
1. In Visual Studio click **Debug -> Attach to process**.
2. Filter the processes and find your desired feed provider application.
3. Attach the debugger.

In order to automatically attach the debugger to the process when it's initially started:

1. In Visual Studio click **Debug -> Other Debug Targets -> Debug Installed App Package**.
2. Filter the packages and find your desired feed provider package.
3. Select it and check the box that says Do not launch, but debug my code when it starts.
4. Click **Attach**.

Convert your console app to a Windows app

To convert the console app created in this walkthrough to a Windows app, right-click the **ExampleFeedProvider** project in **Solution Explorer** and select **Properties**. Under Application->General change the **Output type** from "Console Application" to "Windows Application".



Publishing your feed provider app

After you have developed and tested your feed provider you can publish your app on the Microsoft Store in order for users to install your feeds on their devices. For step by step guidance for publishing an app, see Publish your app in the Microsoft Store.

The Feeds Store Collection

After your app has been published on the Microsoft Store, you can request for your app to be included in the feeds Store Collection that helps users discover apps that feature Windows feeds. To submit your request, see Submit your Feed/Board for addition to the Store Collection.

Implement a feed provider in a win32 app (C++/WinRT)

Article • 12/15/2023

> **Note**
>
> **Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.**

This article walks you through creating a simple feed provider that registers a feed content URI and implements the **IFeedProvider** interface. The methods of this interface are invoked by the Widgets Board to request custom query string parameters, typically to support authentication scenarios. Feed providers can support a single feed or multiple feeds.

To implement a feed provider using C++/WinRT, see Implement a feed provider in a C# Windows app (C++/WinRT).

Prerequisites

- Your device must have developer mode enabled. For more information see Enable your device for development.
- Visual Studio 2022 or later with the **Universal Windows Platform development** workload. Make sure to add the component for C++ (v143) from the optional dropdown.

Create a new C++/WinRT win32 console app

In Visual Studio, create a new project. In the **Create a new project** dialog, set the language filter to "C++" and the platform filter to Windows, then select the Windows Console Application (C++/WinRT) project template. Name the new project "ExampleFeedProvider". For this walkthrough, make sure that **Place solution and project in the same directory** is unchecked. When prompted, set the target Windows version for the app to 10.022631.2787 or later.

Add references to the Windows App SDK and Windows Implementation Library NuGet packages

This sample uses the latest stable Windows App SDK NuGet package. In **Solution Explorer**, right-click **References** and select **Manage NuGet packages...**. In the NuGet package manager, select the **Browse** tab and search for "Microsoft.WindowsAppSDK". Select the latest stable version in the **Version** drop-down and then click **Install**.

This sample also uses the Windows Implementation Library NuGet package. In **Solution Explorer**, right-click **References** and select **Manage NuGet packages...**. In the NuGet package manager, select the **Browse** tab and search for "Microsoft.Windows.ImplementationLibrary". Select the latest version in the **Version** drop-down and then click **Install**.

In the precompiled header file, pch.h, add the following include directives.

C++

```
//pch.h
#pragma once
#include <wil/cppwinrt.h>
#include <wil/resource.h>
...
#include <winrt/Microsoft.Windows.Widgets.Providers.h>
```

> ⓘ **Note**
>
> You must include the wil/cppwinrt.h header first, before any WinRT headers.

In order to handle shutting down the feed provider app correctly, we need to a custom implementation of **winrt::get_module_lock**. We pre-declare the **SignalLocalServerShutdown** method which will be defined in our main.cpp file and will set an event that signals the app to exit. Add the following code to your pch.h file, just below the `#pragma once` directive, before the other includes.

C++

```
//pch.h
#include <stdint.h>
#include <combaseapi.h>

// In .exe local servers the class object must not contribute to the module
```

```
ref count, and use
// winrt::no_module_lock, the other objects must and this is the hook into
the C++ WinRT ref counting system
// that enables this.
void SignalLocalServerShutdown();

namespace winrt
{
    inline auto get_module_lock() noexcept
    {
        struct service_lock
        {
            uint32_t operator++() noexcept
            {
                return ::CoAddRefServerProcess();
            }

            uint32_t operator--() noexcept
            {
                const auto ref = ::CoReleaseServerProcess();

                if (ref == 0)
                {
                    SignalLocalServerShutdown();
                }
                return ref;
            }
        };

        return service_lock{};
    }
}


#define WINRT_CUSTOM_MODULE_LOCK
```

Add a FeedProvider class to handle feed operations

In Visual Studio, right-click the `ExampleFeedProvider` project in **Solution Explorer** and select **Add->Class**. In the **Add class** dialog, name the class "FeedProvider" and click **Add**.

Declare a class that implements the IFeedProvider interface

The **IFeedProvider** interface defines methods that the Widgets Board will invoke to initiate operations with the feed provider. Replace the empty class definition in the

FeedProvider.h file with the following code. This code declares a structure that implements the **IFeedProvider** interface and declares prototypes for the interface methods.

```
C++
```

```
// FeedProvider.h
#pragma once
struct FeedProvider : winrt::implements<FeedProvider,
winrt::Microsoft::Windows::Widgets::Feeds::Providers::IFeedProvider>
{
    FeedProvider() {}

    /* IFeedrovider required functions that need to be implemented */
    void
OnFeedProviderEnabled(winrt::Microsoft::Windows::Widgets::Feeds::Providers::
FeedProviderEnabledArgs args);
    void
OnFeedProviderDisabled(winrt::Microsoft::Windows::Widgets::Feeds::Providers:
:FeedProviderDisabledArgs args);
    void
OnFeedEnabled(winrt::Microsoft::Windows::Widgets::Feeds::Providers::FeedEnab
ledArgs args);
    void
OnFeedDisabled(winrt::Microsoft::Windows::Widgets::Feeds::Providers::FeedDis
abledArgs args);
    void
OnCustomQueryParametersRequested(winrt::Microsoft::Windows::Widgets::Feeds::
Providers::CustomQueryParametersRequestedArgs args);
    /* IFeedProvider required functions that need to be implemented */

};
```

Implement the IFeedProvider methods

In the next few sections, we'll implement the methods of the **IFeedProvider** interface. Before diving into the interface methods, add the following lines to `FeedProvider.cpp`, after the include directives, to pull the feed provider APIs into the **winrt** namespace and allow access to the map we declared in the previous step.

> **Note**
>
> Objects passed into the callback methods of the **IFeedProvider** interface are only guaranteed to be valid within the callback. You should not store references to these objects because their behavior outside of the context of the callback is undefined.

C++

```
// WidgetProvider.cpp
namespace winrt
{
    using namespace Microsoft::Windows::Widgets::Feeds::Providers;
}
```

OnFeedProviderEnabled

The **OnFeedProviderEnabled** method is invoked when a feed associated with the provider is created by the Widgets Board host. In the implementation of this method, generate a query string with the parameters that will be passed to the URL that provides the feed content, including any necessary authentication tokens. Create an instance of **CustomQueryParametersUpdateOptions**, passing in the **FeedProviderDefinitionId** from the event args that identifies the feed that has been enabled and the query string. Get the default **FeedManager** and call **SetCustomQueryParameters** to register the query string parameters with the Widgets Board.

C#

```
// FeedProvider.cs
void
FeedProvider::OnFeedProviderEnabled(winrt::Microsoft::Windows::Widgets::Feed
s::Providers::FeedProviderEnabledArgs args)
{
    std::wstringstream wstringstream;
wstringstream << args.FeedProviderDefinitionId().c_str() << L" feed provider
was enabled." << std::endl;
    _putws(wstringstream.str().c_str());

    auto updateOptions =
winrt::CustomQueryParametersUpdateOptions(args.FeedProviderDefinitionId(),
L"param1&param2");

winrt::FeedManager::GetDefault().SetCustomQueryParameters(updateOptions);
}
```

OnFeedProviderDisabled

OnFeedProviderDisabled is called when the Widgets Board when all of the feeds for this provider have been disabled. Feed providers are not required to perform any actions in response to these this method call. The method invocation can be used for telemetry

purposes or to update the query string parameters or revoke authentication tokens, if needed. If the app only supports a single feed provider or if all feed providers supported by the app have been disabled, then the app can exit in response to this callback.

C#

```
// FeedProvider.cs

void
FeedProvider::OnFeedProviderDisabled(winrt::Microsoft::Windows::Widgets::Fee
ds::Providers::FeedProviderDisabledArgs args)
{
    std::wstringstream wstringstream;
    wstringstream << args.FeedProviderDefinitionId().c_str() << L" feed
provider was disabled." << std::endl;
    _putws(wstringstream.str().c_str());
}
```

OnFeedEnabled, OnFeedDisabled

OnFeedEnabled and **OnFeedDisabled** are invoked by the Widgets Board when a feed is enabled or disabled. Feed providers are not required to perform any actions in response to these method calls. The method invocation can be used for telemetry purposes or to update the query string parameters or revoke authentication tokens, if needed.

C#

```
// FeedProvider.cs

void
FeedProvider::OnFeedEnabled(winrt::Microsoft::Windows::Widgets::Feeds::Provi
ders::FeedEnabledArgs args)
{
    std::wstringstream wstringstream;
    wstringstream << args.FeedDefinitionId().c_str() << L" feed was
enabled." << std::endl;
    _putws(wstringstream.str().c_str());
}
```

C#

```
// FeedProvider.cs

void
FeedProvider::OnFeedDisabled(winrt::Microsoft::Windows::Widgets::Feeds::Prov
iders::FeedDisabledArgs args)
{
    std::wstringstream wstringstream;
```

```
    wstringstream << args.FeedDefinitionId().c_str() << L" feed was
disabled." << std::endl;
    _putws(wstringstream.str().c_str());
}
```

OnCustomQueryParametersRequested

OnCustomQueryParametersRequested is raised when the Widgets Board determines that the custom query parameters associated with the feed provider need to be refreshed. For example, this method may be raised if the operation to fetch feed content from the remote web service fails. The **FeedProviderDefinitionId** property of the **CustomQueryParametersRequestedArgs** passed into this method specifies the feed for which query string params are being requested. The provider should regenerate the query string and pass it back to the Widgets Board by calling **SetCustomQueryParameters**.

C#

```
// FeedProvider.cs

void
FeedProvider::OnCustomQueryParametersRequested(winrt::Microsoft::Windows::Wi
dgets::Feeds::Providers::CustomQueryParametersRequestedArgs args)
{
    std::wstringstream wstringstream;
    wstringstream << L"CustomQueryParameters were requested for " <<
args.FeedProviderDefinitionId().c_str() << std::endl;
    _putws(wstringstream.str().c_str());

    auto updateOptions =
winrt::CustomQueryParametersUpdateOptions(args.FeedProviderDefinitionId(),
L"param1&param2");

winrt::FeedManager::GetDefault().SetCustomQueryParameters(updateOptions);
}
```

Register a class factory that will instantiate FeedProvider on request

Add the header that defines the **FeedProvider** class to the includes at the top of your app's `main.cpp` file. We will also be including **mutex** here.

C++

```
// main.cpp
...
#include "FeedProvider.h"
#include <mutex>
```

Declare the event that will trigger our app to exit and the **SignalLocalServerShutdown** function that will set the event. Paste the following code in main.cpp.

C++

```
// main.cpp
wil::unique_event g_shudownEvent(wil::EventOptions::None);

void SignalLocalServerShutdown()
{
    g_shudownEvent.SetEvent();
}
```

Next, you will need to create a CLSID that will be used to identify your feed provider for COM activation. Generate a GUID in Visual Studio by going to **Tools->Create GUID**. Select the option "static const GUID =" and click **Copy** and then paste that into `main.cpp`. Update the GUID definition with the following C++/WinRT syntax, setting the GUID variable name feed_provider_clsid. Leave the commented version of the GUID because you will need this format later, when packaging your app.

C++

```
// main.cpp
...
// {80F4CB41-5758-4493-9180-4FB8D480E3F5}
static constexpr GUID feed_provider_clsid
{
    0x80f4cb41, 0x5758, 0x4493, { 0x91, 0x80, 0x4f, 0xb8, 0xd4, 0x80, 0xe3,
0xf5 }
};
```

Add the following class factory definition to `main.cpp`. This is mostly boilerplate code that is not specific to feed provider implementations. Note that **CoWaitForMultipleObjects** waits for our shutdown event to be triggered before the app exits.

C++

```
// main.cpp
template <typename T>
struct SingletonClassFactory : winrt::implements<SingletonClassFactory<T>,
```

```
IClassFactory>
{
    STDMETHODIMP CreateInstance(
        ::IUnknown* outer,
        GUID const& iid,
        void** result) noexcept final
    {
        *result = nullptr;

        std::unique_lock lock(mutex);

        if (outer)
        {
            return CLASS_E_NOAGGREGATION;
        }

        if (!instance)
        {
            instance = winrt::make<FeedProvider>();
        }

        return instance.as(iid, result);
    }

    STDMETHODIMP LockServer(BOOL) noexcept final
    {
        return S_OK;
    }

private:
    T instance{ nullptr };
    std::mutex mutex;
};

int main()
{
    winrt::init_apartment();
    wil::unique_com_class_object_cookie feedProviderFactory;
    auto factory =
winrt::make<SingletonClassFactory<winrt::Microsoft::Windows::Widgets::Feeds:
:Providers::IFeedProvider>>();

    winrt::check_hresult(CoRegisterClassObject(
        feed_provider_clsid,
        factory.get(),
        CLSCTX_LOCAL_SERVER,
        REGCLS_MULTIPLEUSE,
        feedProviderFactory.put()));

    DWORD index{};
    HANDLE events[] = { g_shudownEvent.get() };
    winrt::check_hresult(CoWaitForMultipleObjects(CWMO_DISPATCH_CALLS |
CWMO_DISPATCH_WINDOW_MESSAGES,
        INFINITE,
        static_cast<ULONG>(std::size(events)), events, &index));
```

```
    return 0;
}
```

Package your feed provider app

In the current release, only packaged apps can be registered as feed providers. The following steps will take you through the process of packaging your app and updating the app manifest to register your app with the OS as a feed provider.

Create an MSIX packaging project

In **Solution Explorer**, right-click your solution and select **Add->New Project...**. In the **Add a new project** dialog, select the "Windows Application Packaging Project" template and click **Next**. Set the project name to "ExampleFeedProviderPackage" and click **Create**. When prompted, set the target version to version 1809 or later and click **OK**. Next, right-click the ExampleFeedProviderPackage project and select **Add->Project reference**. Select the **ExampleFeedProvider** project and click OK.

Add Windows App SDK package reference to the packaging project

You need to add a reference to the Windows App SDK nuget package to the MSIX packaging project. In **Solution Explorer**, double-click the ExampleFeedProviderPackage project to open the ExampleFeedProviderPackage.wapproj file. Add the following xml inside the **Project** element.

XML

```
<!--ExampleFeedProviderPackage.wapproj-->
<ItemGroup>
    <PackageReference Include="Microsoft.WindowsAppSDK"
Version="1.5.231116003-experimentalpr">
        <IncludeAssets>build</IncludeAssets>
    </PackageReference>
</ItemGroup>
```

> ⓘ **Note**
>
> Make sure the **Version** specified in the **PackageReference** element matches the latest stable version you referenced in the previous step.

If the correct version of the Windows App SDK is already installed on the computer and you don't want to bundle the SDK runtime in your package, you can specify the package dependency in the Package.appxmanifest file for the ExampleFeedProviderPackage project.

XML

```
<!--Package.appxmanifest-->
...
<Dependencies>
...
    <PackageDependency Name="Microsoft.WindowsAppRuntime.1.5.233430000-
experimental1" MinVersion="2000.638.7.0" Publisher="CN=Microsoft
Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US" />
...
</Dependencies>
...
```

Update the package manifest

In **Solution Explorer** right-click the `Package.appxmanifest` file and select **View Code** to open the manifest xml file. Next you need to add some namespace declarations for the app package extensions we will be using. Add the following namespace definitions to the top-level Package element.

XML

```
<!-- Package.appmanifest -->
<Package
  ...
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
  xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"
```

Inside the **Application** element, create a new empty element named **Extensions**. Make sure this comes after the closing tag for **uap:VisualElements**.

XML

```
<!-- Package.appxmanifest -->
<Application>
...
    <Extensions>

    </Extensions>
</Application>
```

The first extension we need to add is the ComServer extension. This registers the entry point of the executable with the OS. This extension is the packaged app equivalent of registering a COM server by setting a registry key, and is not specific to feed providers.Add the following **com:Extension** element as a child of the **Extensions** element. Change the GUID in the **Id** attribute of the **com:Class** element to the GUID you generated in a previous step.

XML

```
<!-- Package.appxmanifest -->
<Extensions>
    <com:Extension Category="windows.comServer">
        <com:ComServer>
            <com:ExeServer
Executable="ExampleFeedProvider\ExampleFeedProvider.exe" Arguments="-
RegisterProcessAsComServer" DisplayName="C++ Feed Provider App">
                <com:Class Id="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
DisplayName="FeedProvider" />
            </com:ExeServer>
        </com:ComServer>
    </com:Extension>
</Extensions>
```

Next, add the extension that registers the app as a feed provider. Paste the uap3:Extension element in the following code snippet, as a child of the **Extensions** element. Be sure to replace the **ClassId** attribute of the **COM** element with the GUID you used in previous steps.

XML

```
<!-- Package.appxmanifest -->
<Extensions>
    ...
    <uap3:Extension Category="windows.appExtension">
        <uap3:AppExtension Name="com.microsoft.windows.widgets.feeds"
DisplayName="ContosoFeed" Id="com.examplewidgets.examplefeed"
PublicFolder="Public">
            <uap3:Properties>
                <FeedProvider Icon="ms-appx:Assets\StoreLogo.png"
Description="FeedDescription">
                    <Activation>
                        <!-- Apps exports COM interface which implements
IFeedProvider -->
                        <CreateInstance ClassId="xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx" />
                    </Activation>
                    <Definitions>
                        <Definition Id="Contoso_Feed"
                            DisplayName="Contoso_Feed Feed"
                            Description="Feed representing Contoso"
```

```
                            ContentUri="https://www.contoso.com/"
                            Icon="ms-appx:Images\StoreLogo.png">
                        </Definition>
                        <Definition Id="Fabrikam_Feed"
                            DisplayName="Fabrikam Feed"
                            Description="Feed representing Example"
                            ContentUri="https://www.fabrikam.com/"
                            Icon="ms-appx:Images\StoreLogo.png">
                        </Definition>
                    </Definitions>
                </FeedProvider>
            </uap3:Properties>
        </uap3:AppExtension>
    </uap3:Extension>
</Extensions>
```

For detailed descriptions and format information for all of these elements, see Feed provider package manifest XML format.

Add icons to your packaging project

In **Solution Explorer**, right-click your **ExampleFeedProviderPackage** and select **Add->New Folder**. Name this folder ProviderAssets as this is what was used in the `Package.appxmanifest` from the previous step. This is where we will store our **Icon** for our feeds. Once you add your desired Icons , make sure the image names match what comes after **Path=ProviderAssets** in your `Package.appxmanifest` or the feeds will not show up in the Widget Board.

Testing your feed provider

Make sure you have selected the architecture that matches your development machine from the **Solution Platforms** drop-down, for example "x64". In **Solution Explorer**, right-click your solution and select **Build Solution**. Once this is done, right-click your **ExampleWidgetProviderPackage** and select **Deploy**. The console app should launch on deploy and you will see the feeds get enabled in the console output. Open the Widgets Board and you should see the new feeds in the tabs along the top of the feeds section.

Debugging your feed provider

After you have pinned your feeds, the Widget Platform will start your feed provider application in order to receive and send relevant information about the feed. To debug the running feed you can either attach a debugger to the running feed provider

application or you can set up Visual Studio to automatically start debugging the feed provider process once it's started.

In order to attach to the running process:

1. In Visual Studio click **Debug -> Attach to process**.
2. Filter the processes and find your desired feed provider application.
3. Attach the debugger.

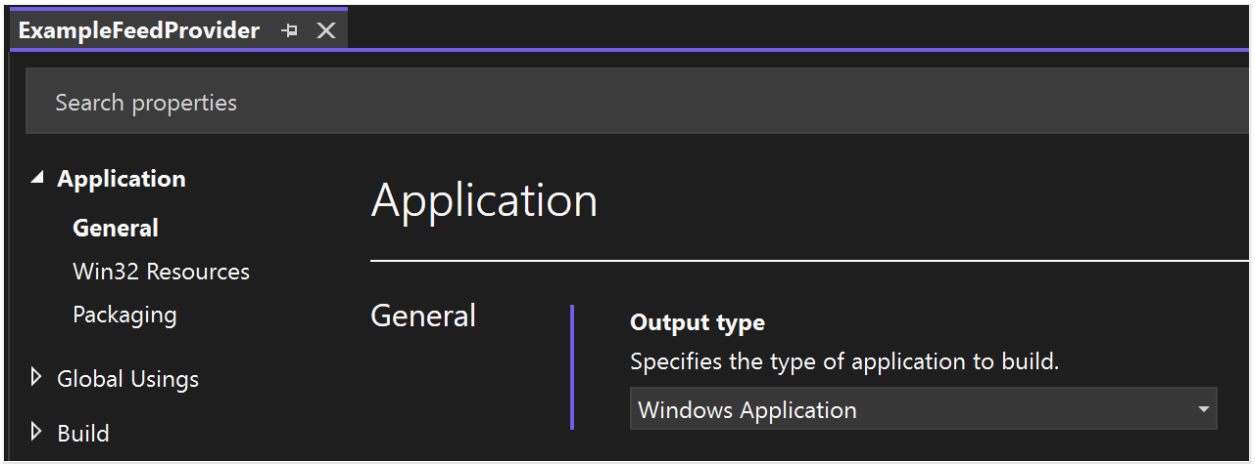
In order to automatically attach the debugger to the process when it's initially started:

1. In Visual Studio click **Debug -> Other Debug Targets -> Debug Installed App Package**.
2. Filter the packages and find your desired feed provider package.
3. Select it and check the box that says Do not launch, but debug my code when it starts.
4. Click **Attach**.

Convert your console app to a Windows app

To convert the console app created in this walkthrough to a Windows app:

1. Right-click on the ExampleWidgetProvider project in **Solution Explorer** and select **Properties**. Navigate to **Linker -> System** and change **SubSystem** from "Console" to "Windows". This can also be done by adding <SubSystem>Windows</SubSystem> to the <Link>..</Link> section of the .vcxproj.
2. In main.cpp, change `int main()` to `int WINAPI wWinMain(_In_ HINSTANCE /*hInstance*/, _In_opt_ HINSTANCE /*hPrevInstance*/, _In_ PWSTR pCmdLine, _In_ int /*nCmdShow*/)`.



Publishing your feed provider app

After you have developed and tested your feed provider you can publish your app on the Microsoft Store in order for users to install your feeds on their devices. For step by step guidance for publishing an app, see Publish your app in the Microsoft Store.

The Feeds Store Collection

After your app has been published on the Microsoft Store, you can request for your app to be included in the feeds Store Collection that helps users discover apps that feature Windows feeds. To submit your request, see Submit your Feed/Board for addition to the Store Collection.

Feed provider package manifest XML format

Article • 09/24/2024

> **Note**
>
> **Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.**

In order to be displayed on the Widgets Board, apps that support Windows feeds must register their feed provider with the system. For Win32 apps, only packaged apps are currently supported and feed providers specify their registration information in the app package manifest file. This article documents the XML format for feed registration. See the Example section for a code listing of an example package manifest for a Win32 feed provider.

App extension

The app package manifest file supports many different extensions and features for Windows apps. The app package manifest format is defined by a set of schemas that are documented in the Package manifest schema reference. Feed providers declare their registration information within the uap3:AppExtension. The **Name** attribute of the extension must be set to "com.microsoft.windows.widgets.feeds".

Feed providers should include the uap3:Properties as the child of **uap3:AppExtension**. The package manifest schema does not enforce the structure of the **uap3:Properties** element other than requiring well-formed XML. The rest of this article describes the XML format that the Widgets Board expects in order to successfully register a feed provider.

XML

```
<uap3:Extension Category="windows.appExtension">
  <uap3:AppExtension Name="com.microsoft.windows.widgets.feeds"
DisplayName="ContosoApp" Id="ContosoApp" PublicFolder="Public">
    <uap3:Properties>
      <!-- Feed provider registration content goes here -->
    </uap3:Properties>
  </uap3:AppExtension>
</uap3:Extension>
```

Element hierarchy

FeedProvider

Activation

CreateInstance

Definitions

Definition

FeedProvider

The root element of the feed provider registration information.

Expand table

Attribute	Type	Required	Description	Default value
Description	string	Yes	A short description of the feed provider.	N/A
DisplayName	string	Yes	The name of the feed provider that is displayed on the Widgets Board.	N/A
Icon	string	Yes	The package-relative path to an icon image file that is displayed in the Widgets Board.	N/A
Id	string	Yes	An ID that identifies the feed provider. Feed provider implementations use this string to determine or specify which of the app's feed providers is being referenced for each operation. This string must be unique for all feed providers defined within the app manifest file.	N/A

Activation

Specifies activation information for the feed provider.

CreateInstance

CreateInstance should be specified for Win32-based feed providers that implement the **IFeedProvider** interface. The system will activate the interface with a call to

CoCreateInstance. The **ClassId** attribute specifies the CLSID for the CreateInstance server that implements the **IFeedProvider** interface.

Expand table

Attribute	Type	Required	Description	Default value
ClassId	GUID	Yes	The CLSID for the CreateInstance server that implements the feed provider.	N/A

Definitions

The container element for one or more feed registrations.

Definition

Represents the registration for a single feed.

Expand table

Attribute	Type	Required	Description	Default value
Id	string	Yes	An ID that identifies the feed. Feed provider implementations use this string to determine or specify which of the app's feeds is being referenced for each operation. This string must be unique for all feeds defined within the app manifest file.	N/A
DisplayName	string	Yes	The name of the feed that is displayed on the Widgets Board.	N/A
Description	string	Yes	A short description of the feed.	N/A
ContentUri	string	Yes	The URI from which feed content is retrieved.	N/A
Icon	string	Yes	The package-relative path to an icon image file that is displayed in the Widgets Board.	N/A
WebRequestFilter	string	No	A web request filter string specifying the set of URLs for which the resource requests will be redirected to the feed provider's implementation of **IFeedResourceProvider**. The pattern is expressed using the format	N/A

Attribute	Type	Required	Description	Default value
			described in Match Patterns. The filter string in the registration must use Punycode where necessary. All content types will be redirected when matched so the filter should only resolve to content intended to be obtained through the **IFeedResourceProvider** in the application.	
ExcludedRegions	string	No	A list of regions where the feed should not be available. Feeds can specify **ExcludedRegions** or **ExclusiveRegions** but must not specify both in a single feed definition. The value of the attribute is a comma separated list of two character region codes.	N/A
ExclusiveRegions	string	No	A list of the only regions where the feed should be available. Feeds can specify **ExcludedRegions** or **ExclusiveRegions** but must not specify both in single feed definition. The value of the attribute is a comma separated list of two character region codes.	N/A

You can use localized resources instead of string literals for the UI-facing attribute values. For more information, see Localize strings in your UI and app package manifest.

Example

The following code example illustrates the usage of the feed package manifest XML format.

XML

```
<uap3:AppExtension Name="com.microsoft.windows.widgets.feeds"
DisplayName="ContosoApp" Id="ContosoApp" PublicFolder="Public">
  <uap3:Properties>
      <FeedProvider Description="ms-resource:ProviderDescription" Icon="ms-
appx:Images\ContosoProviderIcon.png">
          <Activation>
              <CreateInstance ClassId="ECB883FD-3755-4E1C-BECA-D3397A3FF15C"
/>
          </Activation>
          <Definitions>
              <Definition Id="Contoso_Feed"
                  DisplayName="ms-resource:FeedDisplayName"
                  Description="ms-resource:FeedDescription"
```

```
                    ContentUri="https://contoso.com/news"
                    Icon="ms-appx:Images\ContosoFeedIcon.png"

WebRequestFilter="https://contoso.com/*/feed/appResource/*"
                    ExcludedRegions="FR,DE,IT">
                </Definition>
            </Definitions>
        </FeedProvider>
    </uap3:Properties>
</uap3:AppExtension>
```

Widget providers

Article • 03/09/2023

Windows widgets are small UI containers that display text and graphics supplied by a Windows app or Progressive Web App (PWA). The Adaptive Cards format used by Windows widgets enables dynamic binding of the data that populates the widget UI. To update your widget, your app or service will implement a widget service provider that responds to requests from the Widgets host and returns JSON strings specifying both the visual template and the associated data for your widget.

For an overview of the Windows widgets experience and design guidance for creating your own widgets, see Windows widgets.

Currently you can implement a widget provider using a packaged Win32 desktop app or a Progressive Web App (PWA). For more information see:

- Implement a widget provider in a win32 app (C#)
- Implement a widget provider in a win32 app (C++/WinRT)
- Build PWA-driven widgets

For API reference documentation for implementing widget providers, see the Microsoft.Windows.Widgets.Providers namespace.

For a walkthrough of the basics of creating a widget and implementing a widget provider, watch the *Tabs vs Spaces* episode "Create Widgets for Windows 11".
https://learn-video.azurefd.net/vod/player?show=tabs-vs-spaces&ep=create-widgets-for-windows-11&locale=en-us&embedUrl=%2Fwindows%2Fapps%2Fdevelop%2Fwidgets%2Fwidget-providers

Additional resources

- Report an issue via email
- Open an issue in the Windows App SDK github repo

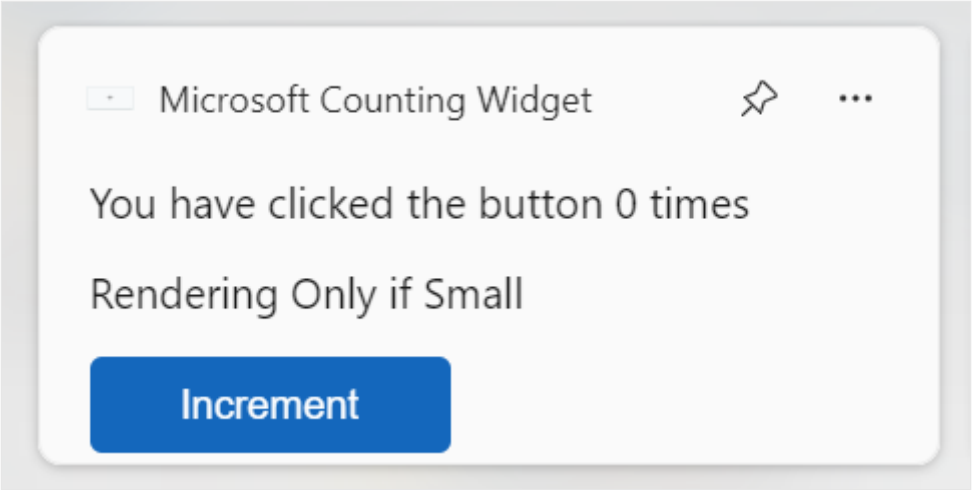
Related articles

- Windows widgets

Implement a widget provider in a C# Windows App

Article • 10/02/2023

This article walks you through creating a simple widget provider that implements the IWidgetProvider interface. The methods of this interface are invoked by the widget host to request the data that defines a widget or to let the widget provider respond to a user action on a widget. Widget providers can support a single widget or multiple widgets. In this example, we will define two different widgets. One widget is a mock weather widget that illustrates some of the formatting options provided by the Adaptive Cards framework. The second widget will demonstrate user actions and the custom widget state feature by maintaining a counter that is incremented whenever the user clicks on a button displayed on the widget.



This sample code in this article is adapted from the Windows App SDK Widgets Sample. To implement a widget provider using C++/WinRT, see Implement a widget provider in a win32 app (C++/WinRT).

Prerequisites

- Your device must have developer mode enabled. For more information see Enable your device for development.
- Visual Studio 2022 or later with the **Universal Windows Platform development** workload. Make sure to add the component for C++ (v143) from the optional dropdown.

Create a new C# console app

In Visual Studio, create a new project. In the **Create a new project** dialog, set the language filter to "C#" and the platform filter to Windows, then select the Console App project template. Name the new project "ExampleWidgetProvider". When prompted, set the target .NET version to 8.0.

When the project loads, in **Solution Explorer** right-click the project name and select **Properties**. On the **General** page, scroll down to **Target OS** and select "Windows". Under **Target OS Version**, select version 10.0.19041.0 or later.

To update your project to support .NET 8.0, in **Solution Explorer** right-click the project name and select **Edit Project File**. Inside of **PropertyGroup**, add the following **RuntimeIdentifiers** element.

XML

```
<RuntimeIdentifiers>win-x86;win-x64;win-arm64</RuntimeIdentifiers>
```

Note that this walkthrough uses a console app that displays the console window when the widget is activated to enable easy debugging. When you are ready to publish your widget provider app, you can convert the console application to a Windows application by following the steps in Convert your console app to a Windows app.

Add references to the Windows App SDK

This sample uses the latest stable Windows App SDK NuGet package. In **Solution Explorer**, right-click **Dependencies** and select **Manage NuGet packages...**. In the NuGet package manager, select the **Browse** tab and search for "Microsoft.WindowsAppSDK". Select the latest stable version in the **Version** drop-down and then click **Install**.

Add a WidgetProvider class to handle widget operations

In Visual Studio, right-click the `ExampleWidgetProvider` project in **Solution Explorer** and select **Add->Class**. In the **Add class** dialog, name the class "WidgetProvider" and click **Add**. In the generated WidgetProvider.cs file, update the class definition to indicate that it implements the **IWidgetProvider** interface.

C#

```
// WidgetProvider.cs
internal class WidgetProvider : IWidgetProvider
```

Prepare to track enabled widgets

A widget provider can support a single widget or multiple widgets. Whenever the widget host initiates an operation with the widget provider, it passes an ID to identify the widget associated with the operation. Each widget also has an associated name and a state value that can be used to store custom data. For this example, we'll declare a simple helper structure to store the ID, name, and data for each pinned widget. Widgets also can be in an active state, which is discussed in the Activate and Deactivate section below, and we will track this state for each widget with a boolean value. Add the following definition to the WidgetProvider.cs file, inside the **ExampleWidgetProvider** namespace, but outside of the **WidgetProvider** class definition.

C#

```
// WidgetProvider.cs

public class CompactWidgetInfo
{
    public string? widgetId { get; set; }
    public string? widgetName { get; set; }
    public int customState = 0;
    public bool isActive = false;

}
```

Inside the **WidgetProvider** class definition in WidgetProvider.cs, add a member for the map that will maintain the list of enabled widgets, using the widget ID as the key for each entry.

C#

```
// WidgetProvider.cs

// Class member of WidgetProvider
public static Dictionary<string, CompactWidgetInfo> RunningWidgets = new
Dictionary<string, CompactWidgetInfo>();
```

Declare widget template JSON strings

This example will declare some static strings to define the JSON templates for each widget. For convenience, these templates are stored in the member variables of the **WidgetProvider** class. If you need a general storage for the templates - they can be included as part of the application package: Accessing Package Files. For information on creating the widget template JSON document, see Create a widget template with the Adaptive Card Designer.

In the latest release, apps that implement Windows widgets can customize the header that is displayed for their widget in the Widgets Board, overriding the default presentation. For more information, see Customize the widget header area.

> ⓘ **Note**
>
> In the latest release, apps that implement Windows widgets can choose to populate the widget content with HTML served from a specified URL instead of supplying content in the Adaptive Card schema format in the JSON payload passed from the provider to the Widgets Board. Widget providers must still provide an

Adaptive Card JSON payload, so the implementation steps in this walkthrough are applicable to web widgets. For more information, see Web widget providers.

C#

```
// WidgetProvider.cs

// Class members of WidgetProvider
        const string weatherWidgetTemplate = """
{
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "type": "AdaptiveCard",
    "version": "1.0",
    "speak": "<s>The forecast for Seattle January 20 is mostly clear with a
High of 51 degrees and Low of 40 degrees</s>",
    "backgroundImage":
"https://messagecardplayground.azurewebsites.net/assets/Mostly%20Cloudy-
Background.jpg",
    "body": [
        {
            "type": "TextBlock",
            "text": "Redmond, WA",
            "size": "large",
            "isSubtle": true,
            "wrap": true
        },
        {
            "type": "TextBlock",
            "text": "Mon, Nov 4, 2019 6:21 PM",
            "spacing": "none",
            "wrap": true
        },
        {
            "type": "ColumnSet",
            "columns": [
                {
                    "type": "Column",
                    "width": "auto",
                    "items": [
                        {
                            "type": "Image",
                            "url":
"https://messagecardplayground.azurewebsites.net/assets/Mostly%20Cloudy-
Square.png",
                            "size": "small",
                            "altText": "Mostly cloudy weather"
                        }
                    ]
                },
                {
                    "type": "Column",
                    "width": "auto",
                    "items": [
```

```
                        {
                            "type": "TextBlock",
                            "text": "46",
                            "size": "extraLarge",
                            "spacing": "none",
                            "wrap": true
                        }
                    ]
                },
                {
                    "type": "Column",
                    "width": "stretch",
                    "items": [
                        {
                            "type": "TextBlock",
                            "text": "°F",
                            "weight": "bolder",
                            "spacing": "small",
                            "wrap": true
                        }
                    ]
                },
                {
                    "type": "Column",
                    "width": "stretch",
                    "items": [
                        {
                            "type": "TextBlock",
                            "text": "Hi 50",
                            "horizontalAlignment": "left",
                            "wrap": true
                        },
                        {
                            "type": "TextBlock",
                            "text": "Lo 41",
                            "horizontalAlignment": "left",
                            "spacing": "none",
                            "wrap": true
                        }
                    ]
                }
            ]
        }
    ]
}
""";

    const string countWidgetTemplate = """
{
    "type": "AdaptiveCard",
    "body": [
        {
            "type": "TextBlock",
            "text": "You have clicked the button ${count} times"
        },
```

```
        {
                "text":"Rendering Only if Small",
                "type":"TextBlock",
                "$when":"${$host.widgetSize==\"small\"}"
        },
        {
                "text":"Rendering Only if Medium",
                "type":"TextBlock",
                "$when":"${$host.widgetSize==\"medium\"}"
        },
        {
            "text":"Rendering Only if Large",
            "type":"TextBlock",
            "$when":"${$host.widgetSize==\"large\"}"
        }
    ],
    "actions": [
        {
            "type": "Action.Execute",
            "title": "Increment",
            "verb": "inc"
        }
    ],
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "version": "1.5"
}
""";
```

Implement the IWidgetProvider methods

In the next few sections, we'll implement the methods of the **IWidgetProvider** interface. The helper method **UpdateWidget** that is called in several of these method implementations will be shown later in this article.

> ⓘ Note
>
> Objects passed into the callback methods of the **IWidgetProvider** interface are only guaranteed to be valid within the callback. You should not store references to these objects because their behavior outside of the context of the callback is undefined.

CreateWidget

The widget host calls CreateWidget when the user has pinned one of your app's widgets in the widget host. First, this method gets the ID and name of the associated widget and adds a new instance of our helper structure, **CompactWidgetInfo**, to the collection of

enabled widgets. Next, we send the initial template and data for the widget, which is encapsulated in the **UpdateWidget** helper method.

C#

```
// WidgetProvider.cs

public void CreateWidget(WidgetContext widgetContext)
{
    var widgetId = widgetContext.Id; // To save RPC calls
    var widgetName = widgetContext.DefinitionId;
    CompactWidgetInfo runningWidgetInfo = new CompactWidgetInfo() { widgetId
= widgetId, widgetName = widgetName };
    RunningWidgets[widgetId] = runningWidgetInfo;


    // Update the widget
    UpdateWidget(runningWidgetInfo);
}
```

DeleteWidget

The widget host calls DeleteWidget when the user has unpinned one of your app's widgets from the widget host. When this occurs, we will remove the associated widget from our list of enabled widgets so that we don't send any further updates for that widget.

C#

```
// WidgetProvider.cs

public void DeleteWidget(string widgetId, string customState)
{
    RunningWidgets.Remove(widgetId);

    if(RunningWidgets.Count == 0)
    {
        emptyWidgetListEvent.Set();
    }
}
```

For this example, in addition to removing the widget with the specified from the list of enabled widgets, we also check to see if the list is now empty, and if so, we set an event that will be used later to allow the app to exit when there are no enabled widgets. Inside your class definition, add the declaration of the ManualResetEvent and a public accessor function.

C#

```
// WidgetProvider.cs
static ManualResetEvent emptyWidgetListEvent = new ManualResetEvent(false);

public static ManualResetEvent GetEmptyWidgetListEvent()
{
    return emptyWidgetListEvent;
}
```

OnActionInvoked

The widget host calls OnActionInvoked when the user interacts with an action you defined in your widget template. For the counter widget used in this example, an action was declared with a **verb** value of "inc" in the JSON template for the widget. The widget provider code will use this **verb** value to determine what action to take in response to the user interaction.

JSON

```
...
    "actions": [
        {
            "type": "Action.Execute",
            "title": "Increment",
            "verb": "inc"
        }
    ],
...
```

In the **OnActionInvoked** method, get the verb value by checking the **Verb** property of the WidgetActionInvokedArgs passed into the method. If the verb is "inc", then we know we are going to increment the count in the custom state for the widget. From the **WidgetActionInvokedArgs**, get the WidgetContext object and then the **WidgetId** to get the ID for the widget that is being updated. Find the entry in our enabled widgets map with the specified ID and then update the custom state value that is used to store the number of increments. Finally, update the widget content with the new value with the **UpdateWidget** helper function.

C#

```
// WidgetProvider.cs

public void OnActionInvoked(WidgetActionInvokedArgs actionInvokedArgs)
{
    var verb = actionInvokedArgs.Verb;
```

```
        if (verb == "inc")
        {
            var widgetId = actionInvokedArgs.WidgetContext.Id;
            // If you need to use some data that was passed in after
            // Action was invoked, you can get it from the args:
            var data = actionInvokedArgs.Data;
            if (RunningWidgets.ContainsKey(widgetId))
            {
                var localWidgetInfo = RunningWidgets[widgetId];
                // Increment the count
                localWidgetInfo.customState++;
                UpdateWidget(localWidgetInfo);
            }
        }
}
```

For information about the **Action.Execute** syntax for Adaptive Cards, see Action.Execute. For guidance about designing interaction for widgets, see Widget interaction design guidance

OnWidgetContextChanged

In the current release, OnWidgetContextChanged is only called when the user changes the size of a pinned widget. You can choose to return a different JSON template/data to the widget host depending on what size is requested. You can also design the template JSON to support all the available sizes using conditional rendering based on the value of **host.widgetSize**. If you don't need to send a new template or data to account for the size change, you can use the **OnWidgetContextChanged** for telemetry purposes.

C#

```
// WidgetProvider.cs

public void OnWidgetContextChanged(WidgetContextChangedArgs
contextChangedArgs)
{
    var widgetContext = contextChangedArgs.WidgetContext;
    var widgetId = widgetContext.Id;
    var widgetSize = widgetContext.Size;
    if (RunningWidgets.ContainsKey(widgetId))
    {
        var localWidgetInfo = RunningWidgets[widgetId];
        UpdateWidget(localWidgetInfo);
    }
}
```

Activate and Deactivate

The Activate method is called to notify the widget provider that the widget host is currently interested in receiving updated content from the provider. For example, it could mean that the user is currently actively viewing the widget host. The Deactivate method is called to notify the widget provider that the widget host is no longer requesting content updates. These two methods define a window in which the widget host is most interested in showing the most up-to-date content. Widget providers can send updates to the widget at any time, such as in response to a push notification, but as with any background task, it's important to balance providing up-to-date content with resource concerns like battery life.

Activate and **Deactivate** are called on a per-widget basis. This example tracks the active status of each widget in the **CompactWidgetInfo** helper struct. In the **Activate** method, we call the **UpdateWidget** helper method to update our widget. Note that the time window between **Activate** and **Deactivate** may be small, so it's recommended that you try to make your widget update code path as quick as possible.

C#

```
// WidgetProvider.cs

public void Activate(WidgetContext widgetContext)
{
    var widgetId = widgetContext.Id;

    if (RunningWidgets.ContainsKey(widgetId))
    {
        var localWidgetInfo = RunningWidgets[widgetId];
        localWidgetInfo.isActive = true;

        UpdateWidget(localWidgetInfo);
    }
}
public void Deactivate(string widgetId)
{
    if (RunningWidgets.ContainsKey(widgetId))
    {
        var localWidgetInfo = RunningWidgets[widgetId];
        localWidgetInfo.isActive = false;
    }
}
```

Update a widget

Define the **UpdateWidget** helper method to update an enabled widget. In this example, we check the name of the widget in the **CompactWidgetInfo** helper struct passed into the method, and then set the appropriate template and data JSON based on which widget is being updated. A **WidgetUpdateRequestOptions** is initialized with the template, data, and custom state for the widget being updated. Call WidgetManager::GetDefault to get an instance of the WidgetManager class and then call UpdateWidget to send the updated widget data to the widget host.

C#

```
// WidgetProvider.cs

void UpdateWidget(CompactWidgetInfo localWidgetInfo)
{
    WidgetUpdateRequestOptions updateOptions = new
WidgetUpdateRequestOptions(localWidgetInfo.widgetId);

    string? templateJson = null;
    if (localWidgetInfo.widgetName == "Weather_Widget")
    {
        templateJson = weatherWidgetTemplate.ToString();
    }
    else if (localWidgetInfo.widgetName == "Counting_Widget")
    {
        templateJson = countWidgetTemplate.ToString();
    }

    string? dataJson = null;
    if (localWidgetInfo.widgetName == "Weather_Widget")
    {
        dataJson = "{}";
    }
    else if (localWidgetInfo.widgetName == "Counting_Widget")
    {
        dataJson = "{ \"count\": " + localWidgetInfo.customState.ToString()
+ " }";
    }

    updateOptions.Template = templateJson;
    updateOptions.Data = dataJson;
    // You can store some custom state in the widget service that you will
be able to query at any time.
    updateOptions.CustomState= localWidgetInfo.customState.ToString();
    WidgetManager.GetDefault().UpdateWidget(updateOptions);
}
```

Initialize the list of enabled widgets on startup

When our widget provider is first initialized, it's a good idea to ask **WidgetManager** if there are any running widgets that our provider is currently serving. It will help to recover the app to the previous state in case of the computer restart or the provider crash. Call **WidgetManager.GetDefault** to get the default widget manager instance for the app. Then call GetWidgetInfos, which returns an array of WidgetInfo objects. Copy the widget IDs, names, and custom state into the helper struct **CompactWidgetInfo** and save it to the **RunningWidgets** member variable. Paste the following code into the class definition for the **WidgetProvider** class.

C#

```
// WidgetProvider.cs

public WidgetProvider()
{
    var runningWidgets = WidgetManager.GetDefault().GetWidgetInfos();

    foreach (var widgetInfo in runningWidgets)
    {
        var widgetContext = widgetInfo.WidgetContext;
        var widgetId = widgetContext.Id;
        var widgetName = widgetContext.DefinitionId;
        var customState = widgetInfo.CustomState;
        if (!RunningWidgets.ContainsKey(widgetId))
        {
            CompactWidgetInfo runningWidgetInfo = new CompactWidgetInfo() {
widgetId = widgetId, widgetName = widgetName };
            try
            {
                // If we had any save state (in this case we might have some
state saved for Counting widget)
                // convert string to required type if needed.
                int count = Convert.ToInt32(customState.ToString());
                runningWidgetInfo.customState = count;
            }
            catch
            {

            }
            RunningWidgets[widgetId] = runningWidgetInfo;
        }
    }
}
```

Implement a class factory that will instantiate WidgetProvider on request

In order for the widget host to communicate with our widget provider, we must call CoRegisterClassObject. This function requires us to create an implementation of the IClassFactory that will create a class object for our **WidgetProvider** class. We will implement our class factory in a self-contained helper class.

In Visual Studio, right-click the `ExampleWidgetProvider` project in **Solution Explorer** and select **Add->Class**. In the **Add class** dialog, name the class "FactoryHelper" and click **Add**.

Replace the contents of the FactoryHelper.cs file with the following code. This code defines the **IClassFactory** interface and implements it's two methods, CreateInstance and LockServer. This code is typical boilerplate for implementing a class factory and is not specific to the functionality of a widget provider except that we indicate that the class object being created implements the **IWidgetProvider** interface.

C#

```
// FactoryHelper.cs

using Microsoft.Windows.Widgets.Providers;
using System.Runtime.InteropServices;
using WinRT;

namespace COM
{
    static class Guids
    {
        public const string IClassFactory = "00000001-0000-0000-C000-
000000000046";
        public const string IUnknown = "00000000-0000-0000-C000-
000000000046";
    }

    ///
    /// IClassFactory declaration
    ///
    [ComImport, ComVisible(false),
InterfaceType(ComInterfaceType.InterfaceIsIUnknown),
Guid(COM.Guids.IClassFactory)]
    internal interface IClassFactory
    {
        [PreserveSig]
        int CreateInstance(IntPtr pUnkOuter, ref Guid riid, out IntPtr
ppvObject);
        [PreserveSig]
        int LockServer(bool fLock);
    }

    [ComVisible(true)]
    class WidgetProviderFactory<T> : IClassFactory
```

```
    where T : IWidgetProvider, new()
    {
        public int CreateInstance(IntPtr pUnkOuter, ref Guid riid, out
IntPtr ppvObject)
        {
            ppvObject = IntPtr.Zero;

            if (pUnkOuter != IntPtr.Zero)
            {
                Marshal.ThrowExceptionForHR(CLASS_E_NOAGGREGATION);
            }

            if (riid == typeof(T).GUID || riid ==
Guid.Parse(COM.Guids.IUnknown))
            {
                // Create the instance of the .NET object
                ppvObject =
MarshalInspectable<IWidgetProvider>.FromManaged(new T());
            }
            else
            {
                // The object that ppvObject points to does not support the
                // interface identified by riid.
                Marshal.ThrowExceptionForHR(E_NOINTERFACE);
            }

            return 0;
        }

        int IClassFactory.LockServer(bool fLock)
        {
            return 0;
        }

        private const int CLASS_E_NOAGGREGATION = -2147221232;
        private const int E_NOINTERFACE = -2147467262;

    }
}
```

Create a GUID representing the CLSID for your widget provider

Next, you need to create a GUID representing the CLSID that will be used to identify your widget provider for COM activation. The same value will also be used when packaging your app. Generate a GUID in Visual Studio by going to **Tools->Create GUID**. Select the registry format option and click **Copy** and then paste that into a text file so that you can copy it later.

Register the widget provider class object with OLE

In the Program.cs file for our executable, we will call **CoRegisterClassObject** to register our widget provider with OLE, so that the widget host can interact with it. Replace the contents of Program.cs with the following code. This code imports the **CoRegisterClassObject** function and calls it, passing in the **WidgetProviderFactory** interface we defined in a previous step. Be sure to update the **CLSID_Factory** variable declaration to use the GUID you generated in the previous step.

C#

```
// Program.cs

using System.Runtime.InteropServices;
using ComTypes = System.Runtime.InteropServices.ComTypes;
using Microsoft.Windows.Widgets;
using ExampleWidgetProvider;
using COM;
using System;

[DllImport("kernel32.dll")]
static extern IntPtr GetConsoleWindow();

[DllImport("ole32.dll")]

static extern int CoRegisterClassObject(
            [MarshalAs(UnmanagedType.LPStruct)] Guid rclsid,
            [MarshalAs(UnmanagedType.IUnknown)] object pUnk,
            uint dwClsContext,
            uint flags,
            out uint lpdwRegister);

[DllImport("ole32.dll")] static extern int CoRevokeClassObject(uint
dwRegister);

Console.WriteLine("Registering Widget Provider");
uint cookie;

Guid CLSID_Factory = Guid.Parse("XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX");
CoRegisterClassObject(CLSID_Factory, new
WidgetProviderFactory<WidgetProvider>(), 0x4, 0x1, out cookie);
Console.WriteLine("Registered successfully. Press ENTER to exit.");
Console.ReadLine();

if (GetConsoleWindow() != IntPtr.Zero)
{
    Console.WriteLine("Registered successfully. Press ENTER to exit.");
    Console.ReadLine();
}
else
```

```
{
    // Wait until the manager has disposed of the last widget provider.
    using (var emptyWidgetListEvent =
WidgetProvider.GetEmptyWidgetListEvent())
    {
        emptyWidgetListEvent.WaitOne();
    }

    CoRevokeClassObject(cookie);
}
```

Note that this code example imports the GetConsoleWindow function to determine if the app is running as a console application, the default behavior for this walkthrough. If function returns a valid pointer, we write debug information to the console. Otherwise, the app is running as a Windows app. In that case, we wait for the event that we set in DeleteWidget method when the list of enabled widgets is empty, and the we exit the app. For information on converting the example console app to a Windows app, see Convert your console app to a Windows app.

Package your widget provider app

In the current release, only packaged apps can be registered as widget providers. The following steps will take you through the process of packaging your app and updating the app manifest to register your app with the OS as a widget provider.

Create an MSIX packaging project

In **Solution Explorer**, right-click your solution and select **Add->New Project...**. In the **Add a new project** dialog, select the "Windows Application Packaging Project" template and click **Next**. Set the project name to "ExampleWidgetProviderPackage" and click **Create**. When prompted, set the target version to version 1809 or later and click **OK**. Next, right-click the ExampleWidgetProviderPackage project and select **Add->Project reference**. Select the **ExampleWidgetProvider** project and click OK.

Add Windows App SDK package reference to the packaging project

You need to add a reference to the Windows App SDK nuget package to the MSIX packaging project. In **Solution Explorer**, double-click the ExampleWidgetProviderPackage project to open the ExampleWidgetProviderPackage.wapproj file. Add the following xml inside the **Project** element.

XML

```
<!--ExampleWidgetProviderPackage.wapproj-->
<ItemGroup>
    <PackageReference Include="Microsoft.WindowsAppSDK"
Version="1.2.221116.1">
        <IncludeAssets>build</IncludeAssets>
    </PackageReference>
</ItemGroup>
```

> **Note**
>
> Make sure the **Version** specified in the **PackageReference** element matches the latest stable version you referenced in the previous step.

If the correct version of the Windows App SDK is already installed on the computer and you don't want to bundle the SDK runtime in your package, you can specify the package dependency in the Package.appxmanifest file for the ExampleWidgetProviderPackage project.

XML

```
<!--Package.appxmanifest-->
...
<Dependencies>
...
    <PackageDependency Name="Microsoft.WindowsAppRuntime.1.2-preview2"
MinVersion="2000.638.7.0" Publisher="CN=Microsoft Corporation, O=Microsoft
Corporation, L=Redmond, S=Washington, C=US" />
...
</Dependencies>
...
```

Update the package manifest

In **Solution Explorer** right-click the `Package.appxmanifest` file and select **View Code** to open the manifest xml file. Next you need to add some namespace declarations for the app package extensions we will be using. Add the following namespace definitions to the top-level Package element.

XML

```
<!-- Package.appmanifest -->
<Package
  ...
```

```
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
  xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"
```

Inside the **Application** element, create a new empty element named **Extensions**. Make sure this comes after the closing tag for **uap:VisualElements**.

XML

```
<!-- Package.appxmanifest -->
<Application>
...
    <Extensions>

    </Extensions>
</Application>
```

The first extension we need to add is the ComServer extension. This registers the entry point of the executable with the OS. This extension is the packaged app equivalent of registering a COM server by setting a registry key, and is not specific to widget providers.Add the following **com:Extension** element as a child of the **Extensions** element. Change the GUID in the **Id** attribute of the **com:Class** element to the GUID you generated in a previous step.

XML

```
<!-- Package.appxmanifest -->
<Extensions>
    <com:Extension Category="windows.comServer">
        <com:ComServer>
            <com:ExeServer
Executable="ExampleWidgetProvider\ExampleWidgetProvider.exe"
DisplayName="ExampleWidgetProvider">
                <com:Class Id="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
DisplayName="ExampleWidgetProvider" />
            </com:ExeServer>
        </com:ComServer>
    </com:Extension>
</Extensions>
```

Next, add the extension that registers the app as a widget provider. Paste the uap3:Extension element in the following code snippet, as a child of the **Extensions** element. Be sure to replace the **ClassId** attribute of the **COM** element with the GUID you used in previous steps.

XML

```
<!-- Package.appxmanifest -->
<Extensions>
    ...
    <uap3:Extension Category="windows.appExtension">
        <uap3:AppExtension Name="com.microsoft.windows.widgets"
DisplayName="WidgetTestApp" Id="ContosoWidgetApp" PublicFolder="Public">
            <uap3:Properties>
                <WidgetProvider>
                    <ProviderIcons>
                        <Icon Path="Images\StoreLogo.png" />
                    </ProviderIcons>
                    <Activation>
                        <!-- Apps exports COM interface which implements
IWidgetProvider -->
                        <CreateInstance ClassId="xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx" />
                    </Activation>

                    <TrustedPackageFamilyNames>

<TrustedPackageFamilyName>Microsoft.MicrosoftEdge.Stable_8wekyb3d8bbwe</Trus
tedPackageFamilyName>
                    </TrustedPackageFamilyNames>

                    <Definitions>
                        <Definition Id="Weather_Widget"
                            DisplayName="Weather Widget"
                            Description="Weather Widget Description"
                            AllowMultiple="true">
                            <Capabilities>
                                <Capability>
                                    <Size Name="small" />
                                </Capability>
                                <Capability>
                                    <Size Name="medium" />
                                </Capability>
                                <Capability>
                                    <Size Name="large" />
                                </Capability>
                            </Capabilities>
                            <ThemeResources>
                                <Icons>
                                    <Icon
Path="ProviderAssets\Weather_Icon.png" />
                                </Icons>
                                <Screenshots>
                                    <Screenshot
Path="ProviderAssets\Weather_Screenshot.png" DisplayAltText="For
accessibility" />
                                </Screenshots>
                                <!-- DarkMode and LightMode are optional -->
                                <DarkMode />
                                <LightMode />
                            </ThemeResources>
```

```
                        </Definition>
                        <Definition Id="Counting_Widget"
                                DisplayName="Microsoft Counting Widget"
                                Description="Couting Widget Description">
                            <Capabilities>
                                <Capability>
                                    <Size Name="small" />
                                </Capability>
                            </Capabilities>
                            <ThemeResources>
                                <Icons>
                                    <Icon
Path="ProviderAssets\Counting_Icon.png" />
                                </Icons>
                                <Screenshots>
                                    <Screenshot
Path="ProviderAssets\Counting_Screenshot.png" DisplayAltText="For
accessibility" />
                                </Screenshots>
                                <!-- DarkMode and LightMode are optional -->
                                <DarkMode>

                                </DarkMode>
                                <LightMode />
                            </ThemeResources>
                        </Definition>
                    </Definitions>
                </WidgetProvider>
            </uap3:Properties>
        </uap3:AppExtension>
    </uap3:Extension>
</Extensions>
```

For detailed descriptions and format information for all of these elements, see Widget provider package manifest XML format.

Add icons and other images to your packaging project

In **Solution Explorer**, right-click your **ExampleWidgetProviderPackage** and select **Add->New Folder**. Name this folder ProviderAssets as this is what was used in the `Package.appxmanifest` from the previous step. This is where we will store our **Icons** and **Screenshots** for our widgets. Once you add your desired Icons and Screenshots, make sure the image names match what comes after **Path=ProviderAssets** in your `Package.appxmanifest` or the widgets will not show up in the widget host.

For information about the design requirements for screenshot images and the naming conventions for localized screenshots, see Integrate with the widget picker.

Testing your widget provider

Make sure you have selected the architecture that matches your development machine from the **Solution Platforms** drop-down, for example "x64". In **Solution Explorer**, right-click your solution and select **Build Solution**. Once this is done, right-click your **ExampleWidgetProviderPackage** and select **Deploy**. In the current release, the only supported widget host is the Widgets Board. To see the widgets you will need to open the Widgets Board and select **Add widgets** in the top right. Scroll to the bottom of the available widgets and you should see the mock **Weather Widget** and **Microsoft Counting Widget** that were created in this tutorial. Click on the widgets to pin them to your widgets board and test their functionality.

Debugging your widget provider

After you have pinned your widgets, the Widget Platform will start your widget provider application in order to receive and send relevant information about the widget. To debug the running widget you can either attach a debugger to the running widget provider application or you can set up Visual Studio to automatically start debugging the widget provider process once it's started.

In order to attach to the running process:

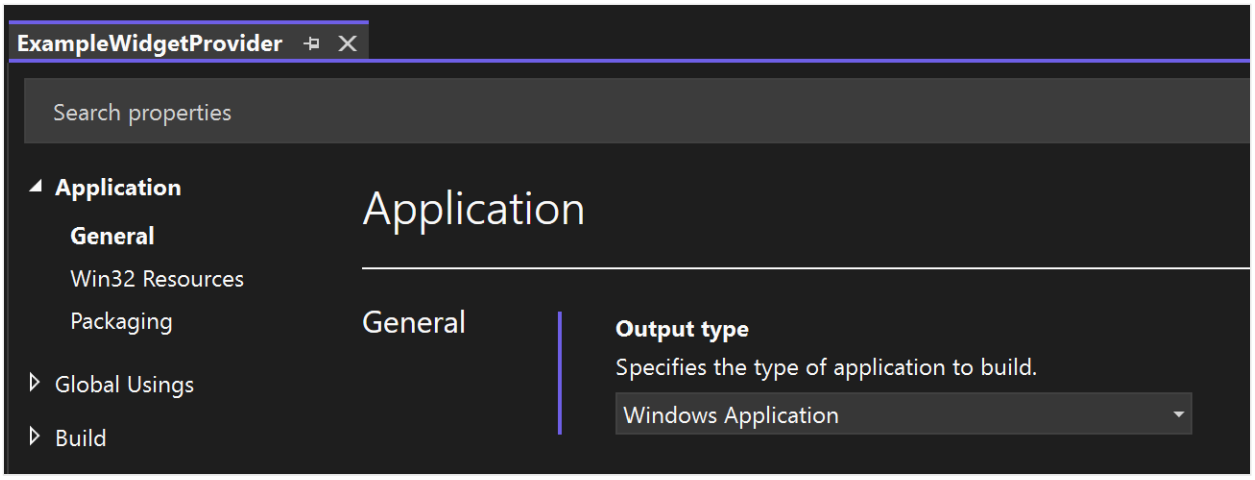
1. In Visual Studio click **Debug -> Attach to process**.
2. Filter the processes and find your desired widget provider application.
3. Attach the debugger.

In order to automatically attach the debugger to the process when it's initially started:

1. In Visual Studio click **Debug -> Other Debug Targets -> Debug Installed App Package**.
2. Filter the packages and find your desired widget provider package.
3. Select it and check the box that says Do not launch, but debug my code when it starts.
4. Click **Attach**.

Convert your console app to a Windows app

To convert the console app created in this walkthrough to a Windows app, right-click the **ExampleWidgetProvider** project in **Solution Explorer** and select **Properties**. Under Application->General change the **Output type** from "Console Application" to "Windows Application".

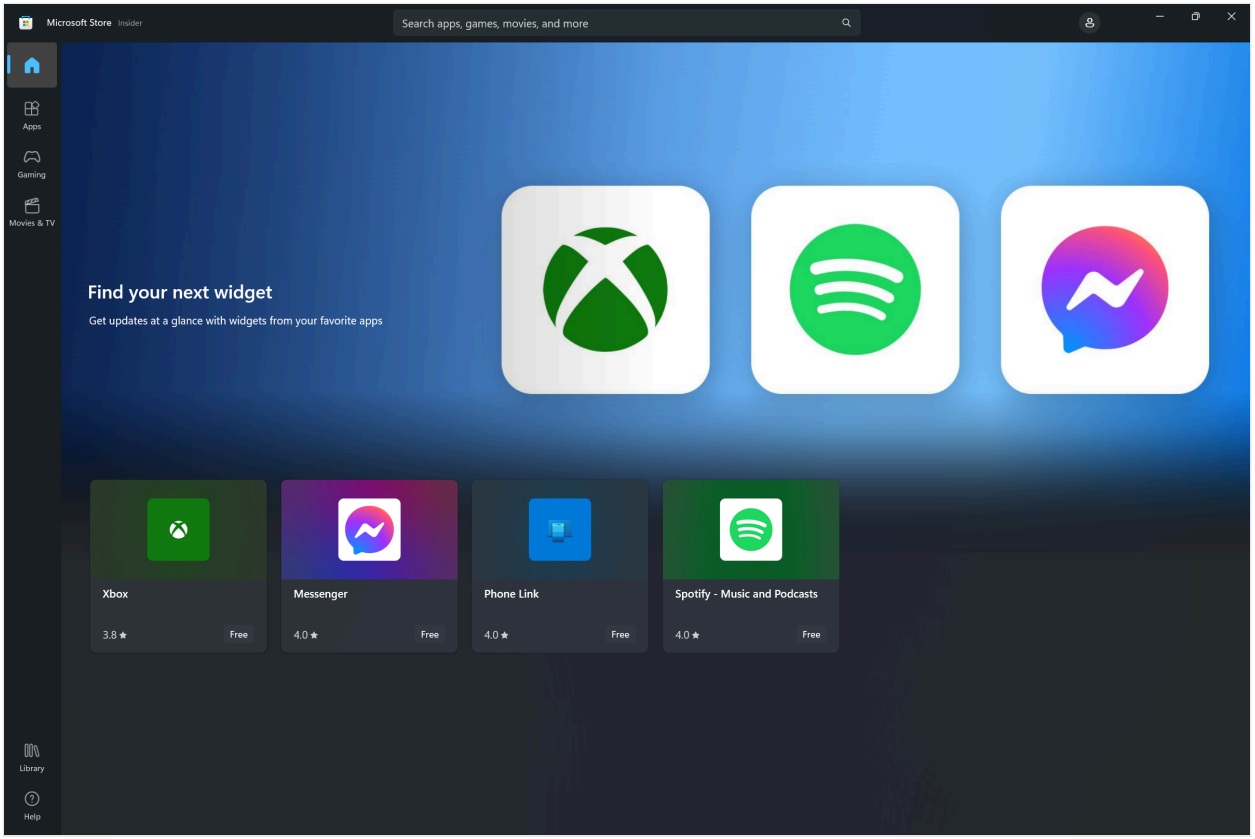


Publishing your widget

After you have developed and tested your widget you can publish your app on the Microsoft Store in order for users to install your widgets on their devices. For step by step guidance for publishing an app, see Publish your app in the Microsoft Store.

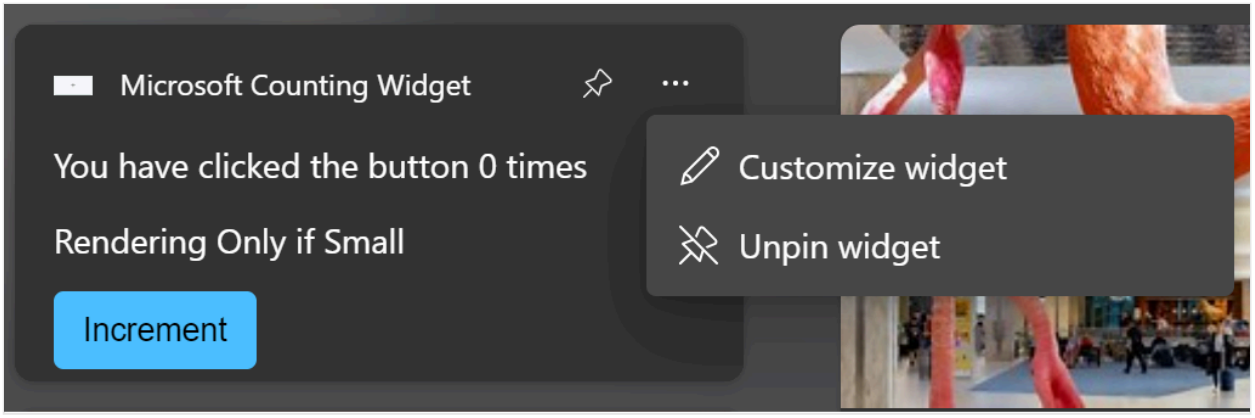
The widgets Store Collection

After your app has been published on the Microsoft Store, you can request for your app to be included in the widgets Store Collection that helps users discover apps that feature Windows Widgets. To submit your request, see Submit your Widget information for addition to the Store Collection .



Implementing widget customization

Starting with Windows App SDK 1.4, widgets can support user customization. When this feature is implemented, a **Customize widget** option is added to the ellipsis menu above the **Unpin widget** option.



The following steps summarize the process for widget customization.

1. In normal operation, the widget provider responds to requests from the widget host with the template and data payloads for the regular widget experience.
2. The user clicks the **Customize widget** button in the ellipsis menu.
3. The widget raises the **OnCustomizationRequested** event on the widget provider to indicate that the user has requested the widget customization experience.
4. The widget provider sets an internal flag to indicate that the widget is in customization mode. While in customization mode, the widget provider sends the JSON templates for the widget customization UI instead of the regular widget UI.
5. While in customization mode, the widget provider receives **OnActionInvoked** events as the user interacts with the customization UI and adjusts its internal configuration and behavior based on the user's actions.
6. When the action associated with the **OnActionInvoked** event is the app-defined "exit customization" action, the widget provider resets its internal flag to indicate that it is no longer in customization mode and resumes sending the visual and data JSON templates for the regular widget experience, reflecting the changes requested during customization. It's possible for the user to close out of the customization experience without clicking the app-defined exit customization action. In this case, the IWidgetProviderAnalytics.OnAnalyticsInfoReported will be raised, and the WidgetAnalyticsInfoReportedArgs.AnalyticsJson will have an *interactionKind* of "exitCustomization".
7. The widget provider persists the customization options to disk or the cloud so that the changes are preserved between invocations of the widget provider.

> ⓘ **Note**

There is a known bug with the Windows Widget Board, for widgets built using the Windows App SDK, that causes the ellipsis menu to become unresponsive after the customization card is shown.

In typical Widget customization scenarios, the user will choose what data is displayed on the widget or adjust visual presentation of the widget. For simplicity, the example in this section will add customization behavior that allows the user to reset the counter of the counting widget implemented in the previous steps.

> ⓘ **Note**
>
> Widget customization is only supported in Windows App SDK 1.4 and later. Make sure you update the references in your project to the latest version of the Nuget package.

Update the package manifest to declare customization support

To let the widget host know that the widget supports customization, add the attribute **IsCustomizable** to the **Definition** element for the widget and set it to true.

XML

```
...
<Definition Id="Counting_Widget"
    DisplayName="Microsoft Counting Widget"
    Description="CONFIG counting widget description"
    IsCustomizable="true">
...
```

Track when a widget is in customization mode

The example in this article uses the helper struct **CompactWidgetInfo** to track the current state of our active widgets. Add the *inCustomization* field, which will be used to track when the widget host is expecting us to send our customization json template rather than the regular widget template.

C#

```
// WidgetProvider.cs
public class CompactWidgetInfo
{
```

```
    public string widgetId { get; set; }
    public string widgetName { get; set; }
    public int customState = 0;
    public bool isActive = false;
    public bool inCustomization = false;
}
```

Implement IWidgetProvider2

The widget customization feature is exposed through the IWidgetProvider2 interface. Update the **WidgetProvider** class definition to implement this interface.

C#

```
// WidgetProvider.cs
internal class WidgetProvider : IWidgetProvider, IWidgetProvider2
```

Add an implementation for the OnCustomizationRequested callback of the **IWidgetProvider2** interface. This method uses the same pattern as the other callbacks we have used. We get the ID for the widget to be customized from the **WidgetContext** and find the **CompactWidgetInfo** helper struct associated with that widget and set the **inCustomization** field to true.

C#

```
// WidgetProvider.cs
public void OnCustomizationRequested(WidgetCustomizationRequestedArgs
customizationInvokedArgs)
{
    var widgetId = customizationInvokedArgs.WidgetContext.Id;
    if (RunningWidgets.ContainsKey(widgetId))
    {
        var localWidgetInfo = RunningWidgets[widgetId];
        localWidgetInfo.inCustomization = true;
        UpdateWidget(localWidgetInfo);
    }
}
```

Now, declare a string variable that defines the JSON template for the widget customization UI. For this example, we have a "Reset counter" button and an "Exit customization" button that will signal our provider to return to regular widget behavior. Place this definition next to the other template definitions.

C#

```
// WidgetProvider.cs
const string countWidgetCustomizationTemplate = @"
{
    ""type"": ""AdaptiveCard"",
    ""actions"" : [
        {
            ""type"": ""Action.Execute"",
            ""title"" : ""Reset counter"",
            ""verb"": ""reset""
            },
            {
            ""type"": ""Action.Execute"",
            ""title"": ""Exit customization"",
            ""verb"": ""exitCustomization""
            }
    ],
    ""$schema"": ""http://adaptivecards.io/schemas/adaptive-card.json"",
    ""version"": ""1.5""
}";
```

Send customization template in UpdateWidget

Next, we'll update our **UpdateWidget** helper method that sends our data and visual JSON templates to the widget host. When we are updating the counting widget, we send either the regular widget template or the customization template depending on the value of the **inCustomization** field. For brevity, code not relevant to customization is omitted in this code snippet.

C#

```
// WidgetProvider.cs
void UpdateWidget(CompactWidgetInfo localWidgetInfo)
{
    ...
    else if (localWidgetInfo.widgetName == "Counting_Widget")
    {
        if (!localWidgetInfo.inCustomization)
        {
            templateJson = countWidgetTemplate.ToString();
        }
        else
        {
            templateJson = countWidgetCustomizationTemplate.ToString();
        }

    }
    ...
    updateOptions.Template = templateJson;
    updateOptions.Data = dataJson;
    // You can store some custom state in the widget service that you will
```

```
be able to query at any time.
    updateOptions.CustomState = localWidgetInfo.customState.ToString();
    WidgetManager.GetDefault().UpdateWidget(updateOptions);
}
```

Respond to customization actions

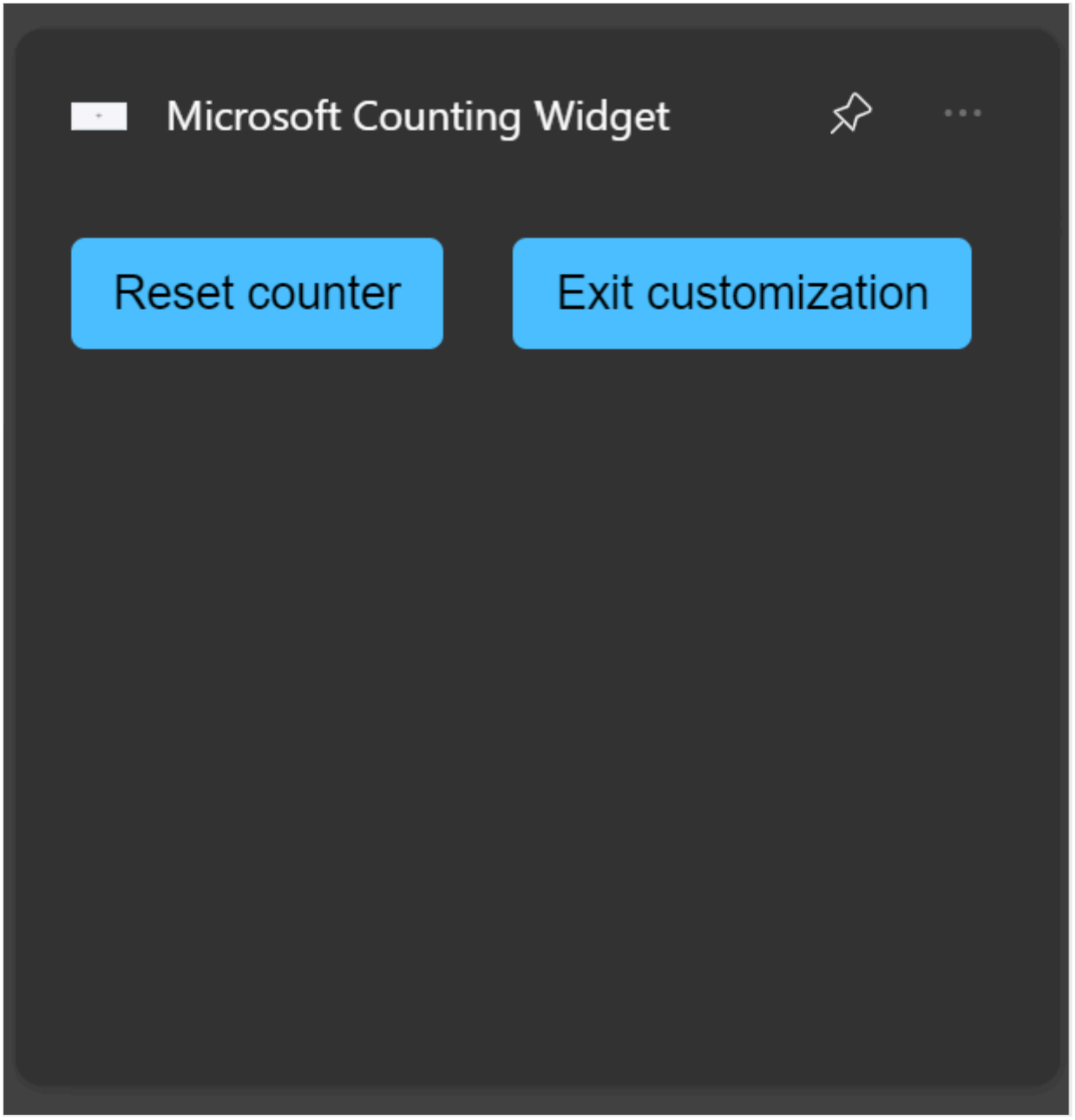
When users interact with inputs in our customization template, it calls the same **OnActionInvoked** handler as when the user interacts with the regular widget experience. To support customization, we look for the verbs "reset" and "exitCustomization" from our customization JSON template. If the action is for the "Reset counter" button, we reset the counter held in the **customState** field of our helper struct to 0. If the action is for the "Exit customization" button, we set the **inCustomization** field to false so that when we call **UpdateWidget**, our helper method will send the regular JSON templates and not the customization template.

C#

```
// WidgetProvider.cs
public void OnActionInvoked(WidgetActionInvokedArgs actionInvokedArgs)
{
    var verb = actionInvokedArgs.Verb;
    if (verb == "inc")
    {
        var widgetId = actionInvokedArgs.WidgetContext.Id;
        // If you need to use some data that was passed in after
        // Action was invoked, you can get it from the args:
        var data = actionInvokedArgs.Data;
        if (RunningWidgets.ContainsKey(widgetId))
        {
            var localWidgetInfo = RunningWidgets[widgetId];
            // Increment the count
            localWidgetInfo.customState++;
            UpdateWidget(localWidgetInfo);
        }
    }
    else if (verb == "reset")
    {
        var widgetId = actionInvokedArgs.WidgetContext.Id;
        var data = actionInvokedArgs.Data;
        if (RunningWidgets.ContainsKey(widgetId))
        {
            var localWidgetInfo = RunningWidgets[widgetId];
            // Reset the count
            localWidgetInfo.customState = 0;
            localWidgetInfo.inCustomization = false;
            UpdateWidget(localWidgetInfo);
        }
    }
```

```
    else if (verb == "exitCustomization")
    {
        var widgetId = actionInvokedArgs.WidgetContext.Id;
        var data = actionInvokedArgs.Data;
        if (RunningWidgets.ContainsKey(widgetId))
        {
            var localWidgetInfo = RunningWidgets[widgetId];
            // Stop sending the customization template
            localWidgetInfo.inCustomization = false;
            UpdateWidget(localWidgetInfo);
        }
    }
}
```

Now, when you deploy your widget, you should see the **Customize widget** button in the ellipses menu. Clicking on the customize button will display your customization template.

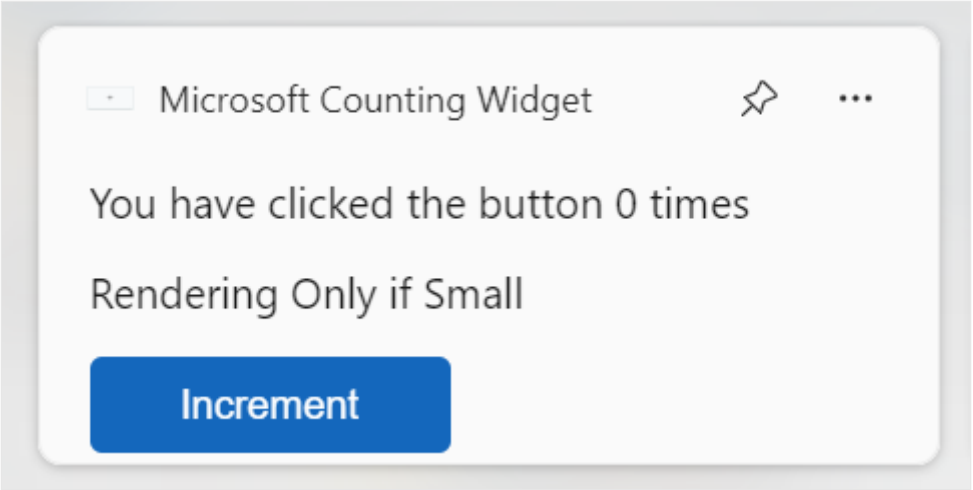


Click the **Reset counter** button to reset the counter to 0. Click the **Exit customization** button to return to your widget's regular behavior.

Implement a widget provider in a win32 app (C++/WinRT)

Article • 10/02/2023

This article walks you through creating a simple widget provider that implements the IWidgetProvider interface. The methods of this interface are invoked by the widget host to request the data that defines a widget or to let the widget provider respond to a user action on a widget. Widget providers can support a single widget or multiple widgets. In this example, we will define two different widgets. One widget is a mock weather widget that illustrates some of the formatting options provided by the Adaptive Cards framework. The second widget will demonstrate user actions and the custom widget state feature by maintaining a counter that is incremented whenever the user clicks on a button displayed on the widget.



This sample code in this article is adapted from the Windows App SDK Widgets Sample. To implement a widget provider using C#, see Implement a widget provider in a win32 app (C#).

Prerequisites

- Your device must have developer mode enabled. For more information see Enable your device for development.
- Visual Studio 2022 or later with the **Universal Windows Platform development** workload. Make sure to add the component for C++ (v143) from the optional dropdown.

Create a new C++/WinRT win32 console app

In Visual Studio, create a new project. In the **Create a new project** dialog, set the language filter to "C++" and the platform filter to Windows, then select the Windows Console Application (C++/WinRT) project template. Name the new project "ExampleWidgetProvider". When prompted, set the target Windows version for the app to version 1809 or later.

Add references to the Windows App SDK and Windows Implementation Library NuGet packages

This sample uses the latest stable Windows App SDK NuGet package. In **Solution Explorer**, right-click **References** and select **Manage NuGet packages...**. In the NuGet package manager, select the **Browse** tab and search for "Microsoft.WindowsAppSDK". Select the latest stable version in the **Version** drop-down and then click **Install**.

This sample also uses the Windows Implementation Library NuGet package. In **Solution Explorer**, right-click **References** and select **Manage NuGet packages...**. In the NuGet package manager, select the **Browse** tab and search for "Microsoft.Windows.ImplementationLibrary". Select the latest version in the **Version** drop-down and then click **Install**.

In the precompiled header file, pch.h, add the following include directives.

C++

```
//pch.h
#pragma once
#include <wil/cppwinrt.h>
#include <wil/resource.h>
...
#include <winrt/Microsoft.Windows.Widgets.Providers.h>
```

> **Note**
>
> You must include the wil/cppwinrt.h header first, before any WinRT headers.

In order to handle shutting down the widget provider app correctly, we need to a custom implementation of **winrt::get_module_lock**. We pre-declare the **SignalLocalServerShutdown** method which will be defined in our main.cpp file and will set an event that signals the app to exit. Add the following code to your pch.h file, just below the `#pragma once` directive, before the other includes.

C++

```
//pch.h
#include <stdint.h>
#include <combaseapi.h>

// In .exe local servers the class object must not contribute to the module
ref count, and use
// winrt::no_module_lock, the other objects must and this is the hook into
the C++ WinRT ref counting system
// that enables this.
void SignalLocalServerShutdown();

namespace winrt
{
    inline auto get_module_lock() noexcept
    {
        struct service_lock
        {
            uint32_t operator++() noexcept
```

```
            {
                return ::CoAddRefServerProcess();
            }

            uint32_t operator--() noexcept
            {
                const auto ref = ::CoReleaseServerProcess();

                if (ref == 0)
                {
                    SignalLocalServerShutdown();
                }
                return ref;
            }
        };

        return service_lock{};
    }
}


#define WINRT_CUSTOM_MODULE_LOCK
```

Add a WidgetProvider class to handle widget operations

In Visual Studio, right-click the `ExampleWidgetProvider` project in **Solution Explorer** and select **Add->Class**. In the **Add class** dialog, name the class "WidgetProvider" and click **Add**.

Declare a class that implements the IWidgetProvider interface

The **IWidgetProvider** interface defines methods that the widget host will invoke to initiate operations with the widget provider. Replace the empty class definition in the WidgetProvider.h file with the following code. This code declares a structure that implements the **IWidgetProvider** interface and declares prototypes for the interface methods.

C++

```
// WidgetProvider.h
struct WidgetProvider : winrt::implements<WidgetProvider,
winrt::Microsoft::Windows::Widgets::Providers::IWidgetProvider>
{
    WidgetProvider();
```

```
    /* IWidgetProvider required functions that need to be implemented */
    void
CreateWidget(winrt::Microsoft::Windows::Widgets::Providers::WidgetContext
widgetContext);
    void DeleteWidget(winrt::hstring const& widgetId, winrt::hstring const&
customState);
    void
OnActionInvoked(winrt::Microsoft::Windows::Widgets::Providers::WidgetActionI
nvokedArgs actionInvokedArgs);
    void
OnWidgetContextChanged(winrt::Microsoft::Windows::Widgets::Providers::Widget
ContextChangedArgs contextChangedArgs);
    void
Activate(winrt::Microsoft::Windows::Widgets::Providers::WidgetContext
widgetContext);
    void Deactivate(winrt::hstring widgetId);
    /* IWidgetProvider required functions that need to be implemented */


};
```

Also, add a private method, **UpdateWidget**, which is a helper method that will send updates from our provider to the widget host.

C++

```
// WidgetProvider.h
private:

void UpdateWidget(CompactWidgetInfo const& localWidgetInfo);
```

Prepare to track enabled widgets

A widget provider can support a single widget or multiple widgets. Whenever the widget host initiates an operation with the widget provider, it passes an ID to identify the widget associated with the operation. Each widget also has an associated name and a state value that can be used to store custom data. For this example, we'll declare a simple helper structure to store the ID, name, and data for each pinned widget. Widgets also can be in an active state, which is discussed in the Activate and Deactivate section below, and we will track this state for each widget with a boolean value. Add the following definition to the WidgetProvider.h file, above the **WidgetProvider** struct declaration.

C++

```
// WidgetProvider.h
struct CompactWidgetInfo
{
    winrt::hstring widgetId;
    winrt::hstring widgetName;
    int customState = 0;
    bool isActive = false;
};
```

Inside the **WidgetProvider** declaration in WidgetProvider.h, add a member for the map that will maintain the list of enabled widgets, using the widget ID as the key for each entry.

C++

```
// WidgetProvider.h
#include <unordered_map>
struct WidgetProvider : winrt::implements<WidgetProvider,
winrt::Microsoft::Windows::Widgets::Providers::IWidgetProvider>
{
...
    private:
        ...
        static std::unordered_map<winrt::hstring, CompactWidgetInfo>
RunningWidgets;
```

Declare widget template JSON strings

This example will declare some static strings to define the JSON templates for each widget. For convenience, these templates are stored in the local variables declared outside of the **WidgetProvider** class definition. If you need a general storage for the templates - they can be included as part of the application package: Accessing Package Files. For information on creating the widget template JSON document, see Create a widget template with the Adaptive Card Designer.

In the latest release, apps that implement Windows widgets can customize the header that is displayed for their widget in the Widgets Board, overriding the default presentation. For more information, see Customize the widget header area.

> **Note**
>
> Starting with Windows Build [TBD - Build number], apps that implement Windows widgets can choose to populate the widget content with HTML served from a

specified URL instead of supplying content in the Adaptive Card schema format in the JSON payload passed from the provider to the Widgets Board. Widget providers must still provide an Adaptive Card JSON payload, so the implementation steps in this walkthrough are applicable to web widgets. For more information, see Web widget providers.

C++

```
// WidgetProvider.h
const std::string weatherWidgetTemplate = R"(
{
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "type": "AdaptiveCard",
    "version": "1.0",
    "speak": "<s>The forecast for Seattle January 20 is mostly clear with a
High of 51 degrees and Low of 40 degrees</s>",
    "backgroundImage":
"https://messagecardplayground.azurewebsites.net/assets/Mostly%20Cloudy-
Background.jpg",
    "body": [
        {
            "type": "TextBlock",
            "text": "Redmond, WA",
            "size": "large",
            "isSubtle": true,
            "wrap": true
        },
        {
            "type": "TextBlock",
            "text": "Mon, Nov 4, 2019 6:21 PM",
            "spacing": "none",
            "wrap": true
        },
        {
            "type": "ColumnSet",
            "columns": [
                {
                    "type": "Column",
                    "width": "auto",
                    "items": [
                        {
                            "type": "Image",
                            "url":
"https://messagecardplayground.azurewebsites.net/assets/Mostly%20Cloudy-
Square.png",
                            "size": "small",
                            "altText": "Mostly cloudy weather"
                        }
                    ]
                },
                {
                    "type": "Column",
```

```
                    "width": "auto",
                    "items": [
                        {
                            "type": "TextBlock",
                            "text": "46",
                            "size": "extraLarge",
                            "spacing": "none",
                            "wrap": true
                        }
                    ]
                },
                {
                    "type": "Column",
                    "width": "stretch",
                    "items": [
                        {
                            "type": "TextBlock",
                            "text": "°F",
                            "weight": "bolder",
                            "spacing": "small",
                            "wrap": true
                        }
                    ]
                },
                {
                    "type": "Column",
                    "width": "stretch",
                    "items": [
                        {
                            "type": "TextBlock",
                            "text": "Hi 50",
                            "horizontalAlignment": "left",
                            "wrap": true
                        },
                        {
                            "type": "TextBlock",
                            "text": "Lo 41",
                            "horizontalAlignment": "left",
                            "spacing": "none",
                            "wrap": true
                        }
                    ]
                }
            ]
        }
    ]
})";

const std::string countWidgetTemplate = R"(
{
    "type": "AdaptiveCard",
    "body": [
        {
            "type": "TextBlock",
            "text": "You have clicked the button ${count} times"
```

```
        },
        {
             "text":"Rendering Only if Medium",
             "type":"TextBlock",
             "$when":"${$host.widgetSize==\"medium\"}"
        },
        {
             "text":"Rendering Only if Small",
             "type":"TextBlock",
             "$when":"${$host.widgetSize==\"small\"}"
        },
        {
         "text":"Rendering Only if Large",
         "type":"TextBlock",
         "$when":"${$host.widgetSize==\"large\"}"
        }
    ],
    "actions": [
        {
            "type": "Action.Execute",
            "title": "Increment",
            "verb": "inc"
        }
    ],
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "version": "1.5"
})";
```

Implement the IWidgetProvider methods

In the next few sections, we'll implement the methods of the **IWidgetProvider** interface. The helper method **UpdateWidget** that is called in several of these method implementations will be shown later in this article. Before diving into the interface methods, add the following lines to `WidgetProvider.cpp`, after the include directives, to pull the widget provider APIs into the **winrt** namespace and allow access to the map we declared in the previous step.

> ⓘ **Note**
>
> Objects passed into the callback methods of the **IWidgetProvider** interface are only guaranteed to be valid within the callback. You should not store references to these objects because their behavior outside of the context of the callback is undefined.

C++

```
// WidgetProvider.cpp
namespace winrt
```

```
{
    using namespace Microsoft::Windows::Widgets::Providers;
}

std::unordered_map<winrt::hstring, CompactWidgetInfo>
WidgetProvider::RunningWidgets{};
```

CreateWidget

The widget host calls CreateWidget when the user has pinned one of your app's widgets in the widget host. First, this method gets the ID and name of the associated widget and adds a new instance of our helper structure, **CompactWidgetInfo**, to the collection of enabled widgets. Next, we send the initial template and data for the widget, which is encapsulated in the **UpdateWidget** helper method.

C++

```
// WidgetProvider.cpp
void WidgetProvider::CreateWidget(winrt::WidgetContext widgetContext)
{
    auto widgetId = widgetContext.Id();
    auto widgetName = widgetContext.DefinitionId();
    CompactWidgetInfo runningWidgetInfo{ widgetId, widgetName };
    RunningWidgets[widgetId] = runningWidgetInfo;

    // Update the widget
    UpdateWidget(runningWidgetInfo);
}
```

DeleteWidget

The widget host calls DeleteWidget when the user has unpinned one of your app's widgets from the widget host. When this occurs, we will remove the associated widget from our list of enabled widgets so that we don't send any further updates for that widget.

C++

```
// WidgetProvider.cpp
void WidgetProvider::DeleteWidget(winrt::hstring const& widgetId,
winrt::hstring const& customState)
{
    RunningWidgets.erase(widgetId);
}
```

OnActionInvoked

The widget host calls OnActionInvoked when the user interacts with an action you defined in your widget template. For the counter widget used in this example, an action was declared with a **verb** value of "inc" in the JSON template for the widget. The widget provider code will use this **verb** value to determine what action to take in response to the user interaction.

JSON

```
...
    "actions": [
        {
            "type": "Action.Execute",
            "title": "Increment",
            "verb": "inc"
        }
    ],
...
```

In the **OnActionInvoked** method, get the verb value by checking the **Verb** property of the WidgetActionInvokedArgs passed into the method. If the verb is "inc", then we know we are going to increment the count in the custom state for the widget. From the **WidgetActionInvokedArgs**, get the WidgetContext object and then the **WidgetId** to get the ID for the widget that is being updated. Find the entry in our enabled widgets map with the specified ID and then update the custom state value that is used to store the number of increments. Finally, update the widget content with the new value with the **UpdateWidget** helper function.

C++

```
// WidgetProvider.cpp
void WidgetProvider::OnActionInvoked(winrt::WidgetActionInvokedArgs
actionInvokedArgs)
{
    auto verb = actionInvokedArgs.Verb();
    if (verb == L"inc")
    {
        auto widgetId = actionInvokedArgs.WidgetContext().Id();
        // If you need to use some data that was passed in after
        // Action was invoked, you can get it from the args:
        auto data = actionInvokedArgs.Data();
        if (const auto iter = RunningWidgets.find(widgetId); iter !=
RunningWidgets.end())
        {
            auto& localWidgetInfo = iter->second;
            // Increment the count
            localWidgetInfo.customState++;
```

```
            UpdateWidget(localWidgetInfo);
        }
    }
}
```

For information about the **Action.Execute** syntax for Adaptive Cards, see Action.Execute. For guidance about designing interaction for widgets, see Widget interaction design guidance

OnWidgetContextChanged

In the current release, OnWidgetContextChanged is only called when the user changes the size of a pinned widget. You can choose to return a different JSON template/data to the widget host depending on what size is requested. You can also design the template JSON to support all the available sizes using conditional rendering based on the value of **host.widgetSize**. If you don't need to send a new template or data to account for the size change, you can use the **OnWidgetContextChanged** for telemetry purposes.

C++

```
// WidgetProvider.cpp
void WidgetProvider::OnWidgetContextChanged(winrt::WidgetContextChangedArgs
contextChangedArgs)
{
    auto widgetContext = contextChangedArgs.WidgetContext();
    auto widgetId = widgetContext.Id();
    auto widgetSize = widgetContext.Size();
    if (const auto iter = RunningWidgets.find(widgetId); iter !=
RunningWidgets.end())
    {
        auto localWidgetInfo = iter->second;

        UpdateWidget(localWidgetInfo);
    }
}
```

Activate and Deactivate

The Activate method is called to notify the widget provider that the widget host is currently interested in receiving updated content from the provider. For example, it could mean that the user is currently actively viewing the widget host. The Deactivate method is called to notify the widget provider that the widget host is no longer requesting content updates. These two methods define a window in which the widget host is most interested in showing the most up-to-date content. Widget providers can

send updates to the widget at any time, such as in response to a push notification, but as with any background task, it's important to balance providing up-to-date content with resource concerns like battery life.

Activate and **Deactivate** are called on a per-widget basis. This example tracks the active status of each widget in the **CompactWidgetInfo** helper struct. In the **Activate** method, we call the **UpdateWidget** helper method to update our widget. Note that the time window between **Activate** and **Deactivate** may be small, so it's recommended that you try to make your widget update code path as quick as possible.

C++

```
void
WidgetProvider::Activate(winrt::Microsoft::Windows::Widgets::Providers::Widg
etContext widgetContext)
{
    auto widgetId = widgetContext.Id();

    if (const auto iter = RunningWidgets.find(widgetId); iter !=
RunningWidgets.end())
    {
        auto& localWidgetInfo = iter->second;
        localWidgetInfo.isActive = true;

        UpdateWidget(localWidgetInfo);
    }
}

void WidgetProvider::Deactivate(winrt::hstring widgetId)
{
    if (const auto iter = RunningWidgets.find(widgetId); iter !=
RunningWidgets.end())
    {
        auto& localWidgetInfo = iter->second;
        localWidgetInfo.isActive = false;
    }
}
```

Update a widget

Define the **UpdateWidget** helper method to update an enabled widget. In this example, we check the name of the widget in the **CompactWidgetInfo** helper struct passed into the method, and then set the appropriate template and data JSON based on which widget is being updated. A **WidgetUpdateRequestOptions** is initialized with the template, data, and custom state for the widget being updated. Call WidgetManager::GetDefault to get an instance of the WidgetManager class and then call UpdateWidget to send the updated widget data to the widget host.

C++

```
// WidgetProvider.cpp
void WidgetProvider::UpdateWidget(CompactWidgetInfo const& localWidgetInfo)
{
    winrt::WidgetUpdateRequestOptions updateOptions{
localWidgetInfo.widgetId };

    winrt::hstring templateJson;
    if (localWidgetInfo.widgetName == L"Weather_Widget")
    {
        templateJson = winrt::to_hstring(weatherWidgetTemplate);
    }
    else if (localWidgetInfo.widgetName == L"Counting_Widget")
    {
        templateJson = winrt::to_hstring(countWidgetTemplate);
    }

    winrt::hstring dataJson;
    if (localWidgetInfo.widgetName == L"Weather_Widget")
    {
        dataJson = L"{}";
    }
    else if (localWidgetInfo.widgetName == L"Counting_Widget")
    {
        dataJson = L"{ \"count\": " +
winrt::to_hstring(localWidgetInfo.customState) + L" }";
    }

    updateOptions.Template(templateJson);
    updateOptions.Data(dataJson);
    // You can store some custom state in the widget service that you will
be able to query at any time.

updateOptions.CustomState(winrt::to_hstring(localWidgetInfo.customState));
    winrt::WidgetManager::GetDefault().UpdateWidget(updateOptions);
}
```

Initialize the list of enabled widgets on startup

When our widget provider is first initialized, it's a good idea to ask **WidgetManager** if there are any running widgets that our provider is currently serving. It will help to recover the app to the previous state in case of the computer restart or the provider crash. Call **WidgetManager::GetDefault** to get the default widget manager instance for the app. Then call GetWidgetInfos, which returns an array of WidgetInfo objects. Copy the widget IDs, names, and custom state into the helper struct **CompactWidgetInfo** and save it to the **RunningWidgets** member variable. Paste the following code into the constructor for the **WidgetProvider** class.

C++

```
// WidgetProvider.cpp
WidgetProvider::WidgetProvider()
{
    auto runningWidgets =
winrt::WidgetManager::GetDefault().GetWidgetInfos();
    for (auto widgetInfo : runningWidgets )
    {
        auto widgetContext = widgetInfo.WidgetContext();
        auto widgetId = widgetContext.Id();
        auto widgetName = widgetContext.DefinitionId();
        auto customState = widgetInfo.CustomState();
        if (RunningWidgets.find(widgetId) == RunningWidgets.end())
        {
            CompactWidgetInfo runningWidgetInfo{ widgetId, widgetName };
            try
            {
                // If we had any save state (in this case we might have some
state saved for Counting widget)
                // convert string to required type if needed.
                int count = std::stoi(winrt::to_string(customState));
                runningWidgetInfo.customState = count;
            }
            catch (...)
            {

            }
            RunningWidgets[widgetId] = runningWidgetInfo;
        }
    }
}
```

Register a class factory that will instantiate WidgetProvider on request

Add the header that defines the **WidgetProvider** class to the includes at the top of your app's `main.cpp` file. We will also be including **mutex** here.

C++

```
// main.cpp
...
#include "WidgetProvider.h"
#include <mutex>
```

Declare the event that will trigger our app to exit and the **SignalLocalServerShutdown** function that will set the event. Paste the following code in main.cpp.

C++

```
// main.cpp
wil::unique_event g_shudownEvent(wil::EventOptions::None);

void SignalLocalServerShutdown()
{
    g_shudownEvent.SetEvent();
}
```

Next, you will need to create a CLSID that will be used to identify your widget provider for COM activation. Generate a GUID in Visual Studio by going to **Tools->Create GUID**. Select the option "static const GUID =" and click **Copy** and then paste that into `main.cpp`. Update the GUID definition with the following C++/WinRT syntax, setting the GUID variable name widget_provider_clsid. Leave the commented version of the GUID because you will need this format later, when packaging your app.

C++

```
// main.cpp
...
// {80F4CB41-5758-4493-9180-4FB8D480E3F5}
static constexpr GUID widget_provider_clsid
{
    0x80f4cb41, 0x5758, 0x4493, { 0x91, 0x80, 0x4f, 0xb8, 0xd4, 0x80, 0xe3,
0xf5 }
};
```

Add the following class factory definition to `main.cpp`. This is mostly boilerplate code that is not specific to widget provider implementations. Note that **CoWaitForMultipleObjects** waits for our shutdown event to be triggered before the app exits.

C++

```
// main.cpp
template <typename T>
struct SingletonClassFactory : winrt::implements<SingletonClassFactory<T>,
IClassFactory>
{
    STDMETHODIMP CreateInstance(
        ::IUnknown* outer,
        GUID const& iid,
        void** result) noexcept final
    {
        *result = nullptr;

        std::unique_lock lock(mutex);
```

```
        if (outer)
        {
            return CLASS_E_NOAGGREGATION;
        }

        if (!instance)
        {
            instance = winrt::make<WidgetProvider>();
        }

        return instance.as(iid, result);
    }

    STDMETHODIMP LockServer(BOOL) noexcept final
    {
        return S_OK;
    }

private:
    T instance{ nullptr };
    std::mutex mutex;
};

int main()
{
    winrt::init_apartment();
    wil::unique_com_class_object_cookie widgetProviderFactory;
    auto factory =
winrt::make<SingletonClassFactory<winrt::Microsoft::Windows::Widgets::Provid
ers::IWidgetProvider>>();

    winrt::check_hresult(CoRegisterClassObject(
        widget_provider_clsid,
        factory.get(),
        CLSCTX_LOCAL_SERVER,
        REGCLS_MULTIPLEUSE,
        widgetProviderFactory.put()));

    DWORD index{};
    HANDLE events[] = { g_shudownEvent.get() };
    winrt::check_hresult(CoWaitForMultipleObjects(CWMO_DISPATCH_CALLS |
CWMO_DISPATCH_WINDOW_MESSAGES,
        INFINITE,
        static_cast<ULONG>(std::size(events)), events, &index));

    return 0;
}
```

Package your widget provider app

In the current release, only packaged apps can be registered as widget providers. The following steps will take you through the process of packaging your app and updating the app manifest to register your app with the OS as a widget provider.

Create an MSIX packaging project

In **Solution Explorer**, right-click your solution and select **Add->New Project...**. In the **Add a new project** dialog, select the "Windows Application Packaging Project" template and click **Next**. Set the project name to "ExampleWidgetProviderPackage" and click **Create**. When prompted, set the target version to version 1809 or later and click **OK**. Next, right-click the ExampleWidgetProviderPackage project and select **Add->Project reference**. Select the **ExampleWidgetProvider** project and click OK.

Add Windows App SDK package reference to the packaging project

You need to add a reference to the Windows App SDK nuget package to the MSIX packaging project. In **Solution Explorer**, double-click the ExampleWidgetProviderPackage project to open the ExampleWidgetProviderPackage.wapproj file. Add the following xml inside the **Project** element.

XML

```
<!--ExampleWidgetProviderPackage.wapproj-->
<ItemGroup>
    <PackageReference Include="Microsoft.WindowsAppSDK"
Version="1.2.221116.1">
        <IncludeAssets>build</IncludeAssets>
    </PackageReference>
</ItemGroup>
```

> **Note**
>
> Make sure the **Version** specified in the **PackageReference** element matches the latest stable version you referenced in the previous step.

If the correct version of the Windows App SDK is already installed on the computer and you don't want to bundle the SDK runtime in your package, you can specify the package dependency in the Package.appxmanifest file for the ExampleWidgetProviderPackage project.

XML

```
<!--Package.appxmanifest-->
...
<Dependencies>
...
    <PackageDependency Name="Microsoft.WindowsAppRuntime.1.2-preview2"
MinVersion="2000.638.7.0" Publisher="CN=Microsoft Corporation, O=Microsoft
Corporation, L=Redmond, S=Washington, C=US" />
...
</Dependencies>
...
```

Update the package manifest

In **Solution Explorer** right-click the `Package.appxmanifest` file and select **View Code** to open the manifest xml file. Next you need to add some namespace declarations for the app package extensions we will be using. Add the following namespace definitions to the top-level Package element.

XML

```
<!-- Package.appmanifest -->
<Package
  ...
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
  xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"
```

Inside the **Application** element, create a new empty element named **Extensions**. Make sure this comes after the closing tag for **uap:VisualElements**.

XML

```
<!-- Package.appxmanifest -->
<Application>
...
    <Extensions>

    </Extensions>
</Application>
```

The first extension we need to add is the ComServer extension. This registers the entry point of the executable with the OS. This extension is the packaged app equivalent of registering a COM server by setting a registry key, and is not specific to widget providers.Add the following **com:Extension** element as a child of the **Extensions**

element. Change the GUID in the **Id** attribute of the **com:Class** element to the GUID you generated in a previous step.

XML

```
<!-- Package.appxmanifest -->
<Extensions>
    <com:Extension Category="windows.comServer">
        <com:ComServer>
            <com:ExeServer
Executable="ExampleWidgetProvider\ExampleWidgetProvider.exe"
DisplayName="ExampleWidgetProvider">
                <com:Class Id="xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
DisplayName="ExampleWidgetProvider" />
            </com:ExeServer>
        </com:ComServer>
    </com:Extension>
</Extensions>
```

Next, add the extension that registers the app as a widget provider. Paste the uap3:Extension element in the following code snippet, as a child of the **Extensions** element. Be sure to replace the **ClassId** attribute of the **COM** element with the GUID you used in previous steps.

XML

```
<!-- Package.appxmanifest -->
<Extensions>
    ...
    <uap3:Extension Category="windows.appExtension">
        <uap3:AppExtension Name="com.microsoft.windows.widgets"
DisplayName="WidgetTestApp" Id="ContosoWidgetApp" PublicFolder="Public">
            <uap3:Properties>
                <WidgetProvider>
                    <ProviderIcons>
                        <Icon Path="Images\StoreLogo.png" />
                    </ProviderIcons>
                    <Activation>
                        <!-- Apps exports COM interface which implements
IWidgetProvider -->
                        <CreateInstance ClassId="xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx" />
                    </Activation>

                    <TrustedPackageFamilyNames>

<TrustedPackageFamilyName>Microsoft.MicrosoftEdge.Stable_8wekyb3d8bbwe</Trus
tedPackageFamilyName>
                    </TrustedPackageFamilyNames>

                    <Definitions>
```

```
                        <Definition Id="Weather_Widget"
                            DisplayName="Weather Widget"
                            Description="Weather Widget Description"
                            AllowMultiple="true">
                            <Capabilities>
                                <Capability>
                                    <Size Name="small" />
                                </Capability>
                                <Capability>
                                    <Size Name="medium" />
                                </Capability>
                                <Capability>
                                    <Size Name="large" />
                                </Capability>
                            </Capabilities>
                            <ThemeResources>
                                <Icons>
                                    <Icon
Path="ProviderAssets\Weather_Icon.png" />
                                </Icons>
                                <Screenshots>
                                    <Screenshot
Path="ProviderAssets\Weather_Screenshot.png" DisplayAltText="For
accessibility" />
                                </Screenshots>
                                <!-- DarkMode and LightMode are optional -->
                                <DarkMode />
                                <LightMode />
                            </ThemeResources>
                        </Definition>
                        <Definition Id="Counting_Widget"
                                DisplayName="Microsoft Counting Widget"
                                Description="Couting Widget Description">
                            <Capabilities>
                                <Capability>
                                    <Size Name="small" />
                                </Capability>
                            </Capabilities>
                            <ThemeResources>
                                <Icons>
                                    <Icon
Path="ProviderAssets\Counting_Icon.png" />
                                </Icons>
                                <Screenshots>
                                    <Screenshot
Path="ProviderAssets\Counting_Screenshot.png" DisplayAltText="For
accessibility" />
                                </Screenshots>
                                <!-- DarkMode and LightMode are optional -->
                                <DarkMode>

                                </DarkMode>
                                <LightMode />
                            </ThemeResources>
                        </Definition>
```

```
                    </Definitions>
                </WidgetProvider>
            </uap3:Properties>
        </uap3:AppExtension>
    </uap3:Extension>
</Extensions>
```

For detailed descriptions and format information for all of these elements, see Widget provider package manifest XML format.

Add icons and other images to your packaging project

In **Solution Explorer**, right-click your **ExampleWidgetProviderPackage** and select **Add->New Folder**. Name this folder ProviderAssets as this is what was used in the `Package.appxmanifest` from the previous step. This is where we will store our **Icons** and **Screenshots** for our widgets. Once you add your desired Icons and Screenshots, make sure the image names match what comes after **Path=ProviderAssets** in your `Package.appxmanifest` or the widgets will not show up in the widget host.

For information about the design requirements for screenshot images and the naming conventions for localized screenshots, see Integrate with the widget picker.

Testing your widget provider

Make sure you have selected the architecture that matches your development machine from the **Solution Platforms** drop-down, for example "x64". In **Solution Explorer**, right-click your solution and select **Build Solution**. Once this is done, right-click your **ExampleWidgetProviderPackage** and select **Deploy**. In the current release, the only supported widget host is the Widgets Board. To see the widgets you will need to open the Widgets Board and select **Add widgets** in the top right. Scroll to the bottom of the available widgets and you should see the mock **Weather Widget** and **Microsoft Counting Widget** that were created in this tutorial. Click on the widgets to pin them to your widgets board and test their functionality.

Debugging your widget provider

After you have pinned your widgets, the Widget Platform will start your widget provider application in order to receive and send relevant information about the widget. To debug the running widget you can either attach a debugger to the running widget

provider application or you can set up Visual Studio to automatically start debugging the widget provider process once it's started.

In order to attach to the running process:

1. In Visual Studio click **Debug -> Attach to process**.
2. Filter the processes and find your desired widget provider application.
3. Attach the debugger.

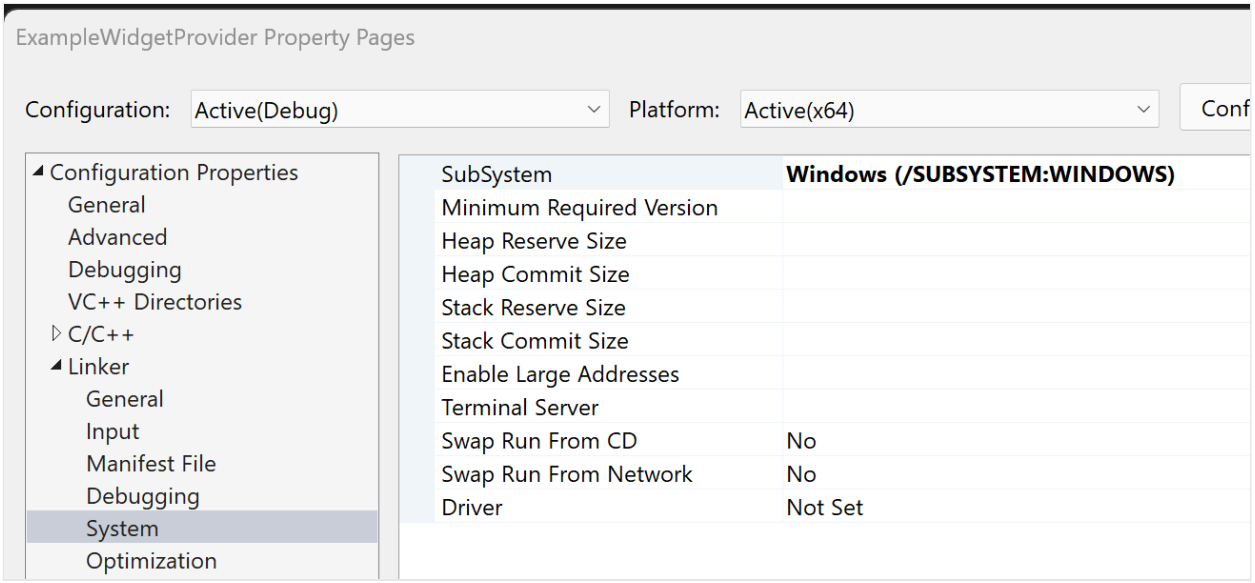
In order to automatically attach the debugger to the process when it's initially started:

1. In Visual Studio click **Debug -> Other Debug Targets -> Debug Installed App Package**.
2. Filter the packages and find your desired widget provider package.
3. Select it and check the box that says Do not launch, but debug my code when it starts.
4. Click **Attach**.

Convert your console app to a Windows app

To convert the console app created in this walkthrough to a Windows app:

1. Right-click on the ExampleWidgetProvider project in **Solution Explorer** and select **Properties**. Navigate to **Linker -> System** and change **SubSystem** from "Console" to "Windows". This can also be done by adding <SubSystem>Windows</SubSystem> to the <Link>..</Link> section of the .vcxproj.
2. In main.cpp, change `int main()` to `int WINAPI wWinMain(_In_ HINSTANCE /*hInstance*/, _In_opt_ HINSTANCE /*hPrevInstance*/, _In_ PWSTR pCmdLine, _In_ int /*nCmdShow*/)`.

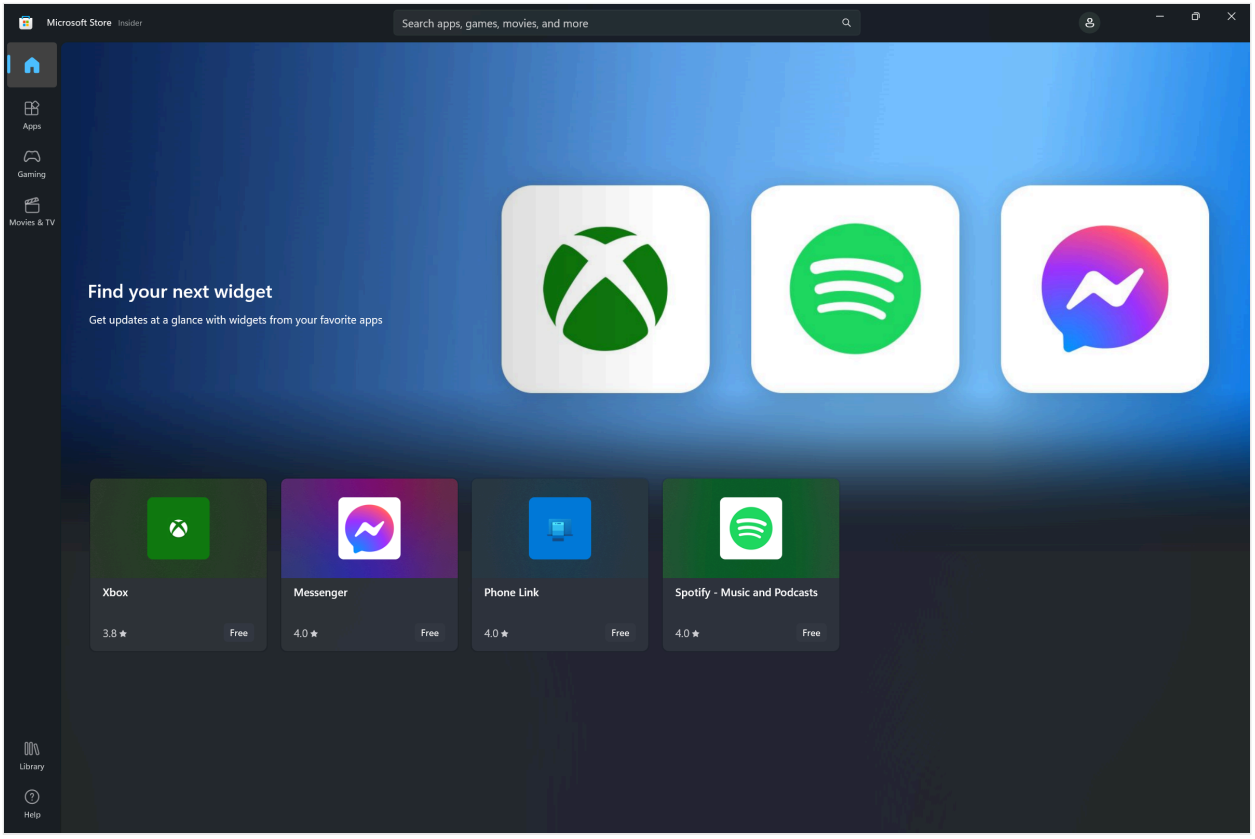


Publishing your widget

After you have developed and tested your widget you can publish your app on the Microsoft Store in order for users to install your widgets on their devices. For step by step guidance for publishing an app, see Publish your app in the Microsoft Store.

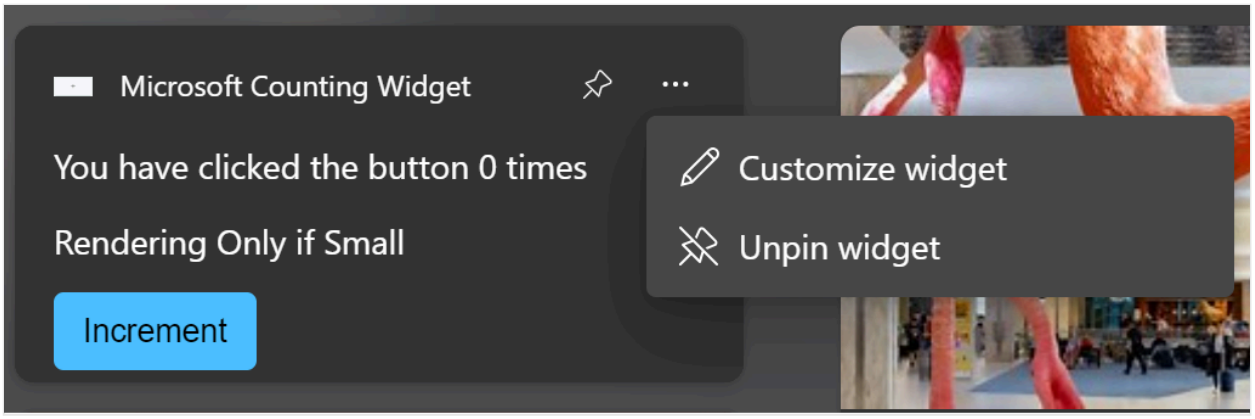
The widgets Store Collection

After your app has been published on the Microsoft Store, you can request for your app to be included in the widgets Store Collection that helps users discover apps that feature Windows Widgets. To submit your request, see Submit your Widget information for addition to the Store Collection .



Implementing widget customization

Starting with Windows App SDK 1.4, widgets can support user customization. When this feature is implemented, a **Customize widget** option is added to the ellipsis menu above the **Unpin widget** option.



The following steps summarize the process for widget customization.

1. In normal operation, the widget provider responds to requests from the widget host with the visual and data JSON templates for the regular widget experience.
2. The user clicks the **Customize widget** button in the ellipsis menu.
3. The widget raises the **OnCustomizationRequested** event on the widget provider to indicate that the user has requested the widget customization experience.
4. The widget provider sets an internal flag to indicate that the widget is in customization mode. While in customization mode, the widget provider sends the JSON templates for the widget customization UI instead of the regular widget UI.
5. While in customization mode, the widget provider receives **OnActionInvoked** events as the user interacts with the customization UI and adjusts its internal configuration and behavior based on the user's actions.
6. When the action associated with the **OnActionInvoked** event is the app-defined "exit customization" action, the widget provider resets it's internal flag to indicate that it is no longer in customization mode and resumes sending the visual and data JSON templates for the regular widget experience, reflecting the changes requested during customization.
7. The widget provider persists the customization options to disk or the cloud so that the changes are preserved between invocations of the widget provider.

> ⓘ **Note**
>
> There is a known bug with the Windows Widget Board, for widgets built using the Windows App SDK, that causes the ellipsis menu to become unresponsive after the customization card is shown.

In typical Widget customization scenarios, the user will choose what data is displayed on the widget or adjust visual presentation of the widget. For simplicity, the example in this section will add customization behavior that allows the user to reset the counter of the counting widget implemented in the previous steps.

> **Note**
>
> Widget customization is only supported in Windows App SDK 1.4 and later. Make sure you update the references in your project to the latest version of the Nuget package.

Update the package manifest to declare customization support

To let the widget host know that the widget supports customization, add the attribute **IsCustomizable** to the **Definition** eleent for the widget and set it to true.

XML

```
...
<Definition Id="Counting_Widget"
    DisplayName="Microsoft Counting Widget"
    Description="CONFIG counting widget description"
    IsCustomizable="true">
...
```

Update WidgetProvider.h

To add customization support to the widget that was created in the previous steps in this article, we will need to update the header file for our widget provider, WidgetProvider.h.

First, update the **CompactWidgetInfo** definition. This helper struct helps us track the current state of our active widgets. Add the *inCustomization* field, which will be used to track when the widget host is expecting us to send our customization json template rather than the regular widget template.

C++

```
// WidgetProvider.h
struct CompactWidgetInfo
{
    winrt::hstring widgetId;
    winrt::hstring widgetName;
    int customState = 0;
    bool isActive = false;
    bool inCustomization = false;
};
```

Update the **WidgetProvider** declaration to implement the IWidgetProvider2 interface.

C++

```
// WidgetProvider.h

struct WidgetProvider : winrt::implements<WidgetProvider,
winrt::Microsoft::Windows::Widgets::Providers::IWidgetProvider,
winrt::Microsoft::Windows::Widgets::Providers::IWidgetProvider2>
```

Add a declaration for the OnCustomizationRequested callback of the **IWidgetProvider2** interface.

C++

```
// WidgetProvider.h

void
OnCustomizationRequested(winrt::Microsoft::Windows::Widgets::Providers::Widg
etCustomizationRequestedArgs args);
```

Finally, declare a string variable that defines the JSON template for the widget customization UI. For this example, we have a "Reset counter" button and an "Exit customization" button that will signal our provider to return to regular widget behavior.

C++

```
// WidgetProvider.h
const std::string countWidgetCustomizationTemplate = R"(
{
    "type": "AdaptiveCard",
    "actions" : [
        {
            "type": "Action.Execute",
            "title" : "Reset counter",
            "verb": "reset"
            },
            {
            "type": "Action.Execute",
            "title": "Exit customization",
            "verb": "exitCustomization"
            }
    ],
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "version": "1.5"
})";
```

Update WidgetProvider.cpp

Now update the WidgetProvider.cpp file to implement the widget customization behavior. This method uses the same pattern as the other callbacks we have used. We get the ID for the widget to be customized from the **WidgetContext** and find the **CompactWidgetInfo** helper struct associated with that widget and set the **inCustomization** field to true.

C++

```
//WidgetProvider.cpp
void
WidgetProvider::OnCustomizationRequested(winrt::WidgetCustomizationRequested
Args args)
{
    auto widgetId = args.WidgetContext().Id();

    if (const auto iter = RunningWidgets.find(widgetId); iter !=
RunningWidgets.end())
    {
        auto& localWidgetInfo = iter->second;
        localWidgetInfo.inCustomization = true;

        UpdateWidget(localWidgetInfo);
    }
}
```

Next, we'll update our **UpdateWidget** helper method that sends our data and visual JSON templates to the widget host. When we are updating the counting widget, we send either the regular widget template or the customization template depending on the value of the **inCustomization** field. For brevity, code not relevant to customization is omitted in this code snippet.

C++

```
//WidgetProvider.cpp
void WidgetProvider::UpdateWidget(CompactWidgetInfo const& localWidgetInfo)
{
    ...
    else if (localWidgetInfo.widgetName == L"Counting_Widget")
    {
        if (!localWidgetInfo.inCustomization)
        {
            std::wcout << L" - not in customization " << std::endl;
            templateJson = winrt::to_hstring(countWidgetTemplate);
        }
        else
        {
            std::wcout << L" - in customization " << std::endl;
            templateJson =
winrt::to_hstring(countWidgetCustomizationTemplate);
        }
```

```
    }
    ...

    updateOptions.Template(templateJson);
    updateOptions.Data(dataJson);
    // !!  You can store some custom state in the widget service that you
will be able to query at any time.

updateOptions.CustomState(winrt::to_hstring(localWidgetInfo.customState));
    winrt::WidgetManager::GetDefault().UpdateWidget(updateOptions);
}
```

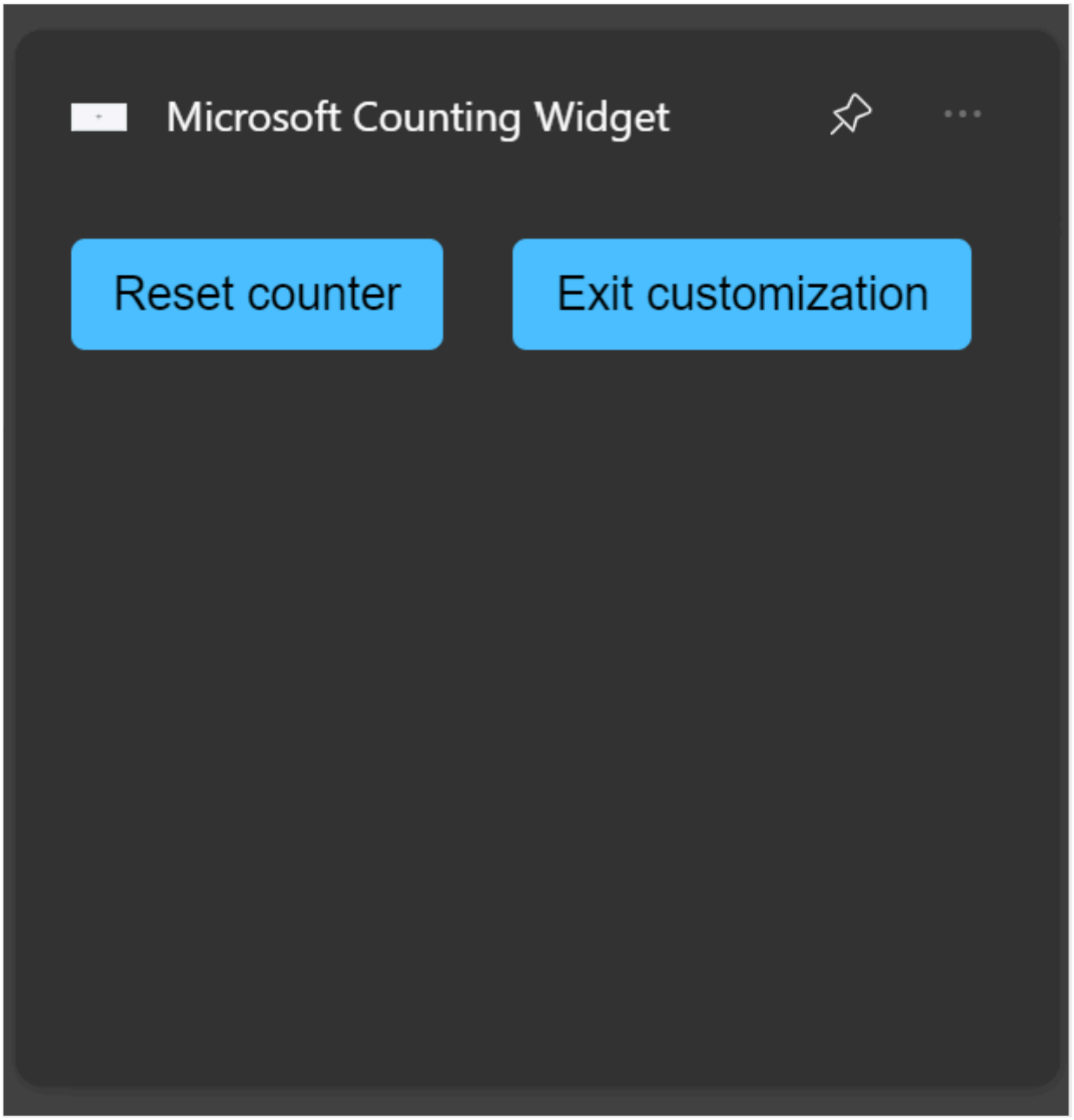
When users interact with inputs in our customization template, it calls the same **OnActionInvoked** handler as when the user interacts with the regular widget experience. To support customization, we look for the verbs "reset" and "exitCustomization" from our customization JSON template. If the action is for the "Reset counter" button, we reset the counter held in the **customState** field of our helper struct to 0. If the action is for the "Exit customization" button, we set the **inCustomization** field to false so that when we call **UpdateWidget**, our helper method will send the regular JSON templates and not the customization template.

C++

```
//WidgetProvider.cpp
void WidgetProvider::OnActionInvoked(winrt::WidgetActionInvokedArgs
actionInvokedArgs)
{
    auto verb = actionInvokedArgs.Verb();
    if (verb == L"inc")
    {
        auto widgetId = actionInvokedArgs.WidgetContext().Id();
        // If you need to use some data that was passed in after
        // Action was invoked, you can get it from the args:
        auto data = actionInvokedArgs.Data();
        if (const auto iter = RunningWidgets.find(widgetId); iter !=
RunningWidgets.end())
        {
            auto& localWidgetInfo = iter->second;
            // Increment the count
            localWidgetInfo.customState++;
            UpdateWidget(localWidgetInfo);
        }
    }
    else if (verb == L"reset")
    {
        auto widgetId = actionInvokedArgs.WidgetContext().Id();
        auto data = actionInvokedArgs.Data();
        if (const auto iter = RunningWidgets.find(widgetId); iter !=
RunningWidgets.end())
        {
            auto& localWidgetInfo = iter->second;
```

```
            // Reset the count
            localWidgetInfo.customState = 0;
            localWidgetInfo.inCustomization = false;
            UpdateWidget(localWidgetInfo);
        }
    }
    else if (verb == L"exitCustomization")
    {
        auto widgetId = actionInvokedArgs.WidgetContext().Id();
        // If you need to use some data that was passed in after
        // Action was invoked, you can get it from the args:
        auto data = actionInvokedArgs.Data();
        if (const auto iter = RunningWidgets.find(widgetId); iter !=
RunningWidgets.end())
        {
            auto& localWidgetInfo = iter->second;
            // Stop sending the customization template
            localWidgetInfo.inCustomization = false;
            UpdateWidget(localWidgetInfo);
        }
    }
}
```

Now, when you deploy your widget, you should see the **Customize widget** button in the ellipses menu. Clicking on the customize button will display your customization template.



Click the **Reset counter** button to reset the counter to 0. Click the **Exit customization** button to return to your widget's regular behavior.

Web widget providers

Article • 12/19/2024

In the latest release, apps that implement Windows widgets can choose to populate the widget content with HTML served from a remote URL. Previously, the widget content could only be supplied in the Adaptive Card schema format in the JSON payload passed from the provider to the Widgets Board. Because web widget providers must still provide an Adaptive Card JSON payload, you should follow the steps for implementing a widget provider in Implement a widget provider in a C# Windows App or Implement a widget provider in a win32 app (C++/WinRT).

Specify the content URL

Widget providers pass a JSON payload to the Widgets Board with a call to WidgetManager.UpdateWidget. For a web widget, instead of providing a **body** object defining the widget content, you should specify an empty **body** object and instead include a **metadata** object with a **webUrl** field that points to the URL that will supply the HTML content for the widget.

JSON

```
{
    "type": "AdaptiveCard",
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "version": "1.6",
    "body": [],
    "metadata":
    {
        "webUrl": "https://www.contoso.com/widgetprovider.html"
    }
}
```

Handle resource requests

Widget providers can specify a web request filter string for a widget in the *WebRequestFilter* attribute of the **Definition** element in the provider's package manifest file. Whenever the widget content requests a resource by URI that matches the filter, the request will be intercepted and redirected to the widget provider's implementation of IWidgetResourceProvider.OnResourceRequested.

The filter pattern is expressed using the format described in Match Patterns. The filter string in the registration must use Punycode where necessary. All content types will be

redirected when matched so the filter should only resolve to content intended to be obtained through the **IWidgetResourceProvider** in the application. For more information on the widget provider package manifest format, see Widget provider package manifest XML format.

To handle resource requests, widget providers must implement the IWidgetResourceProvider interface.

C#

```
internal class WidgetProvider : IWidgetProvider, IWidgetResourceProvider
```

In the implementation of the **OnResourceRequested** method, widget providers can provide the requested resources by setting the WidgetResourceRequestedArgs.Response property to a WidgetResourceResponse object containing the requested resource. When obtaining the resource asynchronously, the provider should request a deferral by calling WidgetResourceRequestedArgs.GetDeferral and then complete the deferral when the resource has been set.

C#

```
async void
IWidgetResourceProvider.OnResourceRequested(WidgetResourceRequestedArgs
args)
{
    var deferral = args.GetDeferral();

    if (args.Request.Uri.Length > 0)
    {
        if (args.Request.Uri == "https://contoso.com/logo-image")
        {
            string fullPath =
Windows.ApplicationModel.Package.Current.InstalledPath +
"/Assets/image.png";
            var file = await StorageFile.GetFileFromPathAsync(fullPath);
            var response = new
WidgetResourceResponse(RandomAccessStreamReference.CreateFromFile(file),
"OK", 200);
            response.Headers.Add("Content-Type", "image/png");
            args.Response = response;
        }
    }

    deferral.Complete();
}
```

If the provider does not set a response on the **WidgetResourceRequestedArgs** object passed into the method, the system will retrieve the resource from the web. In this case, the provider can choose to modify the Headers property of the WidgetResourceRequestedArgs.Request object, such as to provide user context or tokens, and the system will use the updated headers when retrieving the resource from the web.

Handle messages to and from web content

To receive string messages from the widget's content that has been posted using the window.chrome.webview.postMessage JavaScript method, widget providers can implement the IWidgetProviderMessage interface and implement the OnMessageReceived method.

C#

```
internal class WidgetProvider : IWidgetProvider, IWidgetProviderMessage
...
public void OnMessageReceived(WidgetMessageReceivedArgs args)
{
    Console.WriteLine($"Message received from widget
{args.WidgetContext.Id}: {args.Message}");
}
```

Widget providers can send a message to the web content of the widget by calling WidgetManager.SendMessage. You must provide the ID of the widget to which the message is sent, which is the value specified in the *Id* attribute of the **Definition** element in the provider's package manifest file. For more information see Widget provider package manifest XML format. The message string can be simple text or the serialized form of an object interpreted by the web content. For more information, see PostWebMessageAsString.

C#

```
var message = $"{{ \"current_location\": \"{ location }\" }}";
WidgetManager.GetDefault().SendMessageToContent("Weather_Widget", message);
```

Limitations and requirements

- This feature is available only to users in the European Economic Area (EEA). In the EEA, installed apps that implement a feed provider can provide content feed in the Widgets Board.

Customize the widget header area

Article • 12/19/2024

In the latest release, apps that implement Windows widgets can customize the header that is displayed for their widget in the Widgets Board, overriding the default presentation. Header customization is implemented in the Adaptive Card payload you pass to the OS from your widget provider, so the steps are the same regardless of the language your widget provider is implemented in. For a walkthrough of creating a widget provider, see Implement a widget provider in a C# Windows App or Implement a widget provider in a win32 app (C++/WinRT).

The default header

By default, the widget header shows the display name and the icon specified in the app manifest file. The display name is specified with the **DisplayName** attribute of the **Definition** element and the icon is specified with an **Icon** element under **ThemeResources**. For more information about the widget app manifest file format, see Widget provider package manifest XML format.

The following example shows a portion of the Adaptive Card JSON payload for a widget that uses the default presentation. In the sections below, examples will be provided that modify this template to override the default header.

JSON

```
{
    "type": "AdaptiveCard",
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "version": "1.6",
    "body": [
        ...
    ]
  }
```

Override the display name string

You can override the value specified in the **DisplayName** element in the app manifest by adding a `header` field to with the new display name in the JSON payload before sending it to the widget host.

The following example demonstrates overriding the display name string.

JSON

```
{
    "type": "AdaptiveCard",
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "version": "1.6",
    "body": [
        ...
    ] ,
    "header": "Redmond Weather"
  }
```

Override the display name string and icon

To override both the display name string and the icon specified in the app manifest, add a `header` object with fields for `text` and `iconUrl`.

The following example demonstrates overriding the display name string and icon.

JSON

```
{
    "type": "AdaptiveCard",
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "version": "1.6",
    "body": [
        ...
    ] ,
    "header": {
         "text": "Redmond weather",
         "iconUrl": "https://contoso.com/weatherimage.png"
      }
  }
```

Set the header to be empty

Some widget providers may want to allow their full UX to expand into the header region of the widget, even though this area of the widget isn't actionable. For this scenario, you can set the header to be empty by setting the `header` feel to `null`. Note that the UX in the header is not clickable by the user.

The following example demonstrates setting an empty header.

JSON

```
{
    "type": "AdaptiveCard",
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "version": "1.6",
    "body": [
        ...
    ] ,
    "header": null
}
```

Widget provider package manifest XML format

Article • 12/19/2024

In order to be displayed in the widgets host, apps that support Windows widgets must register their widget provider with the system. For Win32 apps, only packaged apps are currently supported and widget providers specify their registration information in the app package manifest file. This article documents the XML format for widget registration. See the Example section for a code listing of an example package manifest for a Win32 widget provider.

App extension

The app package manifest file supports many different extensions and features for Windows apps. The app package manifest format is defined by a set of schemas that are documented in the Package manifest schema reference. Widget providers declare their registration information within the uap3:AppExtension. The **Name** attribute of the extension must be set to "com.microsoft.windows.widgets".

Widget providers should include the uap3:Properties as the child of **uap3:AppExtension**. The package manifest schema does not enforce the structure of the **uap3:Properties** element other than requiring well-formed XML. The rest of this article describes the XML format that the Widget host expects in order to successfully register a widget provider.

XML

```
<uap3:Extension Category="windows.appExtension">
  <uap3:AppExtension Name="com.microsoft.windows.widgets"
DisplayName="WidgetTestApp" Id="ContosoWidgetApp" PublicFolder="Public">
    <uap3:Properties>
    <!-- Widget provider registration content goes here -->
    </uap3:Properties>
  </uap3:AppExtension>
</uap3:Extension>
```

Element hierarchy

WidgetProvider

ProviderIcons

Icon

Activation

CreateInstance

ActivateApplication

Definitions

Definition

Capabilities

Capability

Size

ThemeResources

Icons

Icon

Screenshots

Screenshot

DarkMode

Icons

Icon

Screenshots

Screenshot

LightMode

Icons

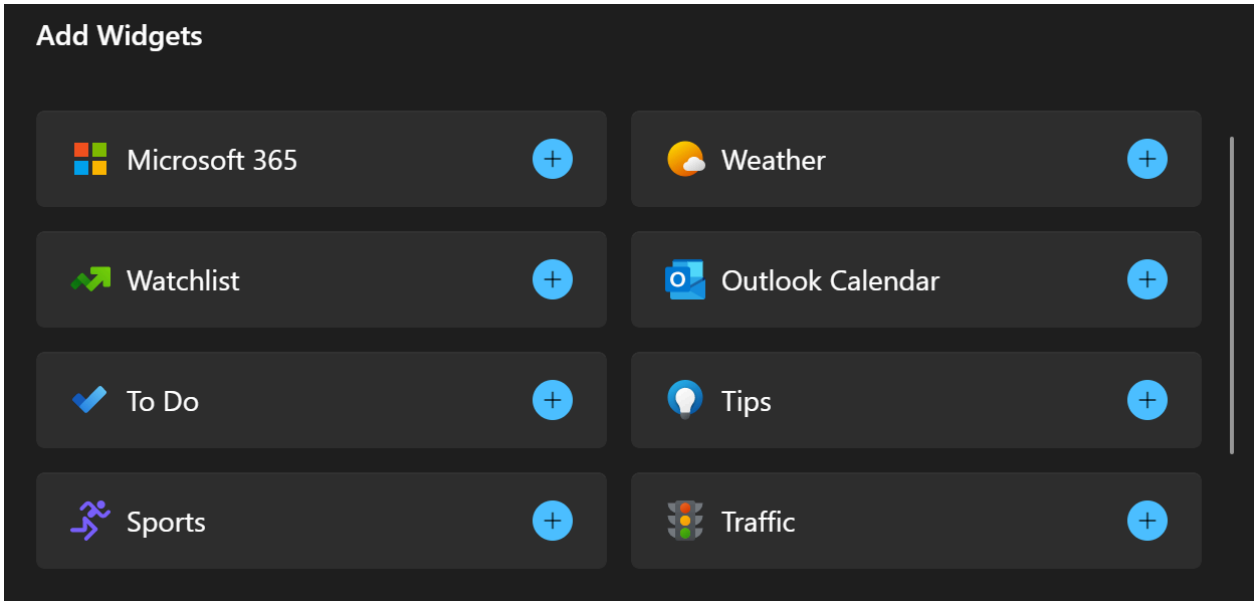
Icon

Screenshots

Screenshot

WidgetProvider

The root element of the widget provider registration information.



WidgetProviderIcons

Specifies icons representing the widget provider app.

Activation

Specifies activation information for the widget provider. If both **CreateInstance** and **ActivateApplication** are specified in the manifest, **CreateInstance** takes precedence.

CreateInstance

CreateInstance should be specified for Win32-based widget providers that implement the **IWidgetProvider** interface. The system will activate the interface with a call to CoCreateInstance. The **ClassId** attribute specifies the CLSID for the CreateInstance server that implements the **IWidgetProvider** interface.

Expand table

Attribute	Type	Required	Description	Default value
ClassId	GUID	Yes	The CLSID for the CreateInstance server that implements the widget provider.	N/A

ActivateApplication

When **ActivateApplication** is specified, the widget provider is activated via the command line, with the arguments provided as base64url encoded JSON strings. It is recommended that widget providers use the **CreateInstance** activation type. For information on the **ActivateApplication** command line format, see Widget provider ActivateApplication protocol.

Definitions

The container element for one or more widget registrations.

Definition

Represents the registration for a single widget.

Expand table

Attribute	Type	Required	Description	Default value
Id	string	Yes	An ID that identifies the widget. This value is also displayed in the navigation bar of the widget picker. Widget provider implementations use this string to determine or specify which of the app's widgets is being referenced for each operation. This string must be unique for all widgets defined within the app manifest file.	N/A
DisplayName	string	Yes	The name of the widget that is displayed on the widgets host.	N/A
Description	string	Yes	Short description of the widget.	N/A
AllowMultiple	boolean	No	Set to false if only one instance of this widget is supported. This attribute is optional and the default value is true.	true
IsCustomizable	boolean	No	Introduced in Windows App SDK 1.4. Set to true if your app supports widget customization. This causes the **Customize widget** button to be displayed in the widget's ellipsis menu.	false
AdditionalInfoUri	string	No	A URI that can be associated with the widget to be used when the user clicks on the title bar of	N/A

Attribute	Type	Required	Description	Default value
			the widget frame or when clicking the **Powered by** element of its context menu.	
ExcludedRegions	string	No	A list of regions where the widget should not be available. Widgets can specify **ExcludedRegions** or **ExclusiveRegions** but must not specify both in a single widget definition. The value of the attribute is a comma separated list of two character region codes.	N/A
ExclusiveRegions	string	No	A list of the only regions where the widget should be available. Widgets can specify **ExcludedRegions** or **ExclusiveRegions** but must not specify both in single widget definition. The value of the attribute is a comma separated list of two character region codes.	N/A
WebRequestFilter	string	No	Specifies the filter that specifies the resource request URLs for which the request will be intercepted and redirected to the widget provider's implementation of IWidgetResourceProvider.OnResourceRequested. The filter pattern is expressed using the format described in Match Patterns. The filter string in the registration must use Punycode where necessary. The filter string must match the origin of the widget registration, which is specified in the *webUrl* field of the adaptive card content.	N/A

Capabilities

Optional. Specifies capabilities for a single widget. If no capabilities are declared, one capability specifying a "large" size is added by default.

Capability

Specifies a capability for a widget.

Size

Specifies supported sizes for the associated widget.

Expand table

Attribute	Type	Required	Description	Default value
Name	string	Yes	Specifies a supported size for a widget. The value must be one of the following: "small", "medium", "large"	N/A

ThemeResources

Specifies theme resources for a widget.

Icons

A container element for one or more **Icon** elements.

Icon

Required. Specifies an icon that is displayed in the attribution area of the widget.

Expand table

Attribute	Type	Required	Description	Default value
Path	string	Yes	The package-relative path to an icon image file.	N/A

Screenshots

Required. Specifies one or more screenshots of the widget.

Screenshot

Required. Specifies a screenshot for a widget. This screenshot is shown in the widgets host in the **Add Widgets dialog** when the user is selecting widgets to add to the widgets host. If you provide a screenshot for the optional **DarkMode** or **LightMode** elements listed below, then the widgets host will use the screenshot that matches the current device theme. If you don't provide a screenshot for the current device theme, the image provided in this **Screenshot** element will be used. For information about the

design requirements for screenshot images and the naming conventions for localized screenshots, see Integrate with the widget picker.

> **Note**
>
> The widget screenshots are not displayed on the widgets board's add widgets dialog in the current preview release..

Expand table

Attribute	Type	Required	Description	Default value
Path	string	Yes	The package-relative path to a screenshot image file.	N/A
DisplayAltText	string	No	The alt-text for the image, for accessibility.	N/A

DarkMode

Optional. Specifies theme resources for when dark mode is active on the device. If you specify one or more screenshot images in the optional **DarkMode** element, the widgets host will select these screenshots when the device is in dark mode. If you don't provide a dark mode image, the widgets host will use the required, top-level **Screenshot** element described above. For information about the design requirements for screenshot images and the naming conventions for localized screenshots, see Integrate with the widget picker.

LightMode

Optional. Specifies theme resources for when light mode is active on the device. If you provide one or more screenshot images in the optional **LightMode** element, the widgets host will select these screenshots when the device is in light mode. If you don't provide a light mode image, the widgets host will use the required, top-level **Screenshot** element described above. For information about the design requirements for screenshot images and the naming conventions for localized screenshots, see Integrate with the widget picker.

Example

The following code example illustrates the usage of the widget package manifest XML format.

XML

```
<uap3:Extension Category="windows.appExtension">
  <uap3:AppExtension Name="com.microsoft.windows.widgets"
DisplayName="Widget Test App" Id="ContosoWidgetApp" PublicFolder="Public">
    <uap3:Properties>
      <WidgetProvider>
        <ProviderIcons>
            <Icon Path="Images\StoreIcon.png" />
        </ProviderIcons>
        <Activation>
          <!-- App exports COM interface which implements IWidgetProvider --
>
          <CreateInstance ClassId="XXXXXXXX-XXXX-XXXX-XXXX-D3397A3FF15C" />
        </Activation>
        <Definitions>
          <Definition
            Id="Weather_Widget"
            DisplayName="Microsoft Weather Widget"
            Description="Weather Widget Description"
            AdditionalInfoUri="https://contoso.com/widgets/Weather"
            ExclusiveRegions="US,UK"
            AllowMultiple="true">
            <Capabilities>
              <Capability>
                 <Size Name="small" />
              </Capability>
              <Capability>
                 <Size Name="medium" />
              </Capability>
              <Capability>
                 <Size Name="large" />
              </Capability>
            </Capabilities>

            <ThemeResources>
              <Icons>
                <Icon Path="Assets\icon.png" />
                <Icon Path="Assets\icon.gif" />
              </Icons>
              <Screenshots>
                <Screenshot Path="Assets\background.png" DisplayAltText
="For accessibility"/>
              </Screenshots>

              <!-- DarkMode and LightMode are optional -->
              <DarkMode>
                <Icons>
                  <Icon Path="Assets\dark.png" />
                </Icons>
                <Screenshots>
```

```
                    <Screenshot Path="Assets\darkBackground.png"
DisplayAltText ="For accessibility"/>
                  </Screenshots>
                </DarkMode>

                <LightMode>
                  <Icons>
                    <Icon Path="Assets\light.png" />
                  </Icons>
                  <Screenshots>
                    <Screenshot Path="Assets\lightBackground.png"/>
                  </Screenshots>
                </LightMode>
              </ThemeResources>
            </Definition>
          </Definitions>
        </WidgetProvider>
      </uap3:Properties>
    </uap3:AppExtension>
</uap3:Extension>
```

Widget provider ActivateApplication protocol

Article • 03/09/2023

This article describes the format of the command line activation parameters for widgets providers that use the **ActivateApplication** activation type.

> **ⓘ Important**
>
> Widget providers specify an activation method in the Widget provider manifest file as described in the Widget provider package manifest XML format. It is recommended that widget providers use the **CreateInstance** activation type and respond to widget host requests using the **IWidgetProvider** interface methods instead of using the **ActivateApplication** activation method. The information in this article is provided for completeness and is not recommended for use by most widget provider implementations.

Arguments string in ActivateApplication and base64url encoding

When the widget provider is activated the command line will have `--widget-call=` prefixed before the base64url encoding in the command line arguments.

`--widget-call=[base64url]`

For example, the base64url part in

`--widget-call=ew0KICAgICJXaWRnZXRDYWxsIjoiQ3JlYXRlV2lkZ2V0IiwNCiAgICAiV2lkZ2V0Q29udGV4dCI6ew0KICAgICAgICAiSWQiOiI5ODU4MjEwOS1jNmJmLTQzNzItODlkNi04OWY1N2ViNzU0ZjYiLA0KICAgICAgICAiRGVmaW5pdGlvbk5hbWUiOiJQV0FfQ291bnRpbmdfV2lkZ2V0IiwNCiAgICAgICAgIlNpemUiOiJMYXJnZSINCiAgICB9DQp9`

is decoded into

JSON

```
{
    "WidgetCall":"CreateWidget",
    "WidgetContext":{
```

```
        "Id":"98582109-c6bf-4372-89d6-89f57eb754f6",
        "DefinitionId":"PWA_Counting_Widget",
        "Size":"Large"
    }
}
```

Command line format details

The Json command line encodes calls to the `IWidgetProvider` methods:

```c++
    interface IWidgetProvider
    {
        void CreateWidget(WidgetContext widgetContext);
        void DeleteWidget(String widgetId);
        void OnActionInvoked(WidgetCallInvokedArgs actionInvokedArgs);
        void OnWidgetContextChanged(WidgetContextChangedArgs
contextChangedArgs);
    }
```

Each activation represents one method call. The Json object has a `WidgetCall` value with the method name, and then a value for each parameter, named as the parameter name except capitalized. Each parameter is mapped as a pair<variable name, value> to the json object. For example, for the `WidgetContext widgetContext` parameter of the `CreateWidget` method, the `WidgetContext` is defined like this:

```c++
    runtimeclass WidgetContext
    {
        String Id { get; };
        String DefinitionId{ get; };
        String Size { get; };
    };
```

> ⓘ **Note**
>
> Widget providers should ignore unexpected parameter values to handle the case where additional parameters are added in the future.

The widget provider `CreateWidget` API call is marshalled to Json object:

```JSON
```

```
{
    "WidgetCall":"CreateWidget",
    "WidgetContext":{
        "Id":"98582109-c6bf-4372-89d6-89f57eb754f6",
        "DefinitionId":"PWA_Counting_Widget",
        "Size":"Large"
    }
}
```

The object will always include the `WidgetCall` value and all parameter values as specified by the `IWidgetProvider` method.

Marshalled objects examples

Create Widget Json object

JSON

```
{
    "WidgetCall":"CreateWidget",
    "WidgetContext":{
        "Id":"98582109-c6bf-4372-89d6-89f57eb754f6",
        "DefinitionId":"PWA_Counting_Widget",
        "Size":"Large"
    }
}
```

Delete Widget Json object

JSON

```
{
  "WidgetId": "1AC74363-177B-4CD2-995F-3B25AEEA3FF4",
  "WidgetCall": "DeleteWidget",
  "CustomState": "usedata"
}
```

OnActionInvoked Json Object

JSON

```
{
    "WidgetCall": "OnActionInvoked",
    "Args":{
```

```
        "Verb": "Verb String",
        "Data": "Data Details",
        "CustomState": "usedata",
        "WidgetContext": {
            "Id":"98582109-c6bf-4372-89d6-89f57eb754f6",
            "DefinitionId":"PWA_Counting_Widget",
            "Size":"Large"
        }
    }
}
```

Activate Json Object

JSON

```
{
    "WidgetCall": "Activate",
    "WidgetContext": {
        "Id":"98582109-c6bf-4372-89d6-89f57eb754f6",
        "DefinitionId":"PWA_Counting_Widget",
        "Size":"Large"
    }
}
```

Deactivate Json Object

JSON

```
{
    "WidgetCall": "Deactivate",
    "WidgetId":"98582109-c6bf-4372-89d6-89f57eb754f6"
}
```

OnWidgetContextChanged Json Object

For example WidgetSize is changed to Medium. In SV2, WidgetSize is the only thing which trig the WidgetContextChanged.

JSON

```
{
  "WidgetCall": "OnWidgetContextChanged",
  "Args":{
    "WidgetContext":{
            "Id":"98582109-c6bf-4372-89d6-89f57eb754f6",
            "DefinitionId":"PWA_Counting_Widget",
```

```
            "Size":"Medium"
        }
    }
}
```

Windows Copilot Runtime Actions providers

Article • 03/31/2025

Actions are atomic units of functionality that can be invoked by the Windows Copilot Runtime. Developers can register actions with the system by implementing an action provider app.

Responsible AI Notes

When building AI backed actions, it is your responsibility as the Action author to perform content moderation and abuse monitoring when it comes to entities returned to the user. For more information about Microsoft Responsible AI policies for more information, see Microsoft Responsible AI: Principles and approach

> **Note**
>
> Consider if children should have access to the action using the 'contentAgeRating' property in the action definition JSON.

Related articles

- Action definition JSON schema for Windows Copilot Runtime Actions providers
- Windows Copilot Runtime Actions provider package manifest XML format

Action definition JSON schema for Windows Copilot Runtime Action providers

Article • 03/28/2025

This article describes the format of the action definition JSON file format for Windows Copilot Runtime Actions. This file must be included in your project with the **Build Action** set to "Content" and **Copy to Output Directory** set to “Copy if newer”. Specify the package-relative path to the JSON file in your package manifest XML file. For more information, see Action provider package manifest XML format.

Example action definition JSON file

JSON

```
"version": 1,
  "actions": [
    {
      "id": "Contoso.SampleGreeting",
      "description": "Send greeting with Contoso",
      "icon": "ms-resource//...",
      "inputs": [
        {
          "name": "UserFriendlyName",
          "kind": "Text"
        },
        {
          "name": "PetName",
          "kind": "Text",
          "required": false
        }
      ],
      "inputCombinations": [
        {
          "inputs": ["UserFriendlyName"],
          "description": "Greet ${UserFriendlyName.Text}"
        },
        {
          "inputs": ["UserFriendlyName", "PetName"],
          "description": "Greet ${UserFriendlyName.Text} and their pet
${PetName.Text}"
        }
      ],
      "contentAgeRating": "child",
      "invocation":
      {
```

```
        {
          "type": "Uri",
          "uri": "contoso://greetUser?
userName=${UserFriendlyName.Text}&petName=${PetName.Text}",
        },
        "where": [
          "${UserFriendlyName.Length > 3}"
        ]
      }
    },
    {
      "id": "Contoso.SampleGetText",
      "description": "Summarize file with Contoso",
      "icon": "ms-resource://...",
      "inputs": [
        {
          "name": "FileToSummarize",
          "kind": "File"
        }
      ],
      "inputCombinations": [
        {
          "inputs": ["FileToSummarize"],
          "description": "Summarize ${FileToSummarize.Path}"
        },
      ],
      "outputs": [
        {
          "name": "Summary",
          "kind": "Text"
        }
      ],
      "contentAgeRating": "child",
      "invocation": {
        "type": "COM",
        "clsid": "{...}"
      }
    }
  ]
}
```

Action definition JSON properties

The tables below describe the properties of the action definition JSON file.

Document root

Expand table

Property	Type	Description	Required
version	string	Schema version. When new functionality added, the version increments by one.	Yes.
actions	Action[]	Defines the actions provided by the app.	Yes.

Action

Expand table

Property	Type	Description	Required
id	string	Action identifier. Must be unique per app package. This value is not localizable.	Yes
description	string	User-facing description for this action. This value is localizable.	Yes
icon	string	Localizable icon for the action. This value is an *ms-resource* string for an icon deployed with the app.	No
inputs	Inputs[]	List of entities that this action accepts as input.	Yes
inputCombinations	InputCombination[]	Provides descriptions for different combinations of inputs.	Yes
outputs	Output[]	If specified, must be an empty string in the current release.	No
invocation	Invocation	Provides information about how the action is invoked.	Yes
contentAgeRating	string	A field name from the UserAgeConsentGroup that specifies the appropriate age rating for the action. The allowed values are "Child", "Minor", "Adult". If no value is specified, the default behavior allows access to all ages.	No

Output

Expand table

Property	Type	Description	Required
name	string	The variable name of the entity. This value is not localizable.	Yes
kind	string	A field name from the **ActionEntityKind** enumeration specifying the entity type. This value is not localizable. The allowed values are "None", "Document", "File", "Photo", "Text".	Yes

InputCombination

Expand table

Property	Type	Description	Required
inputs	string[]	A list of Input names for an action invocation. The list may be empty.	Yes
description	string	Description for the action invocation. This value is localizable.	No
where	string[]	One or more conditional statements determining the conditions under which the action applies.	No

Invocation

Expand table

Property	Type	Description	Required
type	string	The instantiation type for the action. The allowed values are "uri" and "com"	Yes
uri	string	The absolute URI for launching the action. Entity usage can be included within the string.	Yes, for URI instantiated actions.
clsid	string	The class ID for the COM class that implements **IActionProvider**.	Yes, for COM actions
inputData	A list of name/value pairs specifying additional data for URI actions.	No. Only valid for URI actions.	

ActionEntityKind enumeration

The **ActionEntityKind** enumeration specifies the types of entities that are supported by the Windows Copilot Runtime Actions. In the context of a JSON action definition, the entity kinds are string literals that are case-sensitive.

Expand table

Entity kind string	Description
"File"	Includes all file types that are not supported by photo or document entity types.
"Photo"	Image file types. Supported image file extensions are ".jpg", ".jpeg", and ".png"
"Document"	Document file types. Supported document file extensions are ".doc", ".docx", ".pdf", ".txt"
"Text"	Supports strings of text

Entity properties

Each entity type supports one or more properties that provide instance data for the entity. Entity property names are case sensitive.

The following example illustrates how entities are referenced in the query string for actions that are launched via URI activation:

```
...?param1=${entityName.property1}&param2=${entityName.property2}
```

For information on using entity properties to create conditional sections in the action definition JSON, see Where clauses.

File entity properties

Expand table

Property	Type	Description
"FileName"	string	The name of the file.
"Path"	string	The path of the file.
"Extension"	string	The extension of the file.

Document entity properties

The *Document* entity supports the same properties as *File*.

Photo entity properties

The *Photo* entity supports all of the properties of *File* in addition to the following properties.

Expand table

Property	Type	Description
"IsTemporaryPath"	boolean	A value specifying whether the photo is stored in a temporary path. For example, this property is true for photos that are stored in memory from a bitmap, not stored permanently in a file.

Text entity properties

Expand table

Property	Type	Description
"Text"	string	The full text.
"ShortText"	string	A shortened version of the text, suitable for UI display.
"Title"	string	The title of the text.
"Description"	string	A description of the text.
"Length"	double	The length of the text in characters.
"WordCount"	double	The number of words in the text.

Where clauses

The action definition JSON format supports *where* clauses that can be used to implement conditional logic, such as specifying that an action should be invoked only when an entity property has a specified value.

The following operators can be used with *where* clauses.

Expand table

Operator	Description
==	Equality
~=	Case-insensitive equality
!=	Inequality
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
||	Logical OR
&&	Logical AND

Where clauses use the following format:

JSON

```
"where": [
    "${<property_accessor>} <operator> <value>"
]
```

The following example shows a *where* clause that evaluates to true if a **File** entity has the file extension ".txt".

JSON

```
"where": [
    "${File.Extension} ~= \".txt\""
]
```

Multiple *where* clauses can be combined using the logical AND and logical OR operators.

JSON

```
"where": [
  "${File.Extension} ~= \".txt\" || ${File.Extension} ~= \".md\""
]
```

Related articles

Windows Copilot Runtime Action provider package manifest XML format

Article • 03/31/2025

This article describes the package manifest XML format for Windows Copilot Runtime Actions.

App extension

The app package manifest file supports many different extensions and features for Windows apps. The app package manifest format is defined by a set of schemas that are documented in the Package manifest schema reference. Action providers declare their registration information within the uap3:AppExtension. The **Name** attribute of the extension must be set to "com.microsoft.windows.ai.actions".

Action providers should include the uap3:Properties as the child of **uap3:AppExtension**. The package manifest schema does not enforce the structure of the **uap3:Properties** element other than requiring well-formed XML.

Action providers must provide a **Registration** element which specifies the path to the action definition JSON file. For more information, see Action JSON schema for Windows Copilot Action providers.

XML

```
<uap3:Extension Category="windows.appExtension">
    <uap3:AppExtension
        Name="com.microsoft.windows.ai.actions"
        DisplayName="..."
        Id="..."
        PublicFolder="Assets">
      <uap3:Properties>
        <Registration>path\to\registration.json</Registration> <!-- path
relative to the PublicFolder above -->
      </uap3:Properties>
    </uap3:AppExtension>
</uap3:Extension>
```

Additional requirements

Both COM and URI-launched action providers must have package identity. Package identity is declared in the app package manifest file using the Identity element. For more information, see An overview of Package Identity in Windows apps.

COM-based action providers must be *full trust apps* which have an integrity level of *mediumIL*. This is declared in the app package manifest file by setting the *uap10:TrustLevel attribute to "mediumIL".

URI-launched action providers must also have a trust level of *mediumIL*. If a URI-launched action provider will return outputs, the app must implement the ability to be launched for results. For more information, see Launch an app for results. URI-launched action providers that return outputs must also instantiate the runtime.

DispatcherQueue

Article • 08/31/2023

Highlights

- The DispatcherQueue class in the Windows App SDK manages a prioritized queue on which the tasks for a thread execute in a serial fashion.
- It provides a means for background threads to run code on a **DispatcherQueue**'s thread (for example, the UI thread where objects with thread-affinity live).
- The class precisely integrates with arbitrary message loops. For example, it supports the common Win32 idiom of nested message loops.
- The AppWindow class integrates with **DispatcherQueue**—when a **DispatcherQueue** for a given thread is being shut down, the **AppWindow** instances are automatically destroyed.
- It provides a means to register a delegate that's called when a timeout expires.
- It provides events that let components know when a message loop is exiting, and optionally defer that shutdown until outstanding work completes. That ensures components that use the **DispatcherQueue**, but don't own the message loop, may do cleanup on-thread as the loop exits.
- The **DispatcherQueue** is a thread singleton (there can be at most one of them running on any given thread). By default, a thread has no **DispatcherQueue**.
- A thread owner may create a DispatcherQueueController to initialize the **DispatcherQueue** for the thread. At that point, any code can access the thread's **DispatcherQueue**; but only the **DispatcherQueueController**'s owner has access to the DispatcherQueueController.ShutdownQueue method, which drains the **DispatcherQueue**, and raises **ShutdownStarted** and **ShutdownCompleted** events.
- An outermost message loop owner must create a **DispatcherQueue** instance. Only the code in charge of running a thread's outermost message loop knows when dispatch is complete, which is the appropriate time to shut down the **DispatcherQueue**. That means that components that rely on **DispatcherQueue** mustn't create the **DispatcherQueue** unless they own the thread's message loop.

Run-down

After a thread exits its event loop, it must shut down its **DispatcherQueue**. Doing so raises the **ShutdownStarting** and **ShutdownCompleted** events, and drains any final pending enqueued items before disabling further enqueuing.

- To shut down a **DispatcherQueue** that's running on a dedicated thread with a **DispatcherQueue**-owned message loop, call the DispatcherQueueController.ShutdownQueueAsync method.
- For scenarios where the app owns an arbitrary message loop (for example, XAML Islands), call the synchronous DispatcherQueueController.ShutdownQueue method. That method raises shutdown events, and drains the **DispatcherQueue** synchronously on the calling thread.

When you call either **DispatcherQueueController.ShutdownQueueAsync** or **DispatcherQueueController.ShutdownQueue**, the order of events raised is the following:

- **ShutdownStarting**. Intended for apps to handle.
- **FrameworkShutdownStarting**. Intended for frameworks to handle.
- **FrameworkShutdownCompleted**. Intended for frameworks to handle.
- **ShutdownCompleted**. Intended for apps to handle.

The events are separated into application/framework categories so that orderly shutdown can be achieved. That is, by explicitly raising application shutdown ahead of framework shutdown events, there's no danger that a framework component will be in an unusable state as the application winds down.

C++/WinRT

```
namespace winrt
{
    using namespace Microsoft::UI::Dispatching;
}

// App runs its own custom message loop.
void RunCustomMessageLoop()
{
    // Create a DispatcherQueue.
    auto
dispatcherQueueController{winrt::DispatcherQueueController::CreateOnCurrentT
hread()};

    // Run a custom message loop. Runs until the message loop owner decides
to stop.
    MSG msg;
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!ContentPreTranslateMessage(&msg))
        {
            TranslateMesasge(&msg);
            DispatchMessage(&msg);
        }
    }
```

```
    // Run down the DispatcherQueue. This single call also runs down the
system DispatcherQueue
    // if one was created via EnsureSystemDispatcherQueue:
    // 1. Raises DispatcherQueue.ShutdownStarting event.
    // 2. Drains remaining items in the DispatcherQueue, waits for
deferrals.
    // 3. Raises DispatcherQueue.FrameworkShutdownStarting event.
    // 4. Drains remaining items in the DispatcherQueue, waits for
deferrals.
    // 5. Disables further enqueuing.
    // 6. Raises the DispatcherQueue.FrameworkShutdownCompleted event.
    // 7. Raises the DispatcherQueue.ShutdownCompleted event.

    dispatcherQueueController.ShutdownQueue();
}
```

Outermost and recursive message loops

DispatcherQueue supports custom message loops. However, for simple apps that don't need customization, we provide a default implementations. That removes a burden from developers, and helps ensure consistently correct behavior.

C++/WinRT

```
namespace winrt
{
    using namespace Microsoft::UI::Dispatching;
}

// Simple app; doesn't need a custom message loop.
void RunMessageLoop()
{
    // Create a DispatcherQueue.
    auto
dispatcherQueueController{winrt::DispatcherQueueController::CreateOnCurrentT
hread()};

    // Runs a message loop until a call to
DispatcherQueue.EnqueueEventLoopExit or PostQuitMessage.
    dispatcherQueueController.DispatcherQueue().RunEventLoop();

    // Run down the DispatcherQueue.
    dispatcherQueueController.ShutdownQueue();
}

// May be called while receiving a message.
void RunNestedLoop(winrt::DispatcherQueue dispatcherQueue)
{
    // Runs a message loop until a call to
DispatcherQueue.EnqueueEventLoopExit or PostQuitMessage.
    dispatcherQueue.RunEventLoop();
```

```
}

// Called to break out of the message loop, returning from the RunEventLoop
call lower down the
// stack.
void EndMessageLoop(winrt::DispatcherQueue dispatcherQueue)
{
    // Alternatively, calling Win32's PostQuitMessage has the same effect.
    dispatcherQueue.EnqueueEventLoopExit();
}
```

System dispatcher management

Some Windows App SDK components (for example, MicaController) depend on system components that in turn require a system **DispatcherQueue** (**Windows.System.DispatcherQueue**) running on the thread.

In those cases, the component that has a system **DispatcherQueue** dependency calls the EnsureSystemDispatcherQueue method, freeing your app from managing a system **DispatcherQueue**.

With that method called, the Windows App SDK **DispatcherQueue** manages the lifetime of the system **DispatcherQueue** automatically, shutting down the system **DispatcherQueue** alongside the Windows App SDK **DispatcherQueue**. Components might rely on both Windows App SDK and system **DispatcherQueue** shutdown events in order to ensure that they do proper cleanup after the message loop exits.

C++/WinRT

```
namespace winrt
{
    using namespace Microsoft::UI::Composition::SystemBackdrops;
    using namespace Microsoft::UI::Dispatching;
}

// The Windows App SDK component calls this during its startup.
void MicaControllerInitialize(winrt::DispatcherQueue dispatcherQueue)
{
    dispatcherQueue.EnsureSystemDispatcherQueue();

    // If the component needs the system DispatcherQueue explicitly, it can
now grab it off the thread.
    winrt::Windows::System::DispatcherQueue systemDispatcherQueue =
        winrt::Windows::System::DispatcherQueue::GetForCurrentThread();
}

void AppInitialize()
{
```

```
    // App doesn't need to concern itself with the system DispatcherQueue
dependency.
    auto micaController = winrt::MicaController();
}
```

AppWindow integration

The AppWindow class has functionality that integrates it with the **DispatcherQueue**, so that **AppWindow** objects can automatically be destroyed when the DispatcherQueueController.ShutdownQueueAsync or DispatcherQueueController.ShutdownQueue method is called.

There's also a property of **AppWindow** that allows callers to retrieve the **DispatcherQueue** associated with the **AppWindow**; aligning it with other objects in the **Composition** and **Input** namespaces.

AppWindow needs your explicit opt-in in order to be aware of the **DispatcherQueue**.

C++/WinRT

```
namespace winrt
{
    using namespace Microsoft::UI::Dispatching;
    using namespace Microsoft::UI::Windowing;
}

void Main()
{
    // Create a Windows App SDK DispatcherQueue.
    auto
dispatcherQueueController{winrt::DispatcherQueueController::CreateOnCurrentT
hread()};

    var appWindow = AppWindow.Create(nullptr, 0,
dispatcherQueueController.DispatcherQueue());

    // Since we associated the DispatcherQueue above with the AppWindow,
we're able to retreive it
    // as a property. If we were to not associate a dispatcher, this
property would be null.
    ASSERT(appWindow.DispatcherQueue() ==
dispatcherQueueController.DispatcherQueue());

    // Runs a message loop until a call to
DispatcherQueue.EnqueueEventLoopExit or PostQuitMessage.
    dispatcherQueueController.DispatcherQueue().RunEventLoop();

    // Rundown the Windows App SDK DispatcherQueue. While this call is in
progress, the AppWindow.Destoyed
```

```
    // event will be raised since the AppWindow instance is associated with
the DispatcherQueue.
    dispatcherQueueController.ShutdownQueue();
}
```

Windows app performance and fundamentals overview

Article • 02/27/2025

Improving the performance and fundamentals of your application gives it a feeling of polish and craftsmanship, while saving your customers time, money, battery life, and development effort. Your app will consume less power, improving battery life and reducing carbon emissions. Apps run more smoothly on less expensive hardware. Your customers' productivity increases.

This page provides an overview of the technologies and development tools for measuring, understanding, and refining the performance of your Windows application. If you are writing a UWP application, be sure to also consult the UWP Performance documentation.

What is application performance and why is it important?

Performance in the context of an application usually revolves around *cost*. How long is it going to take to complete a particular task? How much of the system's resources will be used?

The answers to these questions play a fundamental role in the quality of a user's experience with an application (you can likely recall times where as a user, you have felt frustrated at an application for its poor performance!). As a developer, by keeping performance in mind, you will ensure that users of your applications don't experience that same frustration.

To learn more about the importance of performance and suggestions on how to get started, see this introduction to the world of performance.

What tools can I use to measure application performance?

There are a variety of different options available for measuring the performance of your Windows application.

If you are not sure what tools to choose, check out the article: Choosing between Visual Studio Performance Profiler and Windows Performance Toolkit.

Visual Studio Performance Profiler

Visual Studio offers tooling to help you monitor your application and give you insights within your source. Visit the resources below to learn about how you can use these tools to optimize your code right from your development environment.

- Measure app performance in Visual Studio
- Case Study: Using Visual Studio Profiler to reduce memory allocations in the Windows Terminal console host startup path
- Case Study: Double Performance in under 30 minutes

Windows Performance Toolkit

Windows Performance Recorder and Windows Performance Analyzer enable detailed monitoring and analysis of your application and the entire system using Event Tracing for Windows (ETW). See the links below on how to get started.

- Getting Started with the Windows Performance Toolkit
- Windows Performance Recorder Intro
- Windows Performance Analyzer Intro

PerfView

PerfView is an open source monitoring and analysis tool created by the .NET team for investigating .NET performance issues. Because of its ability to decode .NET symbols and managed memory, it is an ideal choice for managed applications.

- PerfView GitHub Repository
- PerfView Tutorial Series

SizeBench

SizeBench is a utility that helps you investigate and reduce the size of your compiled native code binaries (DLLs, EXEs, and other PE files).

- SizeBench: A new tool for analyzing Windows binary size .
- Microsoft Store Download Link

Additional Resources

Blogs and news

Go behind the scenes with developer blogs, written by our performance experts to empower building the best version of your app.

- .NET Performance Blog
- Visual Studio Performance Blogs
- Windows Performance and Diagnostics Blog

Community and support

- Performance problems in your compile-debug-test loop? Report them in the Windows Dev Performance repo on GitHub.

Performance and Sustainability

Performance engineering intersects directly with the sustainable software movement. Most electrical grids burn fossil fuels to generate electricity. As your application runs on a PC, it consumes incremental power that may be small for a single user but adds up as your user base grows.

Interested in learning how performance engineering intersects with sustainability? Check out the Principles of Green Software Engineering and Microsoft's Sustainable Software blog.

- Measuring Your Application Power and Carbon Impact
- Principles of Sustainable Software Engineering
- Role of Performance Engineering in Designing Carbon Efficient Applications
- Green Energy Efficient Progressive Web Apps.

Our content road map

Performance can be a tricky part of your development process without the right guidance. Ensuring that the right documentation is available for app developers is key to creating faster apps. The road map below details the next pages to be added to this doc set, and in the order that they will be published. If you see something you like or

something that's missing, let us know on GitHub via the Feedback links at the bottom of this page! Our goal is to curate a well-rounded and educational doc set for app performance that meets your needs, so we'd love to hear from you. Please feel welcome to submit feedback for this page below with your recommendations for what you would like to see covered in regard to Windows application performance.

Expand table

Topic	Description
Introduction to performance areas	Descriptions of what performance means in the context of CPU, memory, GPU, etc.
Identifying what to measure	Depending on your workload, environment, and other factors, certain areas of performance may be more of a focus than others. Learn what to measure and when with this doc
Performance Testing Cycle	Step with us through the performance testing lifecycle which includes setting up your test environment, analyzing your results and making product improvements
Understanding the different performance tools	This doc will introduce more performance tooling and go over the use cases and best practices for using them
Case Studies	A series of end-to-end scenarios and their journeys through the performance testing cycle

Introduction to Windows Application Performance

Article • 11/05/2021

Performance is an aspect of software development that focuses on measuring and improving code. After reading this article, you will have a better idea of: what performance entails, why it is important for you and your customers, and how to get started with measuring your application's performance.

What is application performance?

Performance is the measure of how effectively your application uses the system's resources to do what you've designed it to do. It covers different aspects of how your program interacts with the underlying device. This can include things like:

- CPU usage
- Memory consumption
- Power consumption
- Network and storage utilization
- Animation performance

...and more.

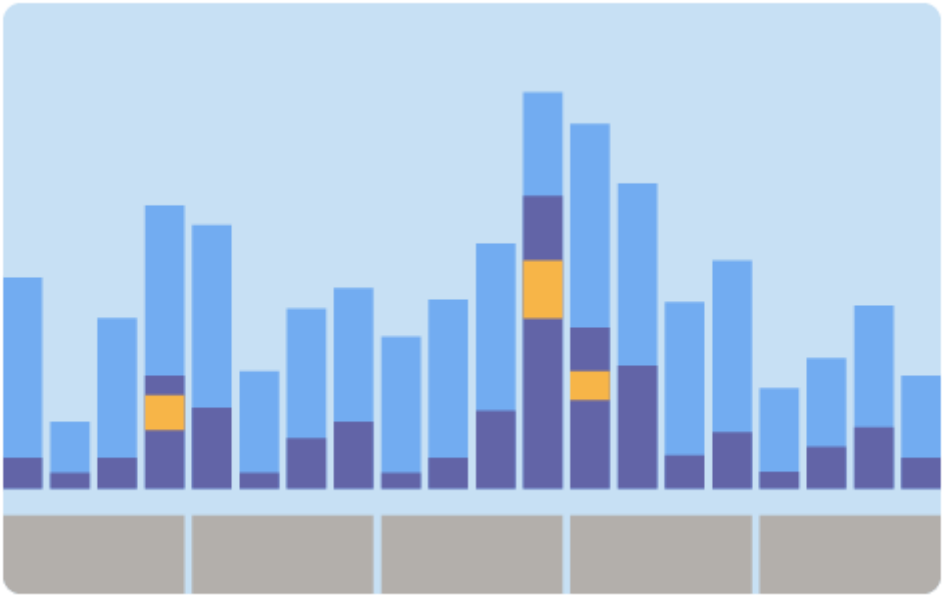
All of these different properties have an element of cost associated with them: for example, how much CPU does my application use? How much of the user's bandwidth

will it consume? How fast does this particular page of my application load? Measurement is therefore central to performance.

Why is it important?

Users expect performance as a fundamental property of the software they use. They want their applications to be responsive and make efficient use of their system's resources. Applications that exhibit poor performance cause frustration, which can lead to reduced user engagement. To provide your customers with the best possible experience, it is therefore crucial to make performance a regular part of your development workflow.

When should you measure application performance?



Application performance can span many stages of the development process. It has implications on everything ranging from your choice of data structure to the technology that you choose for building your application.

Keep performance in mind as you are developing your application, and plan to do regular performance testing as part of updating and maintaining your application.

How is application performance measured?

Here are some suggestions for how to approach testing your application for performance.

- **Leverage your knowledge of your application.** Understanding the most common scenarios for your users will enable you to spend your time wisely on optimizing the right things. Unclear scenarios can make your measurements difficult to interpret, so don't skip this step! If you have data available on how users interact with your application, this would be a great time to look at it.
  - Where are your users spending most of their time?
  - What are the most important things that a customer will do with your software?
  - What are your application's hardware requirements?
- **Set performance goals for your most important user scenarios.**
- **Be precise about what you are trying to optimize.** Is it CPU? Battery? Network throughput?
- **Select the tools you will use to do your measurements.**
- **Apply a scientific mindset when testing.** Create benchmarks in a controlled environment. Then, make your change, and re-measure to see how your changes have affected your application's behavior.
- **Add regression testing into your test environment.** This will ensure that your performance metrics don't regress over time. Modern hardware is complex and measuring the impact of your performance improvements can be a challenge. Investigating, measuring, and testing your application performance in these ways will ensure you know the impact of your work.

Intertwining metrics

While you will typically focus on one area of performance during your analysis, be aware that areas are often intertwined. An improvement in one can cascade into an improvement in the other areas.

For example, fixing power consumption is frequently a synchronization problem. Reducing memory usage can result in reducing the time spent using the CPU.

There can also be situations where additional resources spent in one area yield a more impactful improvement in another area.

For example, increasing memory consumption can decrease network or storage utilization through caching.

The decision to make a change depends on what is most important for your customers. It is also part of the challenge (and fun!) of performance work.

Next steps

With this background in place, here are some actions you can now take:

- **Identify your application's most common scenarios**. These will be the basis of what you are trying to measure.
- **Figure out your test strategy.** What will be your test environment and choice of measurement tools?

Head to the Windows app performance overview to learn more about the tools and guidance that are available for you to use on your performance journey.

Choosing among Visual Studio Performance Profiler, Windows Performance Toolkit, and PerfView

Article • 03/03/2025

This guide provides a general comparison of the primary general-purpose performance profiling technologies at Microsoft: Visual Studio Performance Profiler, Windows Performance Toolkit, and PerfView.

These tools can help you to diagnose and understand the performance characteristics of your applications on Windows. The goal of this guide is to provide an overview of when to use one over another, highlighting each of their strengths, and offering a brief overview of the functionality they contain.

Overview

Visual Studio Performance Profiler is created by the Visual Studio team for quickly understanding performance characteristics of an application under development inside the existing Visual Studio project system. Its strength is integrating tightly with the active development project, simplifying analysis of the most common performance scenarios, and quick, easy collection of only a single application.

Windows Performance Toolkit is created by the Windows team for understanding system wide characteristics of the entire PC at once. While it grew up from the need to analyze hardware and drivers, it is very effective on understanding software problems as well. Its strength is around gathering large quantities of information from the entire machine at once, so multi-process issues, those dealing with hardware or drivers, and complex scenarios are well-matched for these tools.

PerfView is created by the .NET team for understanding the performance of .NET applications. Like Windows Performance Toolkit, it can gather large quantities of information from the entire machine at once. It is differentiated by its ability to display very detailed information about .NET runtime services such as garbage collection, just-in-time compilation, and the managed thread pool. PerfView can be used for managed, native, and mixed-mode applications.

There is overlap between these tools. Often you can start exploring an issue in one tool and switch to another for a different view of the same scenario. Other times, one of the tools will be more effective than the others.

Installation

The Visual Studio Performance Profiler is a component of Visual Studio itself and is installable through the same installation wizard as the rest of the development environment.

The Windows Performance Toolkit is downloadable separately as a part of the Windows Assessment and Deployment Kit. However, the command-line Performance Recorder tool is pre-installed with Windows 10 and Windows 11 as `wpr.exe` available on the default path variable from shells like PowerShell and the Command Prompt.

The Performance Analyzer is available as an application in the Microsoft Store.

PerfView is available as a standalone download in the PerfView GitHub Releases page. No installation is required.

Customizing for advanced performance analysis scenarios

Windows Performance Toolkit additionally offers two extensibility points that can serve advanced performance analysis scenarios.

- The Microsoft Performance Toolkit SDK handles the processing of trace data and enables developers to build their own plugins to be viewed inside the Windows Performance Analyzer.
- .NET TraceProcessing allows the authoring of custom tools that can process trace information into resulting tables and is especially useful for automated analysis of bulk trace data.

PerfView is similarly based on the .NET TraceEvent library for programmatically consuming performance traces. TraceEvent can be used independently of PerfView to perform custom analysis of performance trace data.

> **Note**
>
> For performance analysis scenarios to work, you will need access to the **symbols** that correlate with the Windows application being tested. When building with Visual Studio, they will be located the same as in the **debugging scenario settings**, either built with your solution or captured from symbol servers. When analyzing other libraries or components, you will have to locate the symbols for those components to complete your analysis.

Considerations for choosing a tool

The tool to choose depends on the performance scenario that you are attempting to explore. A comparison of functionalities and traits among the tools is provided in the following table where:

- **Well supported** means the tool is designed for the intended task and achieves robust and detailed results.
- **Supported** means the tool may require additional configuration or steps to achieve the desired results. May contain a limited scope of action within the category.
- **Not supported** means the tool is not designed for this use.

> **Tip**
>
> For a general rule, start with *Visual Studio Profiling* when possible. Move on to *Windows Performance Toolkit* or *PerfView* when reaching the limits of what the Visual Studio tools can do.

Expand table

Situation	Visual Studio Performance Profiler	Windows Performance Toolkit	PerfView
General use considerations			
Trace File Size	**Well supported**	**Supported**	**Supported**
Acquisition of Tooling	**Well supported**	**Well supported**	**Well supported**
Extensibility Kits	Not supported	**Well supported**	**Well supported**
Scope of work considerations			
Single Process or Project	**Well supported**	**Supported**	**Supported**
Multiple Processes	Not supported	**Well supported**	**Well supported**
Hardware considerations			
CPU Usage	**Well supported**	**Well supported**	**Well supported**

Situation	Visual Studio Performance Profiler	Windows Performance Toolkit	PerfView
GPU Usage	**Well supported**	**Well supported**	Not supported
Memory Usage	**Well supported**	**Well supported**	**Well supported**
Device Input/Output	Not supported	**Well supported**	**Well supported**
Power Usage	Not supported	**Well supported**	Not supported
System Handles	Not supported	**Well supported**	**Well supported**
Code language support considerations			
Support for C/C++	**Well supported**	**Well supported**	**Well supported**
Support for .NET	**Well supported**	**Well supported**	**Well supported**
Support for JavaScript	**Supported**	**Supported**	**Supported**
Scenario considerations			
Event Tracing for Windows	**Supported**	**Well supported**	**Well supported**
Composition	Not supported	**Well supported**	Not supported
HTML/Edge/Internet Explorer/Webview	Not supported	**Well supported**	Not supported
XAML/WinUI	**Supported**	**Well supported**	**Well supported**
Audio/Video pipelines and glitches	Not supported	**Well supported**	Not supported
Database timing	**Well supported**	Not supported	Not supported
Managed object allocation and garbage collection	**Well supported**	Not supported	**Well supported**

Situation	Visual Studio Performance Profiler	Windows Performance Toolkit	PerfView
Custom Scenarios	Not supported	**Well supported**	**Well supported**

General use considerations

For those just getting started with performance analysis, we recommend Visual Studio Performance Profiler as a well-integrated feature inside the existing Visual Studio development suite. We recommend using Visual Studio Performance Profiler over PerfView if it meets your needs.

For more complex system performance analysis that may require more power and versatility, we recommend Windows Performance Toolkit, which consists of two tools used to accomplish performance analysis tasks:

- The **Windows Performance Recorder**, available both as a command-line tool and with a graphical interface, is responsible for capturing the trace session.
- The **Windows Performance Analyzer** is opened later to post-process the collected data and provide a highly customizable analysis view.

A few of the benefits that Visual Studio Performance Profiler offers include:

- A **good introduction** to performance analysis in the major domains.
- It **handles many of the complexities of analysis and debugging automatically** based on the project configuration.
- It **automatically highlights major areas of concern**.
- It is **better for focusing on just one application**, with a smaller, more focused data set and a smaller collection scope.
- More specific focus translates to **less impact on other applications** and machine hardware while profiling occurs, a **reduced overall size to the tracing files** generated and stored, and an **increased processing speed** for reviewing information after collection is complete.
- Takes **less time to start and complete a trace**, with a faster ability to review and turn around because Visual Studio Performance Profiler is concerned only with the application and not the entire system.
- Data collection and resulting analysis are **all performed within Visual Studio**, with analysis pages launching automatically on the conclusion of collection. The report view also automatically tends to focus attention on hot spots or areas of action.

A few of the benefits that Windows Performance Toolkit offers include:

- **Ample documentation and blogs** are provided by the Windows Performance and Diagnostics teams to help you get started.
- It is **better for collecting very large files**, especially from systems that are busy with background tasks. It is a catch-all tool for collecting information that will then be filtered later in the Windows Performance Analyzer interface.
- **Ability to be customized for advanced performance analysis scenarios** using extensibility points. (See below)

A few of the benefits that PerfView offers include:

- Comprehensive **built-in documentation and Internet-accessible how-to videos** linked from within the app.
- It is **easy to deploy** to production environments by simply copying PerfView.exe.
- A **flight-recorder mode** for capturing hard-to-reproduce issues.
- Very detailed diagnostics for .NET runtime services.
- Extensible for custom views.

Scope of work considerations

For analyses surrounding a **single application** and especially a **single process**, all of the tools are very capable for collecting and understanding performance. The advantage, however, would side with Visual Studio Performance Profiler, especially when the application source and project system is already available. The Visual Studio Performance Profiler engine is designed to collect CPU, GPU, and memory information from a binary in a similar streamlined fashion to the F5 debugging capability. While only focusing on the one application at hand, this collection mechanism offers a tighter turn around and developer loop.

Choose PerfView if Visual Studio doesn't have the necessary capabilities, can't be run due to collection requirements (common in production environments), or more detailed .NET diagnostic capabilities are needed.

For situations with **larger complexity** that include **multiple cross-process requests**, **hardware devices** and their **drivers**, or deep dives into **Windows platform technologies**, the Windows Performance Toolkit is the optimal choice.

Hardware considerations

Visual Studio Performance Profiler, Windows Performance Toolkit, and PerfView can diagnose CPU and memory for major hardware components, with Visual Studio

Performance Profiler and Windows Performance Toolkit additionally supporting GPU. The tools are generally well-matched at introductory analysis in these areas.

CPU usage can be analyzed with all three tools and is typically captured using sampling. The sampling captures stack traces from the application periodically and provides a ranking on how often they appear. Both tools can adjust this behavior to instead use instrumentation for exact accounting.

GPU usage can be analyzed with both Visual Studio Performance Profiler and Windows Performance Toolkit capturing a general overview of information.

Memory usage can be analyzed with all three tools collecting information on the heap space and the stacks associated with allocations in the heap.

When more complex scenarios arise, such as analyzing networking, disk, devices, handles, or overall power consumption of the system, Windows Performance Toolkit is more equipped to handle the analysis. This data is best collected directly from the operating system as it dispatches requests to the various hardware components. Windows Performance Toolkit is developed in tandem with the operating system, so it is ready and capable of collecting this more system-focused category of information.

Code language support considerations

Windows Performance Toolkit is primarily focused on support for C and C++, as these languages are used in the Windows operating system codebase.

Visual Studio Performance Profiler support is focused on a wider range of programming languages, starting with .NET originating technologies like C# and ASP.NET then expanding outward.

PerfView supports .NET and native (C and C++) applications. It has deep knowledge of .NET runtimes, and capabilities around ASP.NET web workloads.

This is not to say that Visual Studio cannot analyze C or C++ code, or that Windows Performance Analyzer cannot analyze .NET applications, or that PerfView cannot analyze web applications. It is just best to start with the tool most closely matching the application being analyzed to take advantage of each tool's strengths.

Scenario considerations

All of the tools contain several scenario-based options for studying application performance.

Visual Studio Performance Profiler options tend to focus on .NET, user interface (UI), and databases residing within the application being analyzed.

Windows Performance Toolkit tends to focus on operating system components and frameworks like composition, browser views, and glitches in realtime processing pipelines.

For scenarios that require a more customized solution, Windows Performance Recorder can combine both system-wide collection and application-related collection data into a single recording session. This functionality enables an application developer to offer a complete solution to deploy to customers for requesting performance data from hardware that might differ from that used to develop the product. More information on this mechanism can be found at the Authoring Custom Profiles blog series by the Windows Performance and Diagnostics team.

PerfView is targeted at deep investigations of .NET applications and runtimes, but is just as capable in multi-process and operating system-level investigations.

Summary

Visual Studio Performance Profiler, Windows Performance Toolkit, and PerfView are three robust tools from Microsoft available to you for understanding your application's performance. Choosing which tool will best meet your needs requires a variety of considerations regarding your specific situation. We hope that this guide will provide the information needed for you to make wise performance analysis choices, but also welcome you to file feedback on this page below, or for issues specifically pertaining to Windows development performance, please file an issue on the Windows Dev Performance repo.

Improve app performance by reducing the use of memory and disk space

Article • 05/17/2022

This guide will demonstrate how to improve the performance of your Windows application in two primary ways:

- Minimize memory usage
- Efficiently use Disk Space

Minimize memory usage

There are a variety of ways to minimize the amount of memory that your Windows app uses, you can:

- Reduce foreground memory usage
- Minimize background work
- Release resources while in the background
- Ensure your application does not leak memory

In order to appropriately minimize memory usage, it is first important to understand:

- how memory is used and allocated, using the working set, dynamic memory, and virtual allocation,
- how to capture a system trace, and
- how to analyze the system trace.

Once you have a system trace to analyze, we will recommend guidance for Applying your trace analysis to reduce memory usage.

Working set, dynamic memory, and virtual allocation

The working set of an application – the set of pages in its virtual address space that currently resident in memory – is one measure of the app's memory usage.

The amount of memory that an application uses impacts its runtime performance, as well as the responsiveness of the system as a whole. Minimizing the use of memory will help the app to perform better by reducing the CPU costs associated with accessing more memory. Lower memory usage also helps with the system responsiveness, and the

app user's experience in general, as the application does not end up displacing other memory content.

Memory displacement can happen as the system attempts to retain in-memory content accessed recently and, if necessary, will trim and page-out content used earlier. When the user switches back to the shell, or another application, and the necessary data is not resident in-memory, the data will need to be read from the disk. The user will likely notice a slowdown due to this process.

There are two key pieces to the memory used by an application: 1) dynamic memory, and 2) file-backed memory. File-backed memory usage comes from binaries and data files, such as databases, used by an application. This is commonly not a significant chunk of an application's memory usage and often a constant. *(Exceptions would be data processing applications, code compilation etc.)* The more significant source of memory usage and where leaks manifest is dynamic memory.

Dynamic memory corresponds to virtual memory allocated by an application using memory allocation routines. Unlike file-backed memory, which persists across system reboots, dynamic memory only exists for the lifetime of the application. Dynamic memory is a commonly significant source of memory usage and where memory leaks manifest.

Virtual Allocation routines (`VirtualAlloc`) handle memory allocation requests from a Windows application independent of the application-layer routine used for memory allocation. While not all of the memory allocated by an application may be resident in memory all of the time, analyzing such allocations provides a consistent way to understand the memory usage of an application.

To understand your application's memory usage and find places to make improvements, we recommend capturing a VirtualAllocation trace as described below.

Capture a system trace to analyze memory usage

Recording device activity over a period of time is known as *system tracing*. System tracing produces a trace file that can be used to generate a report and help you identify how to be improve your app's performance.

Traces can vary in length:

- A short running trace can be used to capture the launch of an application. This can include your app's transition to the inactive state, where the application window is

minimized or the application windows is closed while the application process persists.
- A long running trace, typically several minutes in duration, is useful in diagnosing memory leaks. If the memory usage continues to trend up over time, this is usually suggestive of a leak.

There are multiple tools available for monitoring memory use, including:

- Windows Performance Analyzer
- Visual Studio Memory usage diagnostic tool

For the purposes of this article, we will focus on using the Windows Performance Analyzer. To learn more about choosing a tool for profiling the performance of your application, see Choosing among Visual Studio Performance Profiler, Windows Performance Toolkit, and PerfView.

To capture a trace:

1. Open a command line (PowerShell or Command Prompt) in administrator mode. (If not run in admin mode, you may receive error code: 0xc5585011, "Failed to enable the policy to profile system performance.")
2. Enter the command: `wpr -start VirtualAllocation -filemode`
3. Run the scenario that you are investigating. (Launching your application, for example.)
4. Enter the command: `wpr -stop Trace.etl`

Analyze the system trace

In order to find which of your app's functions have allocated memory that you may be able to reduce, you now need to analyze the system trace that was captured. To analyze the trace:

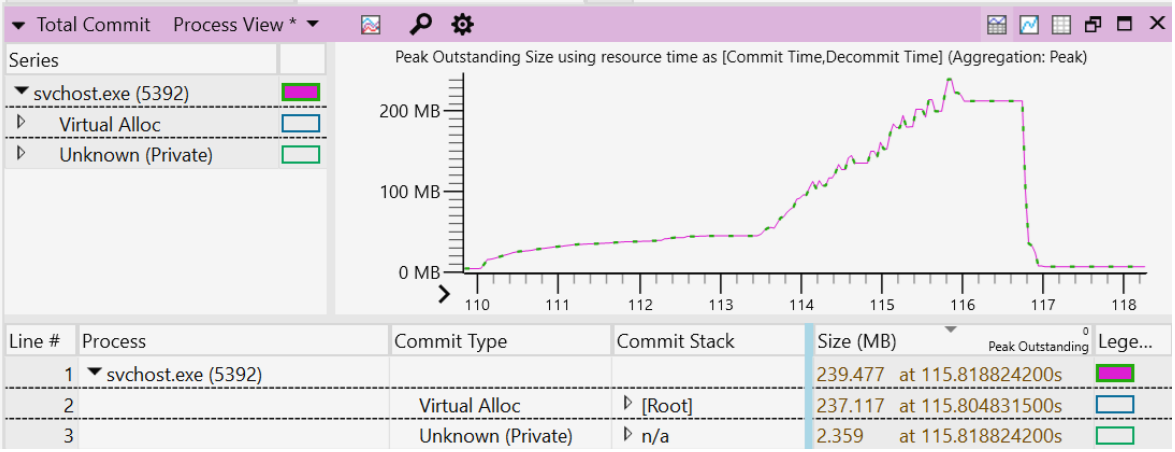
1. Open the trace using Windows Performance Analyzer, by entering the command: `wpa.exe Trace.etl`
2. In the Graph Explorer window, expand the **Memory** section, right-click on the **Total Commit** graph, and select **Add graph to New Analysis View**.
3. Open the **View Editor** by clicking on the **Settings** gear and selecting the following column arrangement: **Process**, **Commit Type**, **Commit Stack** and **Size**.

4. Click on the **Size** column header so that the results are sorted in descending order. The **Commit Stack** shows the code path leading to memory being allocated. These results may help to understand the reason for the allocation. Sorting by size enables you to focus on the larger allocations and investigate whether there is an opportunity to optimize.

5. Filter to the process(es) that you are interested in analyzing by right-clicking on the process and selecting **Filter To Selection**.

6. To zoom in to your region of interest in the viewport, select a range, right-click on the graph, and select **Zoom**.

7. Navigate through the Commit Stack to understand which functions have allocated memory. Commit stacks will need symbols loaded. To load symbols, select **Trace** > **Load symbols** from the top navigation menu bar.



Applying your trace analysis to reduce memory usage

In analyzing the allocated memory, you will find clues to help you decide where memory usage can be minimized.

A few areas to consider in regard to applying your trace analysis toward updating your code to reduce memory usage, include:

- **Reduce memory usage when in the foreground**: Analyzing the memory trace may help you to identify any unnecessary memory use in the foreground and update your code to reduce or remove that usage.

- **Minimizing work while in the background**: The system has policies to age out pages from the process working sets. Using less memory in the background allows the system to be more efficient by keeping less of the application memory

resident. Learn more about how to Improve power consumption and battery life by minimizing background work, which will also translate into using less memory usage while in the background.

- **Release resources in the background**: At runtime, an application may create some memory caches, as well as creating graphics allocations to support its UI. These allocations may be able to be freed when the application is minimized or not visible. An application can register for low-memory notifications to take such action, but a better strategy may be to free memory after a period of not being used, when the application concludes it is inactive. This period of disuse can vary by application, so possible indicators of inactive use can range from a handful of minutes to a ½ hour or more. Care should be taken to balance this kind of memory savings with responsiveness. If a cache is expensive to rebuild, the application can choose to retain it for the lifetime of the application.

- **Ensure your application does not leak memory**: To check for memory leaks, first establish a steady-state benchmark, where the application memory usage plateaus or does not grow beyond a certain value. You can establish this steady-state by continuously using the application or leaving it idle in the background. Using the trace you've captured to identify a possible memory leak, you can find where that memory is being allocated in your code and how it can be freed of use after it has served its purpose. If the memory continues to grow as the application is running, this is a likely indication of a memory leak. Zoom in to the region corresponding to the growth within your trace and carefully analyze the commit stacks.

Efficiently use Disk Space

Disk footprint refers to the size of an application when it's stored in an inactive state (not executing code). If the application takes up a lot of disk footprint it can be an opportunity to optimize.

There are several ways that reducing the disk footprint of your app can improve performance:

- As a disk gets full, the file system can no longer store new content in a contiguous fashion. A full disk becomes fragmented, storing new content on non-contiguous sectors. This translates to longer latency time when that content is accessed from the disk. IO systems will provide much better disk throughput when the content is contiguous and can be accessed sequentially or using larger IOs.

- A full disk may translate to longer write latencies for SSD-based systems. When there are fewer empty cells to absorb writes, a write may incur a read-modify-write

operation, slowing performance.

- A full disk can hinder the ability to update your application. While the OS is resilient and able to keep the system up to date and secure, even with low available disk space, a healthy amount of available disk space for staging the content for your app update will translate to a faster, smoother update experience.

- Requiring a significant amount of a large disk footprint to be accessed at run time will translate to memory use as well. This will impact the responsiveness of your application and the system in general. Additionally, if a small proportion of the disk footprint is required at run time, then the application may be using disk space inefficiently.

A few ways to reduce, or be more efficient with, your disk space include:

- **Apply "pay-for-play" principles to your disk footprint (download only what you need)**: An application may include a broad range of features with not all features applying to all users. This can be one reason for a large on disk footprint. By applying "pay-for-play" principles, you can ask users to select only to download the features that they need, translating to a smaller disk footprint when they download your app. Additional content is made optional for download only when the user has a need for richer capabilities. In addition to features, you may apply the same "pay-for-play" principles to language support. The application can include a subset of popular language choices by default, with additional languages optionally included or dependent on the location set on the user's system.

- **Apply efficient cache sizing**: In some cases, an application can use on-disk caches to make the user experience more responsive. Policies can be set for how your application manages the cache, with an upper limit set on the cache size based on disk capacity and resizing the cache when available space on the disk is low.

- **Apply efficient use of assets**: An application will often include image assets and may consist of a range of image size to support multiple resolutions. Optimizing image size, dimensions, format, and compression for a subset of resolutions, and leveraging scaling to support remaining resolutions, can significantly reduce disk footprint.

- **Investigate binary optimization opportunities**: Tools, such as SizeBench, enable application authors to investigate what contributes to the binary footprint and find opportunities to reduce the amount of disk space used.

Additional resources

- Windows Performance Analyzer step-by-step guide
- Windows Performance Analyzer common scenarios
- Tracelogging
- Event Tracing for Windows
- Memory Footprint Optimization
- Measure memory usage in Visual Studio
- Planning for performance
- Q&A forum

Improve power consumption and battery life by minimizing background work

Article • 05/17/2022

This guide will demonstrate how to improve the way that your Windows application consumes power, offering principles to help you improve overall device performance and battery life.

The most basic principles for improving power consumption are to ensure that while your app is in the background, it does not:

- use system resources,
- wake the CPU through timers,
- wake the CPU by waiting for vsync events.

Being "in the background" means that your app is not visible, nor audible, to the user. There should be few, if any, reasons to wake the CPU or utilize system resources while in the background. Checking whether your app is triggering these events and addressing the cause can significantly improve your app's power consumption, battery life, and reduce the carbon emissions that will result from customers using your app.

To check that the CPU is not waking up to do unnecessary work while your app is not in use, you can capture and analyze a trace using Windows Performance Analyzer (WPA). We will walk through the steps explaining how to do so below.

Capture a trace and check for unnecessary work

It is important to set up the correct conditions when running your app before capturing a trace to analyze whether unnecessary work is being performed that can drain power. You also may want to run multiple traces in order to test your app under a variety of conditions.

Setup

1. Get your test device to idle by opening Task Manager and ensuring that CPU utilization is less than 5%. Doing so will minimize measurement interference and provide reasonable size traces.

2. Launch your app and navigate to a view that's expected to be commonly used.

3. Minimize your app. *You may want to repeat the following steps while your app window is fully occluded (blocked from view) behind other windows and when your app is in the foreground, after a period of no interaction by the user, to ensure unnecessary work is not being initiated by the app.

Measure and evaluate CPU use while in the background

1. Open a command line (PowerShell or Command Prompt) in administrator mode. (If not run in admin mode, you may receive error code: 0xc5585011, "Failed to enable the policy to profile system performance.")

2. Enter the command: `wpr -start power -filemode`

3. Leave device idle for 5 minutes.

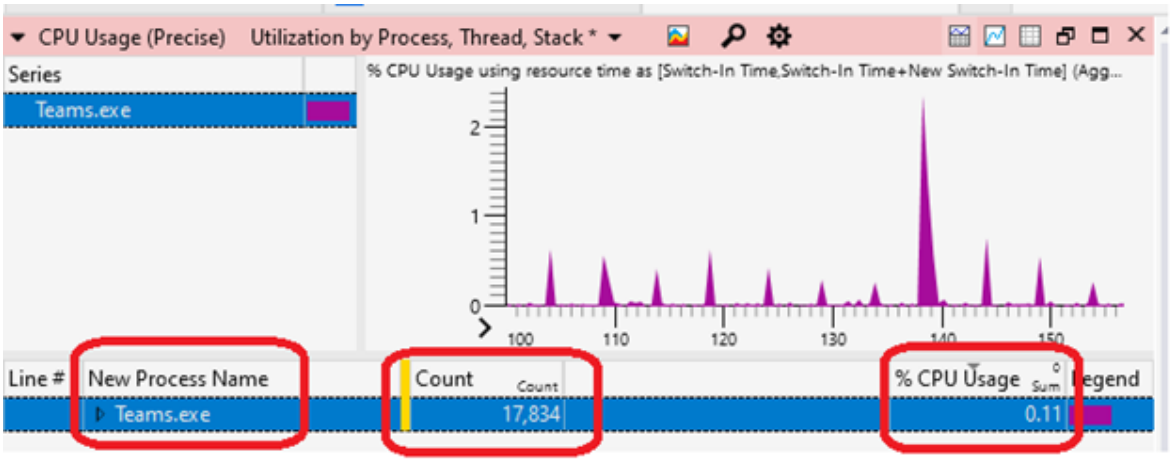
4. Enter the command: `wpr -stop idletrace.etl`

5. To open the trace using Windows Performance Analyzer (WPA), enter the command: `wpa.exe idletrace.etl`

6. Open the **Computation** > **CPU Usage (Precise)** graph.

    - Navigate to your process (or if you have a multi-process model all processes related to your app) and evaluate the Cswitch count, "Utilization by CPU". Ideally these are as close to 0 as possible while the app is in background. Looking at the stacks is a great way to find where resources are being used and how to improve it.

    - Look for excessive wakes by adding a "New Thread Stack" to see which stack the thread woke on.

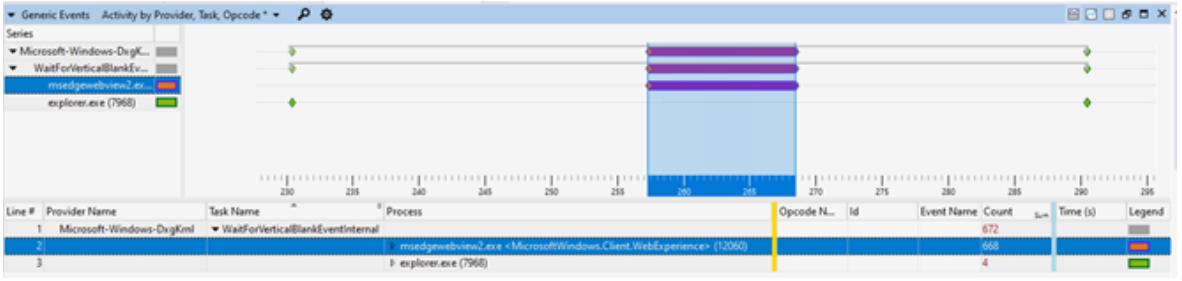


7. Open the **Computation** > **CPU Usage (Sampled)** graph. Track down CPU usage issues by evaluating where in-code time is being spent executing.

Measure and evaluate vsync waiting while in the background

> **Note**
>
> Focus on basic CPU usage in the previous section before investigating this more advanced scenario.

1. Open a command line (PowerShell or Command Prompt) in administrator mode.
2. Enter the command: `wpr -start gpu -filemode`
3. Leave the device idle for 5 minutes.
4. Enter the command: `wpr -stop gputrace.etl`
5. To open the trace using Windows Performance Analyzer (WPA), enter the command: `wpa.exe gputrace.etl`
6. Open the **System Activity** > **Generic events** graph.
   - Filter to "Microsoft-Windows-Dxgkrnl" events.
   - Drag the **Process** column over to be the 2nd column.
   - Check if you are calling into WaitForVsync. For example, through APIs like IDXGIOutput::WaitForVBlank (dxgi.h) through the `WaitForVerticalBlankEventInternal` event.



To learn more, see the article Using Windows Performance Analyzer to analyze Modern Standby issues.

Additional resources

- Windows Performance Analyzer step-by-step guide
- Windows Performance Analyzer common scenarios
- Using Windows Performance Analyzer to analyze Modern Standby issues
- CPU Analysis
- Event Tracing for Windows
- Q&A forum
- List of Windows Performance Toolkit (WPA) Graphs

Improve the responsiveness of your Windows app

Article • 05/17/2022

This guide will help you to improve the responsiveness of your Windows application by optimizing latency for launch and key interactions. Quick, responsive interactions (otherwise known as low-latency interactions) creates a better user experience. Understanding interaction expectations, measuring current durations, and setting goals for improvement depending on the interaction class can lead to significant improvements in your users experience and satisfaction.

Step by step guide to optimizing interactions for responsive behavior

Customers can feel when an application launch, menu navigation, or page/content load is slow. They have come to expect a fast, seamless experience and we have shared some steps and guidance to better understand those expectations below.

The basic steps to drive latency optimization are:

1. Define the scenario and add TraceLogging events.

   Make a list of the key interactions that a user will go through while using your app, such as launch, opening a menu, navigating to a new page and rendering content, etc. For each of these interactions, add a start event and stop event to be used for measurement and analysis. Learn more about how to add TraceLogging, a format for self-describing Event Tracing for Windows (ETW).

2. Set goals based on the interaction class.

   Users have different expectations for an app's performance and responsiveness depending on the type of interaction. For example, how quickly an app launches versus how quickly a page loads. Think of the acceptable range of elapsed time that it takes for users to complete the key interactions in your app. This may range from 200 milliseconds (ms) to 5 seconds (sec). Then assign each task an interaction class label with an associated goal. Below are a few basic guidelines, along with suggestions for how you might include a user interface (UI) to improve the perception of responsiveness:

Expand table

Interaction class label	User perception	Range of delay	Examples	Suggested UI
Fast	Minimally noticeable delay	100 - 200 milliseconds	Open app bar, right click menu	
Interactive	Quick, but not fast	300 - 500 ms	Exit an app, display cached search results	
Pause	Not quick, but feels responsive	500 ms - 1 sec	Navigate to a different page, resume the app from a suspended state, display web search results	An entrance animation (e.g. fly in new content) may be used to mask the time taken for this scenario.
Wait	Not quick due to amount of work for scenario	1 - 3 sec	Launching the app	A spinning/waiting cursor may be used to note progress. Both an exit and entrance animation (e.g. fly old page out, fly new page in) may be used to mask the time taken for this scenario.
Long wait	No longer feels responsive	2 - 5 sec	Large app launches (use extended splash screen), starting an HD video stream	A "loading UI" is displayed – where possible, include a "cancel" option for the user. The loading UI should appear within the Fast interaction class. The loading UI does not need to display a percentage or time remaining.
Captive	A long wait – reserved for unavoidably long/complex scenarios	5 - 10 sec	System login	A "loading UI" or dialog is displayed – where possible, include a "cancel" option for the user. The dialog should appear within the Fast interaction class. The dialog should display a percentage or time remaining if this would provide useful context to the user.

Interaction class label	User perception	Range of delay	Examples	Suggested UI
Long-running	Long operations – users will probably multitask (switch away during operation)	10 - 30+ sec	Installing new features or updates, large file downloads	UI should be designed to reflect multitasking possibility. A progress dialog should be displayed including an estimate of completion (percentage, time remaining, etc.). Alternatively, the UI can minimize completely and only notify the user when the scenario has finished by using a toast notification.

3. To check the exact durations for specific interactions, you can capture and analyze a trace using Windows Performance Analyzer (WPA).

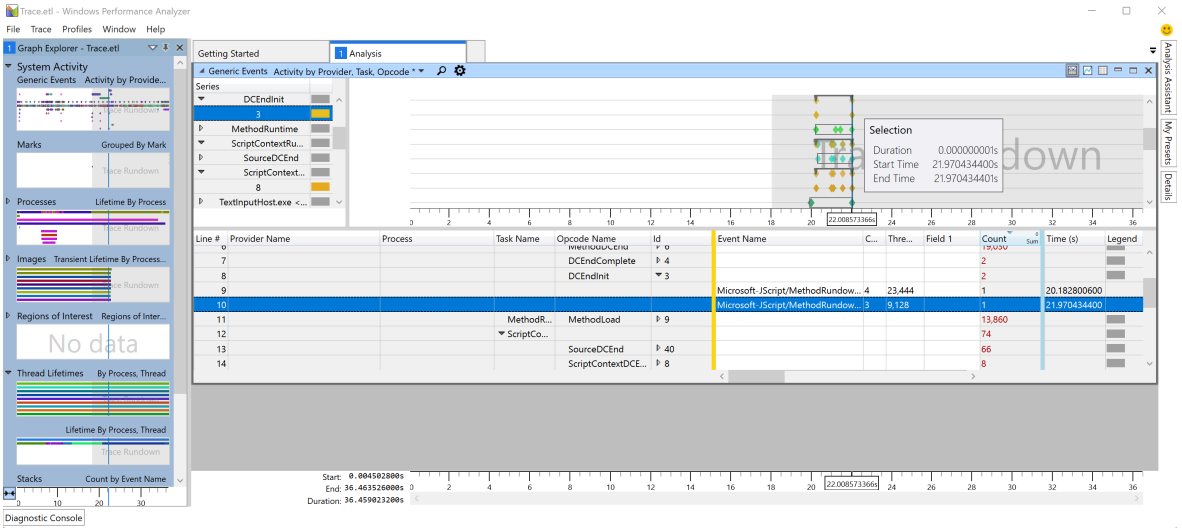
    - Before capturing your trace, get your test device to idle by opening Task Manager and ensuring that CPU utilization is less than 5%. Doing so will minimize measurement interference and provide reasonable size traces. This will help you to better isolate the interaction that you aim to measure.
    - To capture a trace, open a command line (PowerShell or Command Prompt) in administrator mode.
    - Enter the command: `wpr -start GeneralProfile -filemode`
    - Run the interaction scenario on your app.
    - Enter the command: `wpr -stop Trace.etl`

4. Analyze the trace and find improvement opportunities.

    Analyze the duration of each key interaction supported by your app by opening the trace that you just captured in Windows Performance Analyzer (WPA).

    - To open the trace in WPA, in your command line, enter: `wpa.exe Trace.etl`
    - Expand the **System Activity** dropdown and double-click on "Generic Events" to open the analysis view.
    - Choose the event series related to your application and expand the "Provider Name" dropdown until you are able to find the Process, Task Name, and Event name associated with the key interaction you are seeking. The duration of the interaction event will be listed in the **Time** column, as well as in the graph, showing **Duration**, **Start Time**, and **End Time**.

If the duration from your trace does not meet the interaction class goals (ex: 500ms), identify the key threads for your App (likely the UI thread), and look at the top stacks for CPU usage and waits. *Remember that as you perform the analysis, some issues may not be specific to your own application's code.



Additional resources

- Windows Performance Analyzer step-by-step guide
- Windows Performance Analyzer: Optimizing Performance and Responsiveness
- TraceLogging
- Event Tracing for Windows
- Planning for performance
- Q&A forum
- List of Windows Performance Toolkit (WPA) Graphs

WinUI performance optimization

Article • 03/22/2024

This topic describes how to use performance monitoring tools from the Windows Performance Toolkit to produce in-depth performance profiles for WinUI applications.

How do I use the Windows Performance Recorder to monitor WinUI apps?

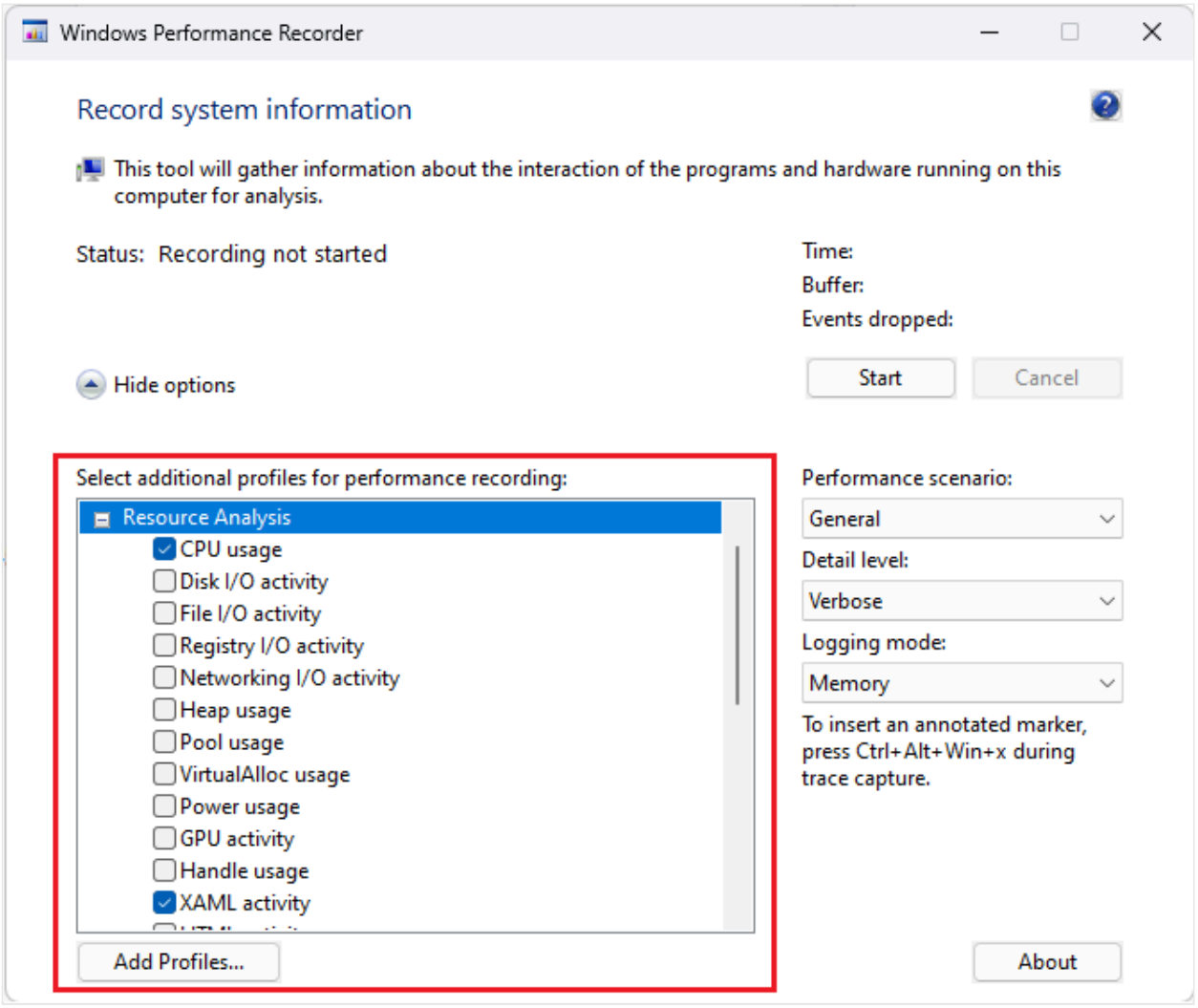
The Windows Performance Recorder (WPR) can be used to create detailed Event Tracing for Windows (ETW) recordings of system and application behavior and resource usage based on built-in profiles. These ETW recordings can then be processed by the Windows Performance Analyzer (WPA) to produce a set of graphs and tables for easier consumption and in-depth analysis of CPU usage, power issues, poor system or application performance, and other performance issues.

> **Note**
>
> While there are both GUI and command-line versions of the WPR, this topic refers only to the GUI version (see **Introduction to WPR** for more details on both versions).

WPR profiles

WPR profiles are used to collect information on various aspects and behaviors of your app.

In the following image, the Windows Performance Recorder window is shown with the "CPU usage" profile (CPU utilization for each CPU on the system) and "XAML activity" profile (events from XAML-related providers, such as WinUI) selected.

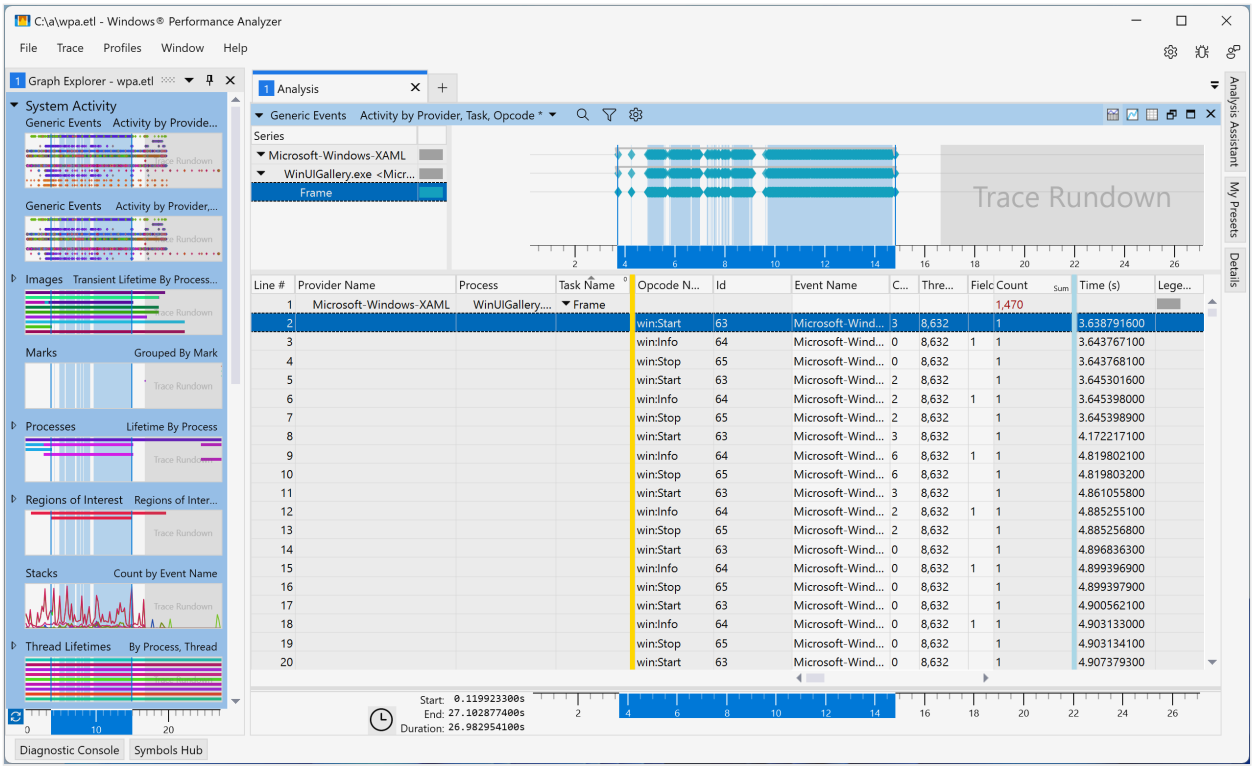


How do I use the Windows Performance Analyzer with WinUI apps?

WinUI is a declarative, retained-mode API where the app describes a tree of UIElements and WinUI runs layout and renders it. This is done on the UI thread in batches called "frames", which should complete quickly, ideally within one refresh interval of the display. When frames run long, not only does it delay updates from making it to the display, but it also prevents the UI thread from handling input. Slow frames, while not the only reason for responsiveness problems, are one of the most common.

Install the "XAML Frame Analysis" plugin

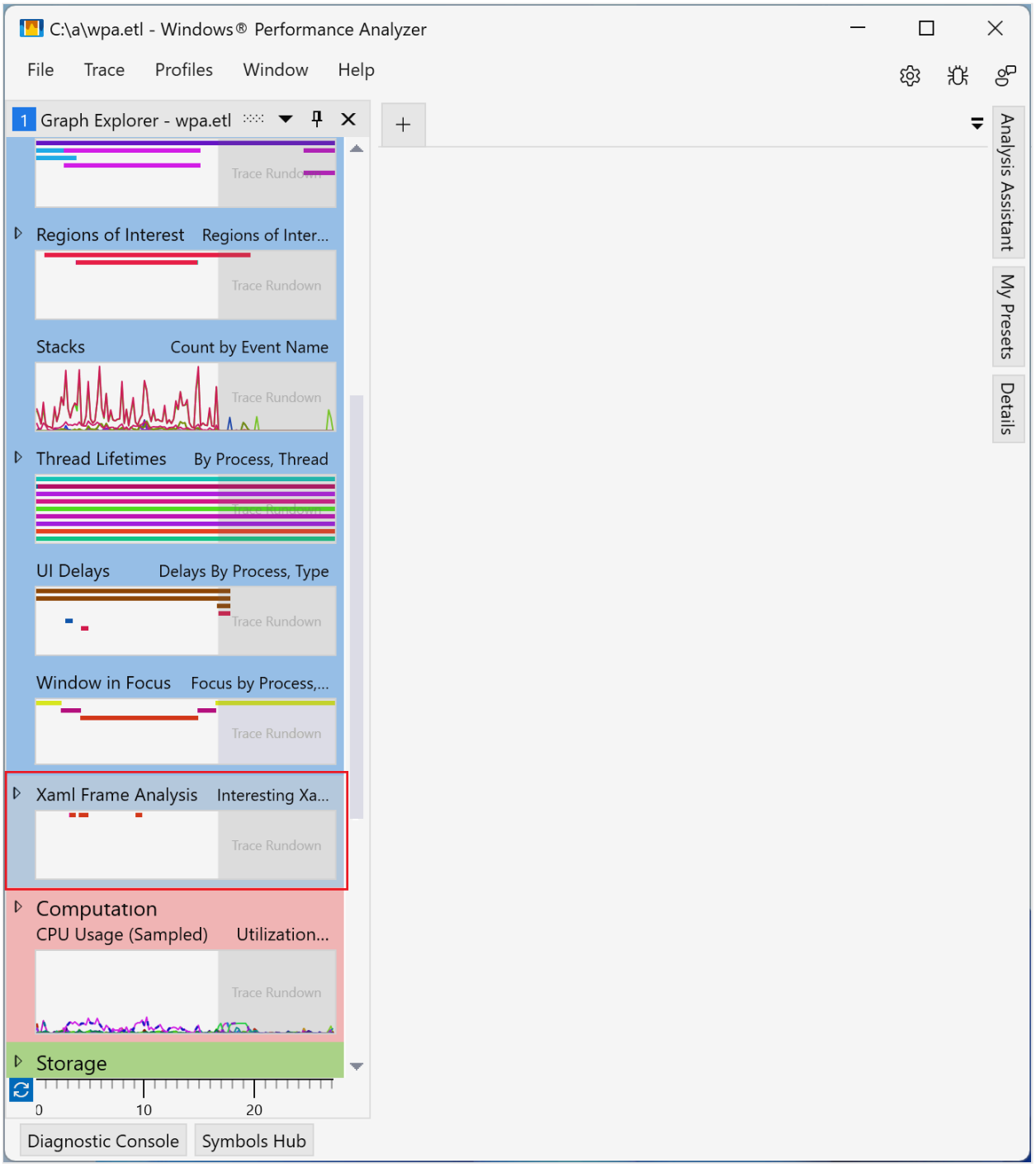
WinUI logs ETW events that track the start and stop of each frame (shown in the following screenshot of the WPA "Generic Events" table). However, because the duration of each frame needs to be calculated manually, it's difficult to identify slow frame occurrences.



To address this issue, a new "XAML Frame Analysis" table plugin is included with the Windows Assessment Toolkit (ADK) preview, build 26020 and later. This table calculates and shows the duration of each frame (along with other time-consuming operations).

> **Note**
>
> While only the preview version of the Windows Performance Analyzer (WPA) has the "XAML Frame Analysis" table, the version of WPR used to take the trace does not matter.

Once the ADK preview is installed, the "XAML Frame Analysis" table must be enabled by editing the "perfcore.ini" config file in the WPA folder (typically, C:\Program Files (x86)\Windows Kits\10\Windows Performance Toolkit). To do this, close any open instances of WPA, open "perfcore.ini" in a text editor, add `perf_xaml.dll` to the list of dlls, and save and close the file. Restart WPA, which should now show the "XAML Frame Analysis" graph at the bottom of the System Activity section.



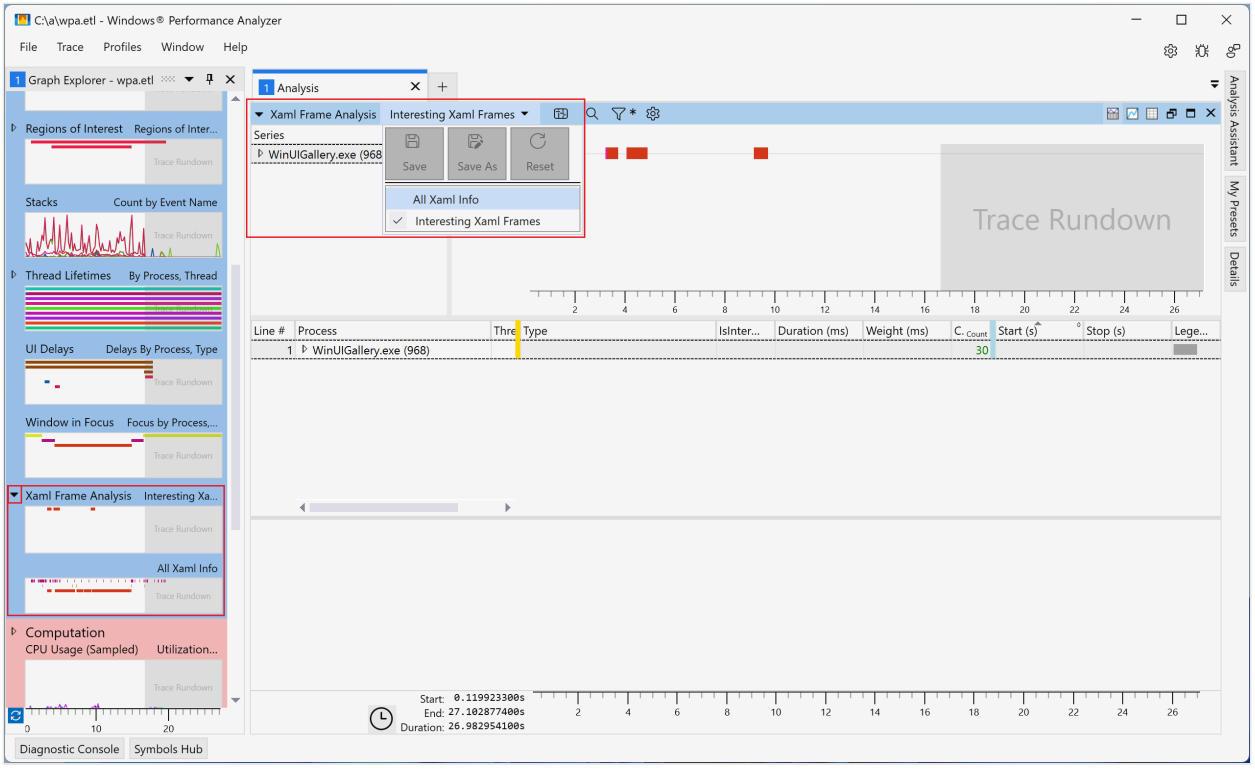
Use the "XAML Frame Analysis" plugin

The Xaml Frame Analysis supports two views (both views show the same columns):

- "Interesting Xaml Frames" (default) - Shows WinUI frames based on heuristics that identify those most likely to cause responsiveness problems. These correspond to regions that start with operations like WinUI initialization, frame navigation, or flyout display, and stop with the end of the next frame. These scenarios typically involve extensive changes to the UIElement tree and are the most susceptible to performance problems.
- "All Xaml Info" - Shows all WinUI frames from all process found in the trace. For operations like a frame or a layout pass, the plugin automatically computes and

displays the durations based on the Start and Stop events.

The following screenshot highlights how to switch between Xaml Frame Analysis views.



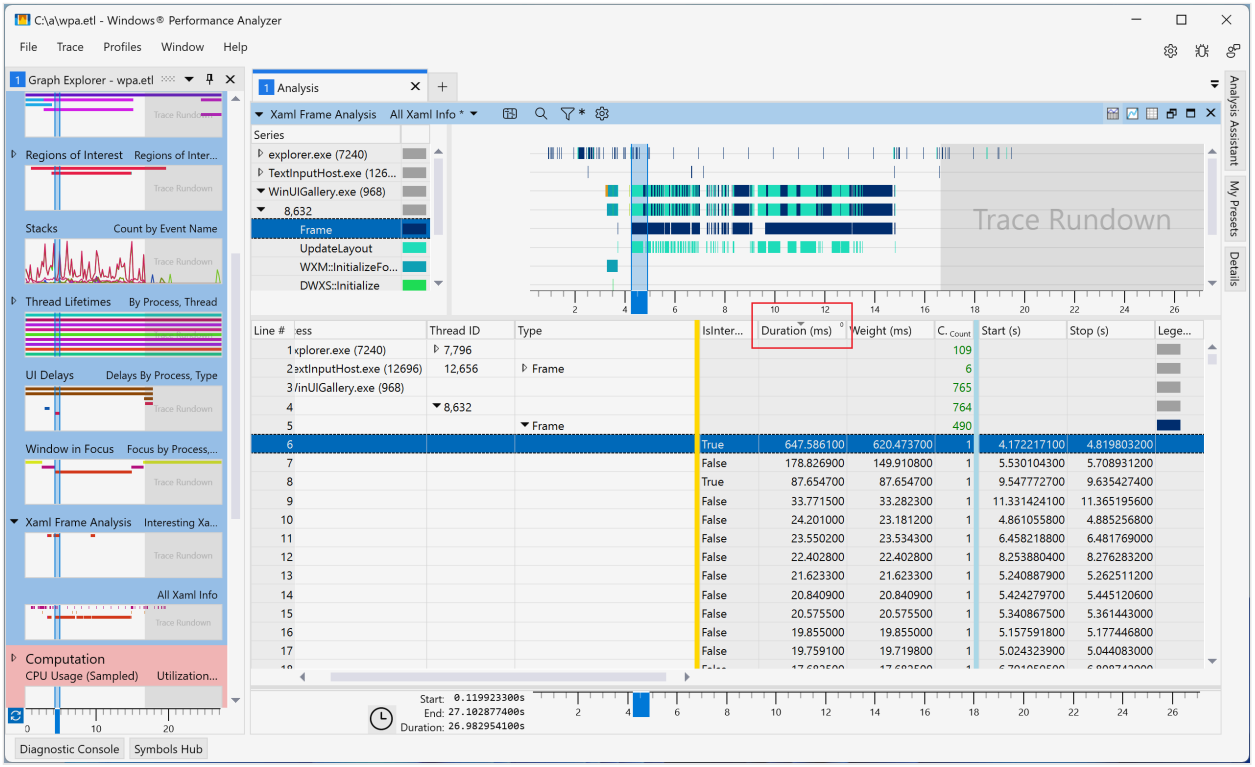
Both Xaml Frame Analysis views include the following columns:

Expand table

Title	Value
Process	Process name and ID
Thread ID	Thread ID
Type	Describes the event corresponding to the row. Possible values include: • *WXM::InitializeForCurrentThread* - A call to WindowsXamlManager.InitializeForCurrentThread. Initializes WinUI on the thread. • *DWXS::Initialize* - A call to DesktopWindowXamlSource.Initialize. Initializes a WinUI island. • *Frame* - A frame on the UI thread. Runs layout and renders changes to the tree since the previous frame. • *Create graphics device* - Create a D3D and a D2D device for WinUI. Occurs on a background thread. • *UpdateLayout* - A layout pass on the UI thread. Occurs as part of a frame, but can also be triggered by the app through UIElement.UpdateLayout. • *Frame::Navigating* - WinUI raises the Frame.Navigating event. • *Frame::Navigated* - WinUI raises the Frame.Navigated event. • *Hwnd Focus* - The active hwnd changed.

Title	Value
	• *Region of Interest* - A region computed by the plugin for interesting scenarios. Starts after events such as WinUI initialization, frame navigation, and opening menus. Stops at the end of the next frame.
IsInteresting	Whether the row is considered interesting. Only interesting rows show up in the Interesting Xaml Frames view.
Duration (ms)	The duration of the row. Computed from Start and Stop events.
Weight (ms)	The actual CPU execution time corresponding to the duration.
Start (s)	The time of the Start event
Stop (s)	The time of the Stop event

Columns can be sorted by Type or Duration to help identify potential issues such as the most expensive, longest duration frames in the trace (see following image). You can also drill down into specific rows to identify the expensive operations and potential optimizations.



Collaborate with us on GitHub

Windows developer feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Security and identity

Article • 03/19/2025

This article provides an index of development features that are related to scenarios involving security and identity in Windows apps.

Windows OS features

Windows provides a wide variety of APIs related to security and identity scenarios for apps. These features are available via a combination of Windows App SDK, Windows Runtime (WinRT), and Win32 (C++ and COM) APIs provided by the Windows SDK.

Windows App SDK APIs

The Windows App SDK provides APIs related to OAuth 2.0 authorization flows. There are also a few helper APIs in the Microsoft.Windows.Security.AccessControl namespace. These APIs are related to named object sharing between packaged apps and Win32 applications.

Expand table

Article	Description
Implement OAuth 2.0 functionality in Windows apps	The new OAuth2Manager in Windows App SDK enables desktop applications such as WinUI to seamlessly perform OAuth 2.0 authentication in Windows apps. This article describes how to implement OAuth 2.0 with the Windows App SDK.

WinRT APIs

The following articles provide information about features available via WinRT APIs provided by the Windows SDK.

Expand table

Article	Description
Security	Learn about the breadth of security features for Windows apps.
Authentication and user identity	Windows apps have several options for user authentication, ranging from simple single sign-on (SSO) using Web authentication broker to highly

Article	Description
	secure two-factor authentication.
Credential locker	This article describes how Windows apps can use the Credential Locker to securely store and retrieve user credentials, and roam them between devices with the user's Microsoft account.
Cryptography	Learn about cryptography features available to Windows apps.
Fingerprint biometrics	This article explains how to add fingerprint biometrics to your Windows app, including a request for fingerprint authentication when the user must consent to a particular action increases the security of your app.
Share certificates between apps	This article shows how you can authenticate multiple Windows apps using the same certificate, and how you can provide a method for users to import a certificate that was provided for access to secured web services.
Smart cards	This topic explains how packaged Windows apps can use smart cards to connect users to secure network services.
Windows Hello	This article describes the Windows Hello technology and discusses how developers can implement this technology to protect their apps and backend services. It highlights specific capabilities of Windows Hello that help mitigate threats from conventional credentials and provides guidance about designing and deploying these technologies as part of your packaged Windows apps.
Create a Windows Hello login app	Part 1 of a complete walkthrough on how to create a packaged Windows app that uses Windows Hello as an alternative to traditional username and password authentication systems.
Create a Microsoft Passport login service	Part 2 of a complete walkthrough on how to use Windows Hello as an alternative to traditional username and password authentication systems in packaged Windows apps.

Win32 (C++ and COM) APIs

The following articles provide information about features available via Win32 (C++ and COM) APIs provided by the Windows SDK.

Expand table

Article	Description
Security and identity	Learn about the breadth of security features available to Windows apps via Win32 APIs.
Authentication	Learn about authentication features available via Win32 APIs.

Article	Description
Cryptography	Learn about cryptography features available via Win32 APIs.

.NET features

The .NET SDK also provides APIs related to security and identity scenarios for WPF and Windows Forms apps. The security and cryptography APIs in .NET can also be used in C# WinUI apps.

Expand table

Article	Description
Security in .NET	Learn about security concepts and features for all .NET apps.
Security (WPF)	Learn about security concepts and features for WPF apps.
Windows Forms Security	Learn about security concepts and features for Windows Forms apps.

Other features

The following articles provide information about features related to security and identity scenarios with passkeys for Windows apps.

Expand table

Topic	Description
Intro to passkeys	Passkeys are simpler, stronger, passwordless sign-ins.
Implement passkeys	Describes how to implement passkey sign-ins across online, enterprise, and government applications, and for payments.
Design guidelines for passkeys	There are 14 design patterns for passkeys. You can get started with two essential patterns, and then add optional patterns to your passkeys deployment based on your unique business needs.
Use cases for passkeys	This topic describes some use cases for passkeys.
Tools and libraries for passkeys	This topic contains info about tools and libraries to help you implement passkeys.
Reference for passkeys	This topic offers some reference info, demos, and examples.

Related content

- App lifecycle and system services
- Develop Windows desktop apps

Credential locker for Windows apps

Article • 08/09/2024

This article describes how Windows apps can use the Credential Locker to securely store and retrieve user credentials, and roam them between devices with the user's Microsoft account.

The Windows Runtime (WinRT) APIs for Credential Locker access are part of the Windows Software Development Kit (SDK). These APIs were created for use in Universal Windows Platform (UWP) apps, but they can also be used in WinUI apps or in packaged desktop apps, including WPF and Windows Forms. For more information about using WinRT APIs in your Windows desktop app, see Call Windows Runtime APIs in desktop apps.

Overview of the sample scenario

For example, you have an app that connects to a service to access protected resources such as media files, or social networking. Your service requires login information for each user. You've built UI into your app that gets the username and password for the user, which is then used to log the user into the service. Using the Credential Locker API, you can store the username and password for your user and easily retrieve them and log the user in automatically the next time they open your app, regardless of what device they're on.

User credentials stored in the Credential Locker do *not* expire, are *not* affected by the ApplicationData.RoamingStorageQuota, and will *not* be cleared out due to inactivity like traditional roaming data. However, you can only store up to 20 credentials per app in the Credential Locker.

Credential Locker works a little differently for domain accounts. If there are credentials stored with your Microsoft account, and you associate that account with a domain account (such as the account that you use at work), your credentials will roam to that domain account. However, any new credentials added when signed on with the domain account won't roam. This ensures that private credentials for the domain aren't exposed outside of the domain.

Storing user credentials

1. Obtain a reference to the Credential Locker using the PasswordVault object from the Windows.Security.Credentials namespace.

2. Create a PasswordCredential object that contains an identifier for your app, the username and the password, and pass that to the PasswordVault.Add method to add the credential to the locker.

```cs
var vault = new Windows.Security.Credentials.PasswordVault();
vault.Add(new Windows.Security.Credentials.PasswordCredential(
    "My App", username, password));
```

Retrieving user credentials

You have several options for retrieving user credentials from the Credential Locker after you have a reference to the PasswordVault object.

- You can retrieve all the credentials the user has supplied for your app in the locker with the PasswordVault.RetrieveAll method.
- If you know the username for the stored credentials, you can retrieve all the credentials for that username with the PasswordVault.FindAllByUserName method.
- If you know the resource name for the stored credentials, you can retrieve all the credentials for that resource name with the PasswordVault.FindAllByResource method.
- Finally, if you know both the username and the resource name for a credential, you can retrieve just that credential with the PasswordVault.Retrieve method.

Let's look at an example where we have stored the resource name globally in an app and we log the user on automatically if we find a credential for them. If we find multiple credentials for the same user, we ask the user to select a default credential to use when logging on.

```cs
private string resourceName = "My App";
private string defaultUserName;

private void Login()
{
    var loginCredential = GetCredentialFromLocker();

    if (loginCredential != null)
    {
        // There is a credential stored in the locker.
        // Populate the Password property of the credential
        // for automatic login.
        loginCredential.RetrievePassword();
    }
```

```
    else
    {
        // There is no credential stored in the locker.
        // Display UI to get user credentials.
        loginCredential = GetLoginCredentialUI();
    }

    // Log the user in.
    ServerLogin(loginCredential.UserName, loginCredential.Password);
}

private Windows.Security.Credentials.PasswordCredential
GetCredentialFromLocker()
{
    Windows.Security.Credentials.PasswordCredential credential = null;

    var vault = new Windows.Security.Credentials.PasswordVault();

    IReadOnlyList<PasswordCredential> credentialList = null;

    try
    {
        credentialList = vault.FindAllByResource(resourceName);
    }
    catch(Exception)
    {
        return null;
    }

    if (credentialList.Count > 0)
    {
        if (credentialList.Count == 1)
        {
            credential = credentialList[0];
        }
        else
        {
            // When there are multiple usernames,
            // retrieve the default username. If one doesn't
            // exist, then display UI to have the user select
            // a default username.
            defaultUserName = GetDefaultUserNameUI();

            credential = vault.Retrieve(resourceName, defaultUserName);
        }
    }

    return credential;
}
```

Deleting user credentials

Deleting user credentials in the Credential Locker is also a quick, two-step process.

1. Obtain a reference to the Credential Locker using the PasswordVault object from the Windows.Security.Credentials namespace.
2. Pass the credential you want to delete to the PasswordVault.Remove method.

```cs
var vault = new Windows.Security.Credentials.PasswordVault();
vault.Remove(new Windows.Security.Credentials.PasswordCredential(
    "My App", username, password));
```

Best practices

Only use the credential locker for passwords and not for larger data blobs.

Save passwords in the credential locker only if the following criteria are met:

- The user has successfully signed in.
- The user has opted to save passwords.

Never store credentials in plain-text using app data or roaming settings.

Related content

- PasswordVault
- Call Windows Runtime APIs in desktop apps

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Fingerprint biometrics

Article • 08/09/2024

This article explains how to add fingerprint biometrics to your Windows app, including a request for fingerprint authentication when the user must consent to a particular action increases the security of your app. For example, you could require fingerprint authentication before authorizing an in-app purchase, or access to restricted resources. Fingerprint authentication is managed using the UserConsentVerifier class in the Windows.Security.Credentials.UI namespace.

The Windows Runtime (WinRT) APIs for fingerprint biometrics are part of the Windows Software Development Kit (SDK). These APIs were created for use in Universal Windows Platform (UWP) apps, but they can also be used in WinUI apps or in packaged desktop apps, including WPF and Windows Forms. For more information about using WinRT APIs in your Windows desktop app, see Call Windows Runtime APIs in desktop apps.

Check the device for a fingerprint reader

To find out whether the device has a fingerprint reader, call UserConsentVerifier.CheckAvailabilityAsync. Even if a device supports fingerprint authentication, your app should still provide users with an option in Settings to enable or disable it.

cs

```
public async System.Threading.Tasks.Task<string>
CheckFingerprintAvailability()
{
    string returnMessage = "";

    try
    {
        // Check the availability of fingerprint authentication.
        var ucvAvailability = await
Windows.Security.Credentials.UI.UserConsentVerifier.CheckAvailabilityAsync()
;

        switch (ucvAvailability)
        {
            case
Windows.Security.Credentials.UI.UserConsentVerifierAvailability.Available:
                returnMessage = "Fingerprint verification is available.";
                break;
            case
Windows.Security.Credentials.UI.UserConsentVerifierAvailability.DeviceBusy:
```

```
                returnMessage = "Biometric device is busy.";
                break;
            case
Windows.Security.Credentials.UI.UserConsentVerifierAvailability.DeviceNotPre
sent:
                returnMessage = "No biometric device found.";
                break;
            case
Windows.Security.Credentials.UI.UserConsentVerifierAvailability.DisabledByPo
licy:
                returnMessage = "Biometric verification is disabled by
policy.";
                break;
            case
Windows.Security.Credentials.UI.UserConsentVerifierAvailability.NotConfigure
dForUser:
                returnMessage = "The user has no fingerprints registered.
Please add a fingerprint to the " +
                                "fingerprint database and try again.";
                break;
            default:
                returnMessage = "Fingerprints verification is currently
unavailable.";
                break;
        }
    }
    catch (Exception ex)
    {
        returnMessage = $"Fingerprint authentication availability check
failed: {ex.ToString()}";
    }

    return returnMessage;
}
```

Request consent and return results

1. To request user consent from a fingerprint scan, call the UserConsentVerifier.RequestVerificationAsync method. For fingerprint authentication to work, the user must have previously added a fingerprint "signature" to the fingerprint database.
2. When you call the UserConsentVerifier.RequestVerificationAsync, the user is presented with a modal dialog requesting a fingerprint scan. You can supply a message to the **UserConsentVerifier.RequestVerificationAsync** method that will be displayed to the user as part of the modal dialog, as shown in the following image.

```
cs
```

```
private async System.Threading.Tasks.Task<string> RequestConsent(string
userMessage)
{
    string returnMessage;

    if (String.IsNullOrEmpty(userMessage))
    {
        userMessage = "Please provide fingerprint verification.";
    }

    try
    {
        // Request the logged on user's consent via fingerprint swipe.
        var consentResult = await
Windows.Security.Credentials.UI.UserConsentVerifier.RequestVerificationAsync
(userMessage);

        switch (consentResult)
        {
            case
Windows.Security.Credentials.UI.UserConsentVerificationResult.Verified:
                returnMessage = "Fingerprint verified.";
                break;
            case
Windows.Security.Credentials.UI.UserConsentVerificationResult.DeviceBusy:
                returnMessage = "Biometric device is busy.";
                break;
            case
Windows.Security.Credentials.UI.UserConsentVerificationResult.DeviceNotPrese
nt:
                returnMessage = "No biometric device found.";
                break;
            case
Windows.Security.Credentials.UI.UserConsentVerificationResult.DisabledByPoli
cy:
                returnMessage = "Biometric verification is disabled by
policy.";
                break;
            case
Windows.Security.Credentials.UI.UserConsentVerificationResult.NotConfiguredF
orUser:
                returnMessage = "The user has no fingerprints registered.
Please add a fingerprint to the " +
                                "fingerprint database and try again.";
                break;
            case
Windows.Security.Credentials.UI.UserConsentVerificationResult.RetriesExhaust
ed:
                returnMessage = "There have been too many failed attempts.
Fingerprint authentication canceled.";
                break;
            case
Windows.Security.Credentials.UI.UserConsentVerificationResult.Canceled:
                returnMessage = "Fingerprint authentication canceled.";
```

```
                break;
            default:
                returnMessage = "Fingerprint authentication is currently
unavailable.";
                break;
        }
    }
    catch (Exception ex)
    {
        returnMessage = $"Fingerprint authentication failed:
{ex.ToString()}";
    }

    return returnMessage;
}
```

Related content

- UserConsentVerifier
- Windows.Security.Credentials.UI namespace
- Call Windows Runtime APIs in desktop apps

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Intro to passkeys

Article • 09/09/2024

Learn about simpler, stronger, passwordless sign-ins. Passkeys can use your fingerprint or other biometric to log you into your websites and apps, just like unlocking your device.

Passkeys—based on Fast IDentity Online (FIDO) standards—are a replacement for passwords. Using a passkey to sign in to websites and apps across your devices is faster, easier, and more secure. Unlike a password, a passkey is always strong and phishing-resistant.

Some aspects of passkeys:

- Simplify account registration for apps and websites.
- Are easy to use.
- Work across most of your devices.
- And even work on other devices in physical proximity.

Why passkeys?

The motivation for using passkeys is, initially, an acknowledgment of the following considerations related to passwords.

Some aspects of passwords:

- They're knowledge-based.
- Can be difficult to use and to remember.
- Can be easy to phish, harvest, and replay (a very high percentage of organizations experience phishing attacks each year).
- Are a root cause of the vast majority of data breaches.

And also consider:

- There's a labor cost involved in each password reset.
- Nearly all users have a large number of online accounts.
- Most passwords are reused.

So passkeys are the answer to those issues. Passkeys address the security problems, and they're suitable for large-scale use by consumers.

Passkey protocols don't provide information that can be used by different online services to collaborate and track a user across the services. Biometric information, if

used, never leaves the user's device. With passkeys, there are no shared secrets.

Passkeys optimize access and usability for authentication

Your organization can deploy passkeys across a variety of use cases, including users signing in to your Windows apps, and users authenticating themselves to your websites. Passkeys enable users to access their sign-in credentials on many of their devices. Device-bound passkeys are bound to a FIDO security key or platform.

How do users use passkeys?

A relying party (RP)—such as a website—that wants to use passkeys first sets up a credential with you. So a user can first get a passkey from the website (instead of or in addition to a password), and then subsequently when the user wants to sign in, they can approve the use of their passkey for that site.

When a user signs in to an app or to a website by using a passkey, they approve the sign-in with the same biometric or PIN that they use to unlock the device (phone, computer, or security key). The app or website can use that mechanism instead of the traditional username and password. For example, if your passkey is stored with Windows Hello, then you can use your face, fingerprint, or PIN that you used with Windows Hello, to approve the use of that passkey.

The workflow is described in more detail in Implement passkeys and in Design guidelines for passkeys.

The benefits

Here's what that means for:

- **User experience**. The user experience (UX) will be familiar and consistent across many of your users' devices—a simple verification of their fingerprint or face, or a device PIN. The same simple action that consumers take multiple times each day to unlock their devices.
- **Security**. Passkeys are based on FIDO Authentication, which is proven to be resistant to threats of phishing, credential stuffing, and other remote attacks. Also, service providers can offer passkeys without needing passwords as an alternative sign-in or account recovery method.

Passkeys reduce the risk of phishing, and eliminate credential reuse. They also help to speed up sign-ins, improve service delivery, and lower costs. Faster, more secure, and more successful sign-ins means less breach risk, more transactions, better service delivery, and fewer credential resets and helpdesk calls for account recovery.

The FIDO passkey logo

Look for the FIDO passkey logo on sites that allow you to use passkeys instead of passwords. If you're a service provider interested in using this logo, then visit FIDO Trademark and Service Mark Usage Agreement for Websites to agree to terms, and to download the logo files.

Next steps

Next, see Implement passkeys.

Further info

- Implement passkeys
- Passkeys on the FIDO Alliance website
- Sign In with a Passkey on the FIDO Alliance website

Implement passkeys

Article • 09/09/2024

Also see Intro to passkeys

This topic describes how to implement passkey sign-ins across online, enterprise, and government applications, and for payments. You can implement passkeys (instead of passwords) to provide your websites and apps with user-friendly and cryptographically secure sign-ins—all courtesy of the public WebAuthn API.

Consumer service providers, enterprises, and governments around the world are moving from forms authentication (such as passwords and SMS OTPs) to passkey-based sign-ins. Every organization has its own varying use cases and business requirements.

How passkeys work

Passkeys use standard public key cryptography techniques. When a user registers with an online service, the user's client device creates a new cryptographic key pair that is bound to the web service domain. The device retains the private key, and registers the public key with the online service. These cryptographic key pairs, called passkeys, are unique and bound to the online service domain.

The way authentication works is that the user's device must prove possession of the private key by presenting a challenge for sign-in to be completed. This occurs after the user approves the sign-in locally on their device, via quick and easy entry of a biometric (such as a thumbprint), or local PIN, or touch of a FIDO security key. Sign-in is completed via a challenge-response from the user's device and the online service. The service doesn't see or ever store the private key.

To enroll a passkey with an online service:

- The user is prompted to create a passkey.
- The user verifies the passkey creation via a local authentication method (for example, biometrics).
- The user's device creates a new public/private key pair (a passkey) that's unique for the local device, the online service, and the user's account.
- The public key (and *only* that) is sent to the online service, and is associated with the user's account.

To subsequently sign in by using a passkey:

- The user is prompted to sign in by using a passkey.

- The user verifies the sign-in-with-passkey via a local authentication method (for example, biometrics).
- The device uses the user's account identifier (provided by the service) to select the correct key, and sign the service's challenge.
- The client device sends the signed challenge back to the service, which verifies it with the stored public key, and signs the user in.

Implement passkeys for consumers, enterprise, government, or payments

This section is for you if you're considering implementing passkeys.

If you're new to FIDO, then we recommend that you first review the User Authentication Specifications Overview. Then, you can access the latest versions of the FIDO Alliance user authentication specifications at Download Authentication Specifications.

The FIDO Certified Showcase highlights FIDO Alliance members and their FIDO Certified solutions. It's a great resource if you're looking to deploy FIDO.

The FIDO Enterprise Deployment Working Group (EDWG) has developed a series of white papers that provide guidance for leaders and practitioners considering passkeys—scaling from SMBs to large enterprises. To understand the key decision points, see Enterprise.

If your field is along the lines of citizen services or government employee applications, then see Government.

For more info about payment scenarios that are ideal for passkeys, see Payments.

Next steps

Next, see Design guidelines for passkeys.

Further info

- Intro to passkeys
- WebAuthn APIs for passwordless authentication on Windows
- Passkeys.dev
- Get Started on Your Passwordless Journey on the FIDO Alliance website

Design guidelines for passkeys

Article • 09/09/2024

There are 14 design patterns for passkeys. You can get started with two essential patterns, and then add optional patterns to your passkeys deployment based on your unique business needs.

Essential patterns

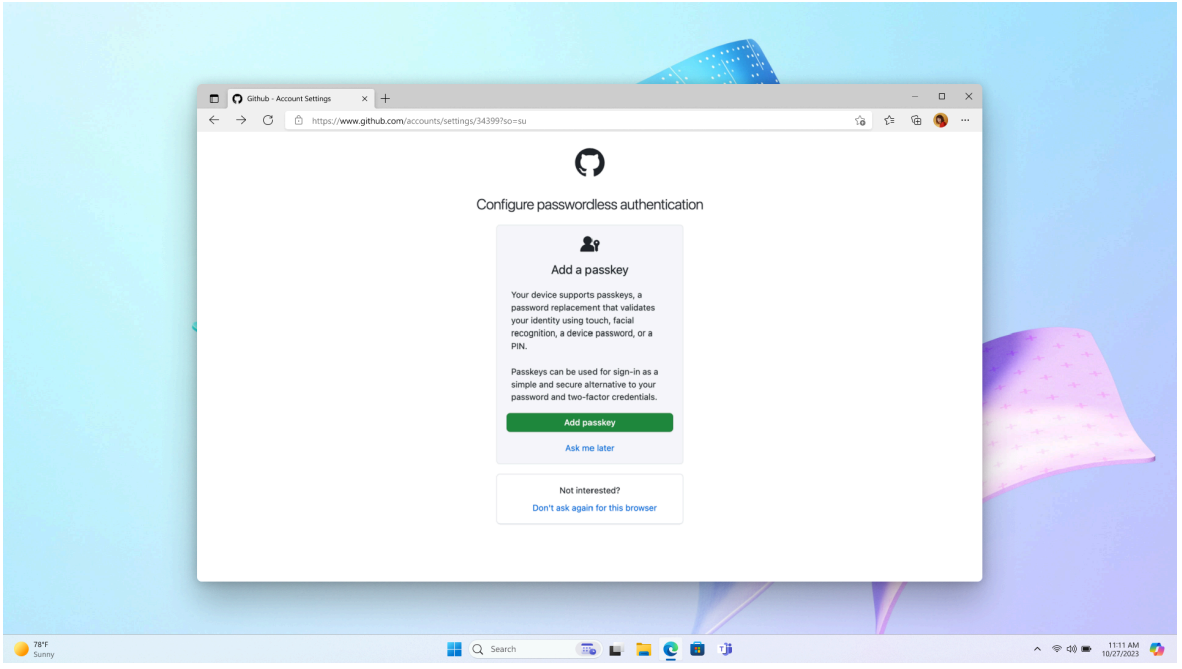
The two essential (foundational) patterns are that:

- A user needs first to *create* a passkey (and view/manage it),
- and then the user can *sign in* with it.

Every rollout of passkeys should make use of these patterns.

Pattern 1: Create, view, and manage passkeys in account settings

- If a passkey doesn't exist for the current user, then display the passkey creation prompt, like this:

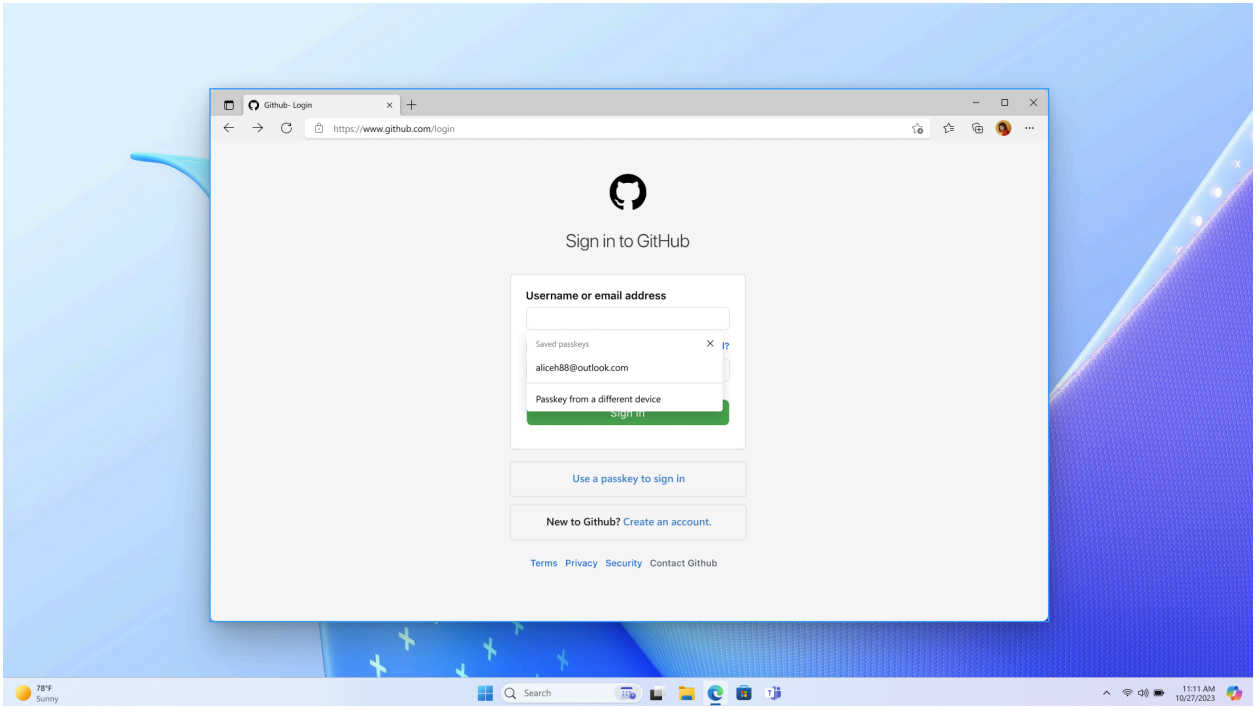


Pattern 2: Sign in with a passkey

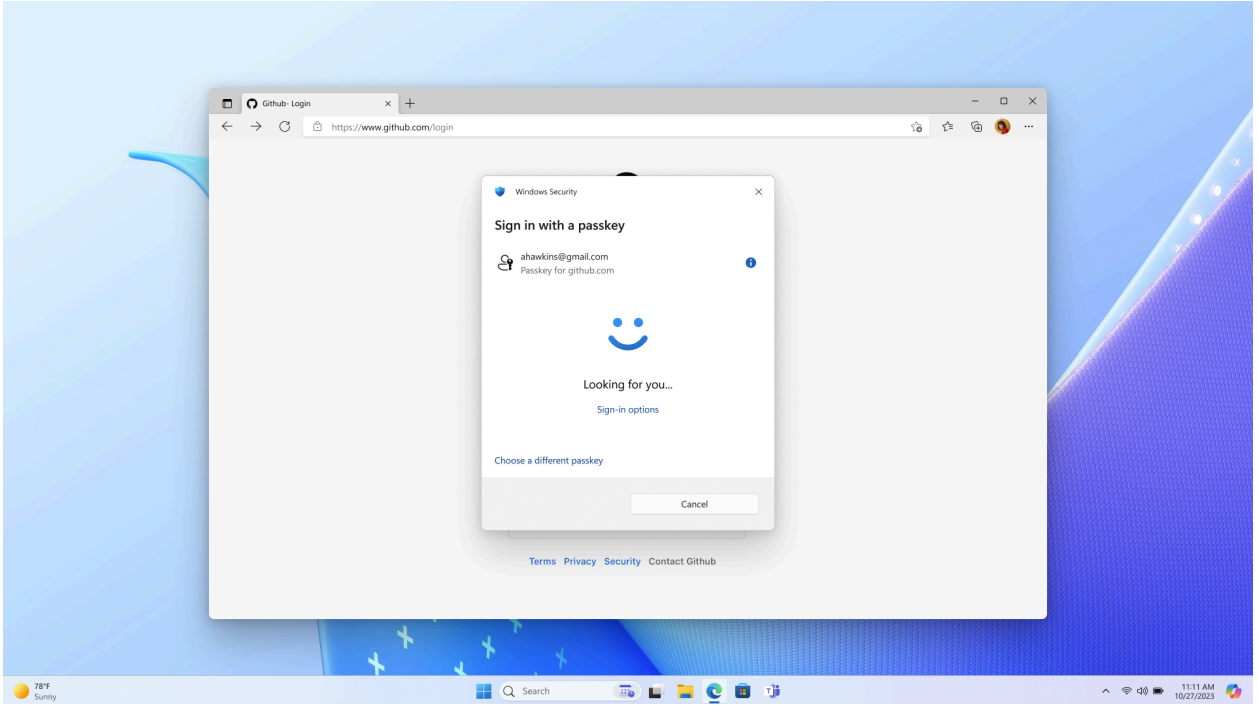
- Enable autofill by adding `autocomplete="webauthn"` to your **username** input field.

- Support graceful fallback to other sign-in methods. Allow users to enter some different credential (not a passkey) to sign in or to create an account.
- Research explored users' success and satisfaction with: a dedicated "Sign in with a passkey" link, buttons, and autofill. Testing indicated that autofill ensured the highest success for people to sign in with a passkey.

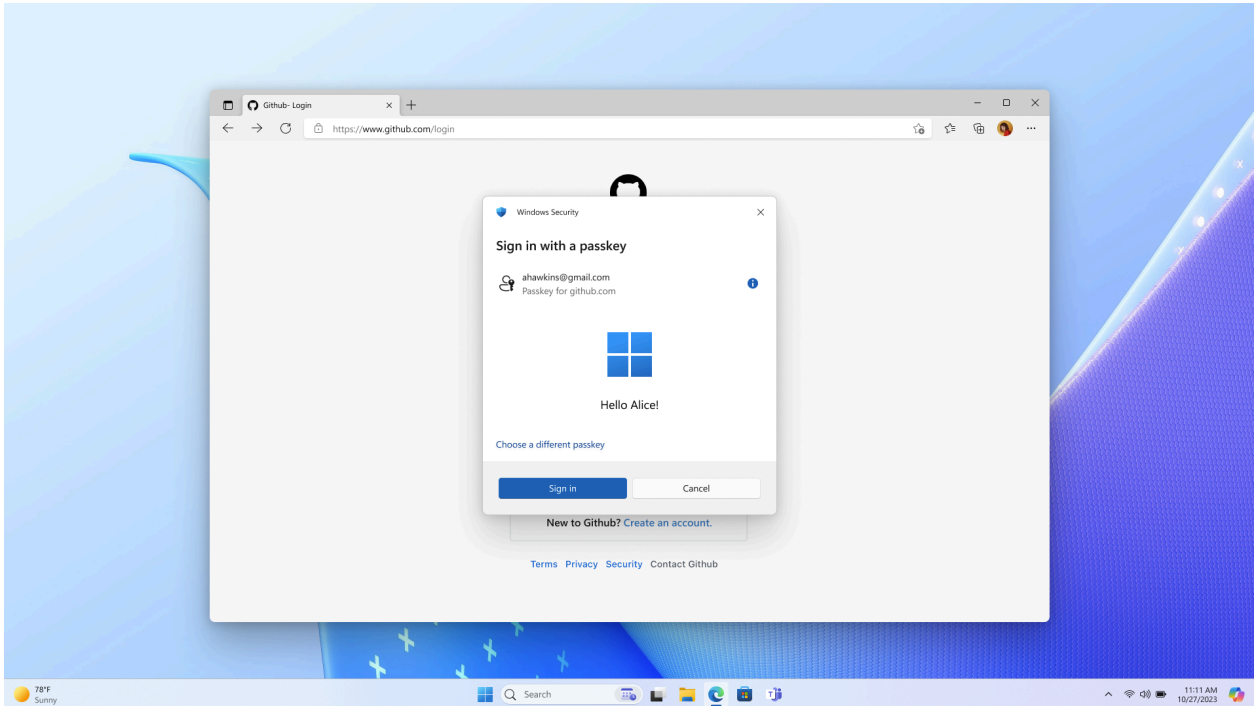
Step 2.1: Login page



Step 2.2: Passkey sign-in prompt with lock screen



Step 2.3: Login success with Windows Hello



UI text, research, and rollout

Here are some user-tested button labels and phrases. Copy and edit them to suit your needs.

With passkeys, you don't need to remember complex passwords.

What are passkeys? Passkeys are encrypted digital keys you create using your fingerprint, face, or screen lock.

Where are passkeys saved? Passkeys are saved in your password manager, so you can sign in on other devices.

Research indicates that your users will appreciate the option to create a passkey for their account *at any point* within the account settings workflow. Candidates described an account settings UI as an intuitive and useful place to proactively learn about and update their authentication settings.

The research also indicates that users seek to understand the nature and value of passkeys. Vague or technical explanations often confuse them and lead them to abandon creating a passkey. Familiar biometric iconography, and brief messaging that compares passkeys to known technologies, is most effective at providing the right level of information. For users who already have a passkey, the passkey card(s) afford them an unmistakable object in the interface that they later visualize and then find in your account settings UI. They use that to get actionable information about the passkey.

On your homepage, we recommend that you offer one affordance for both sign-in and account creation. Research indicates that the action of creating an account is more discoverable that way, rather than surfacing the account creation option *after* the user has signed in. Also, some users report that occasionally they're uncertain whether they have an account—so a multipurpose button serves their needs.

Rollout

Passkeys might require specific hardware or software support on users' devices. Ensure that users are aware of the compatibility requirements for using passkeys, and provide guidance on compatible devices and browsers including Windows and Edge.

In the context of native mobile apps, signing in with a passkey is different from the biometric sign-in experience that has existed for many years. Signing in with a passkey requires an additional tap.

As for security, an implementation might gracefully fall back to an email OTP. The graceful fallback option that you choose should match your unique goals for security and business. So plan your UI and UX accordingly.

For service providers, you might choose a phased rollout where initially users can create passkeys only from their account settings. Then, as you gather data about the passkey sign-in experiences unique to your users' needs, you can increase the number of places where a passkey can be created, including during account creation. And you should develop a communication plan to educate users about the availability of passkeys, and their benefits. To reach users effectively, you might leverage several communication channels (in-app notifications, email, and online help resources) that align with your unique goals.

Optional patterns

For info about optional patterns, see Optional patterns.

Next steps

Next, see Use cases for passkeys.

Further info

- Intro to passkeys
- Passkeys.dev

- Get Started on Your Passwordless Journey on the FIDO Alliance website

Use cases for passkeys

Article • 09/09/2024

This topic describes some use cases for passkeys.

Use case 1: Bootstrapping

Bootstrapping an account on the web.

1.1: Authenticating the user

This section applies when the relying party (RP) doesn't yet know who is controlling the client device. There's no browser artifact available to the RP (such as a cookie or a credential ID in local storage), although for now we assume that the user has an existing account with the RP.

To bootstrap an account, serve the user a sign-in page.

Begin by asking the user for their account identifier; typically a username or email address.

To support the autofill UI for passkeys, make sure to:

1. Add the `username` and `webauthn` value to any existing autocomplete annotations on the **username** input field.

HTML

```
<div>
  <label for="username">Username:</label>
  <input name="username" id="loginform.username"
         autocomplete="username webauthn">
</div>
```

2. On page load, use an `if` statement to check to see whether autofill UI (conditional mediation) is available, then call `navigator.credentials.get()` with `mediation: "conditional"` and `userVerification: "preferred"`.

JavaScript

```
  <script>
    (async () => {
      if (
      typeof window.PublicKeyCredential !== 'undefined'
```

```
      && typeof window.PublicKeyCredential.isConditionalMediationAvailable
=== 'function'
      ) {
        const available = await
PublicKeyCredential.isConditionalMediationAvailable();

      if (available) {
          try {
            // Retrieve authentication options for
`navigator.credentials.get()`
            // from your server.
            const authOptions = await getAuthenticationOptions();
      // This call to `navigator.credentials.get()` is "set and forget."
      // The Promise will resolve only if the user successfully interacts
      // with the browser's autofill UI to select a passkey.
      const webAuthnResponse = await navigator.credentials.get({
          mediation: "conditional",
      publicKey: {
          ...authOptions,
          // See note about userVerification below.
          userVerification: "preferred",
              }
            });
      // Send the response to your server for verification, and
      // authenticate the user if the response is valid.
      await verifyAutoFillResponse(webAuthnResponse);
          } catch (err) {
          console.error('Error with conditional UI:', err);
          }
        }
      }
    })();
  </script>
```

The above will cause the following to happen:

- Retrieve the authentication options from your server. Return at least a random `challenge` and `rpId` to be associated with this authentication request.
- When the user interacts with your **username** field, the browser and platform will check whether a passkey exists (in the platform authenticator) that can be used with the relying party.
- If that *is* the case, then the passkey will be presented to the user as an option to choose (along with other credentials that can be auto-filled, such as usernames stored in the browser's password manager). The browser/platform might render a UI similar to the one shown below. Although, the exact look and feel varies from one platform or form factor to another:

- If the user selects the passkey, then the platform UI will guide the user through an (often biometrics-based) user verification check.
- If the user successfully passes the user verification, then the `navigator.credentials.get()` call succeeds and returns a WebAuthn response.
- If the user selects a credential other than a passkey, then the browser/platform chooses a different appropriate action (such as auto-filling the username), and the `navigator.credentials.get()` call doesn't resolve.
- If the user selects the "Passkey from another device" option (the exact text will vary slightly by platform), then the browser/platform will guide the user through using a FIDO2 security key or the Cross-Device Authentication (CDA) flow to use a passkey from their smartphone or tablet to deliver a WebAuthn response to the `navigator.credentials.get()` call.
- Send the WebAuthn response to your server for verification and additional security checks. If all checks succeed, then begin an authenticated session for this user.

This is why this is called the *conditional UI* (or, more commonly, the *autofill UI*) mode of WebAuthn—the platform authenticator UI that guides the user through the verification, or through using their phone, is shown only if the user has a passkey on this device (or chooses the "another device" option).

As you can see, in this mode the `navigator.credentials.get()` call either succeeds, or it doesn't because it never resolves. If it does succeed, then the result of the call will reveal both a user id and a signed WebAuthn assertion, which the relying party (RP) will use to authenticate the user.

If the call doesn't succeed, then you should perform a *legacy* user authentication. You'll obtain a username from this first page, and in subsequent pages you then serve appropriate further login challenges (such as passwords, responding to SMS challenges, etc.) to the user. Those might include *account recovery* steps in case the user has forgotten their password, or is otherwise not able to pass the regular login challenges. Once the user has passed all the login challenges, they're considered authenticated and signed in.

When the user doesn't already have an account with the relying party (RP), you'll usually give the user the option on the sign-in page to create an account. If the user chooses that option, then you'll collect the necessary information from them to open a new account. If they successfully open a new account, then they're also considered authenticated and signed-in.

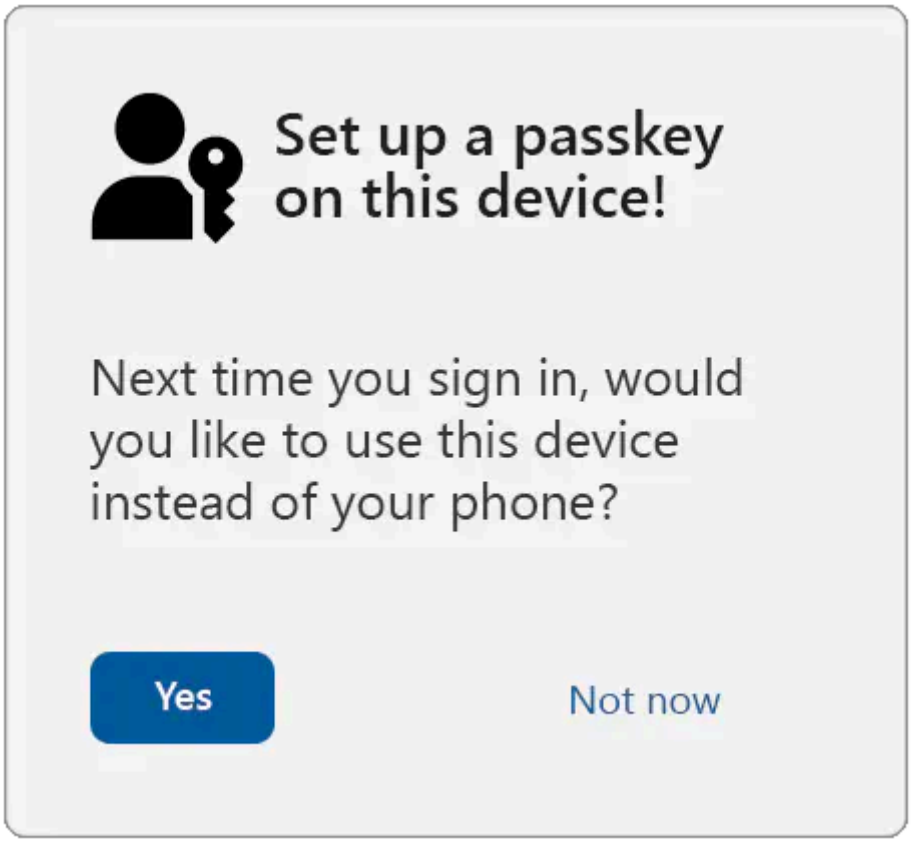
Once the user is signed in, it might be time to set up a new passkey for them. Do that for any of the following cases:

- The user bootstrapped their account on the device by passing non-passkey login challenges (such as using a password).
- The user just created a new account at the relying party (RP), and they're considered signed in because of that.
- The user was using a passkey, but they used a different device from the one they're currently using (by selecting "another device" shown in the example above). This can be confirmed by inspecting the **authenticatorAttachment** attribute in the returned **PublicKeyCredential** object.

1.2: Cross-device authentication

If the user used a passkey from another device (such as a phone, tablet, or FIDO2 security key), then the **authenticatorAttachment** property in the authentication response (**getAssertion**) will have the value `cross-platform`.

In that scenario, offer the user the choice to create a passkey on their local device. That will result in a more seamless user experience in the future, because the user won't be required to use their other device.



1.3: A note about user verification

This guidance sets **userVerification** to `preferred`, meaning that user verification will be attempted when possible.

Some devices, such as desktop computers and older laptops, might not have biometric sensors. On those devices, if **userVerification** is set to `required`, then the user might be asked to enter their system login password for each sign in using a passkey. And that can be frustrating for them.

When `preferred` is used, some platform authenticators will always require a user verification check when the device has biometric sensors, but might skip user verification on devices without them.

The user verification result (conveyed in authenticator data flags) will reflect the actual user verification result, and should always be validated against your requirements on the server.

1.4: Opt the user in to passkeys

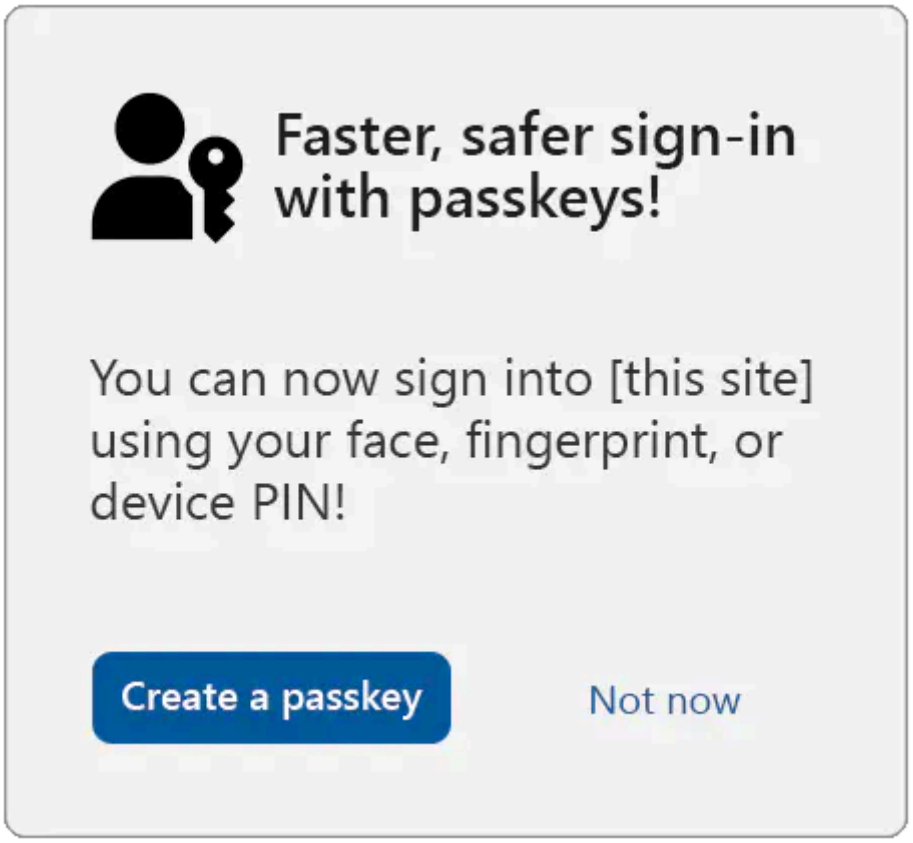
First, verify that the user is sufficiently strongly authenticated using other login methods, including multi-factor authentication.

Second, ensure that the user's device and operating system (OS) combo supports passkeys by calling:

```
JavaScript

PublicKeyCredential.isUserVerifyingPlatformAuthenticatorAvailable()
```

If passkeys are supported, then that will return `true`. If they aren't supported, then it will return `false`, and you should abort the passkey enrollment flow.

Serve an opt-in or "upsell" modal/interstitial or page to the user offering them to create a passkey:



> **Tip**
>
> In order to ensure that the user is giving fully informed consent, consider showing (or linking to) longer descriptions explaining that all users who are able to unlock the current device will be able to access the account at the relying party (RP).

If the user consents, then call `navigator.credentials.create()` with the options shown in the example below:

```
JavaScript
```

```
navigator.credentials.create({
  publicKey: {
    rp: {
      // User-friendly name of your service.
      name: "Passkeys Developer",
      // Relying party (RP) identifier (hostname/FQDN).
      id: passkeys.contoso"
    },

    user: {
      // Persistent, unique identifier for the user account in your backend.
      id: Uint8Array.from("0525bc79-5a63-4e47-b7d1-597e25f5caba", c =>
c.charCodeAt(0)),
      // User-friendly identifier often displayed to the user (for example,
email address).
      name: "amanda@contoso.com",
      // Human-readable display name, sometimes displayed by the client.
      displayName: "Amanda Brady"
    },
    // The challenge is a buffer of cryptographically random bytes generated
on your backend,
    // and should be tightly bound to the current user session.
    challenge:
Uint8Array.from("XZJscsUqtBH7ZB90t2g0EbZTZYlbSRK6lq7zlN2lJKuoYMnp7Qo2OLzD7xa
wL3s", c => c.charCodeAt(0)),
    pubKeyCredParams: [
      // An array of objects describing what public key types are acceptable
to a server.
      {
        "type": "public-key",
        "alg": -7 // EC P256
      },
      {
        "type": "public-key",
        "alg": -257 // RSA
      }
    ],
    excludeCredentials: [
      // Array of credential IDs for existing passkeys tied to the user
account.
      // This avoids creating a new passkey in an authenticator that already
has
      // a passkey tied to the user account.
      {
        // Example only.
        type: "public-key",
        id: new Uint8Array([21, 31, 56, ...]).buffer
      },
      {
        // Example only.
        type: "public-key",
        id: new Uint8Array([21, 31, 56, ...]).buffer
      }
    ],
```

```
    authenticatorSelection: {
      // Tells the authenticator to create a passkey.
      residentKey: "required",
      // Tells the client/authenticator to request user verification where
possible;
      // for example, a biometric or a device PIN.
      userVerification: "preferred"
    },
    "extensions": {
      // Returns details about the passkey.
      "credProps": true
    }
  }
})
```

> ⓘ **Note**
>
> We recommend that most relying parties (RPs) not specify the attestation conveyance parameter `attestation` (thus defaulting to none), or instead explicitly use the value `indirect`. That guarantees the most streamlined user experience (platforms are likely to obtain consent from the user for other types of attestation conveyances, which likely results in a larger fraction of unsuccessful credential creations due to users canceling the creation).

When the WebAuthn call resolves, send the response to your server, and associate the returned public key and credential ID with the previously authenticated user account.

Use case 2: Reauthentication

Using passkeys for a reauthentication might be necessary for any of the following reasons:

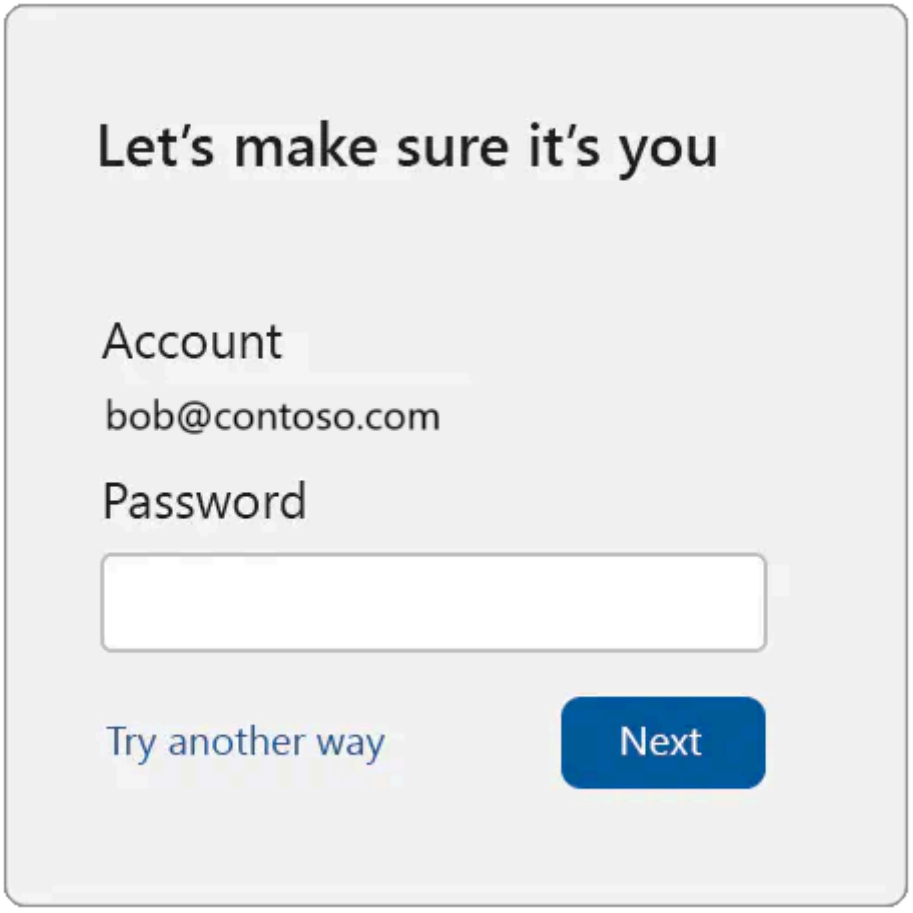
- The user signed out, and now wants to sign in again.
- The user session expired due to inactivity, and the user wants to sign in again.
- The user is about to perform a sensitive action, and needs to re-confirm control over the user session.

To reauthenticate the user in each of these situations, you'll use passkeys that you set up in the previous use case. The WebAuthn API call is the same in all three cases, but the UI treatment that you provide is slightly different. Since the particular account is specified by you, the platform won't prompt the user to select a different account on your service.

2.1: Sensitive actions

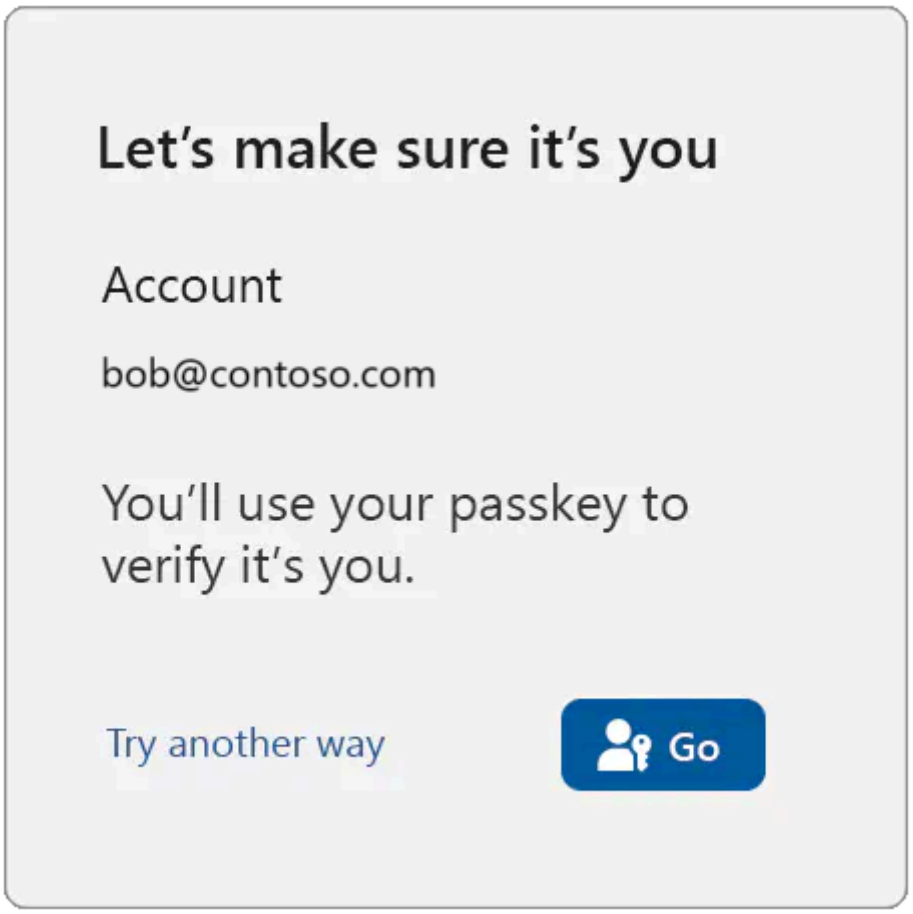
Let's look at the UI for the third reason first—when it's time to reauthenticate for a sensitive action, check whether you have a credential ID for at least one passkey for the user.

If no such credential ID is available, then serve a traditional login challenge suitable for reauthentication, for example:



> **Tip**
>
> We recommend that on this login challenge page, users can't change their account identifier. Also, the login challenge should be something that an unauthorized user of the device can't pass.

If, on the other hand, you *do* find at least one passkey credential ID for the user, then you can use passkeys for reauthentication:



When the user is ready (in the above example, when they click on the "Go" button), call `navigator.credentials.get()`, passing in all the user's passkey credential IDs:

JavaScript

```
navigator.credentials.get({
  publicKey: {
    challenge: ...,
    rpId: ...,
     allowCredentials: [{
      type: "public-key",
      id: new UInt8Array([21, 31, 56, ...]).buffer,
    }, {
      type: "public-key",
      id: new UInt8Array([21, 31, 56, ...]).buffer,
    }, {
      ...
    }],
    // see note below
    userVerification: "preferred",
  }
});
```

ⓘ Note

Be sure to read the guidance around userVerification from the previous use case.

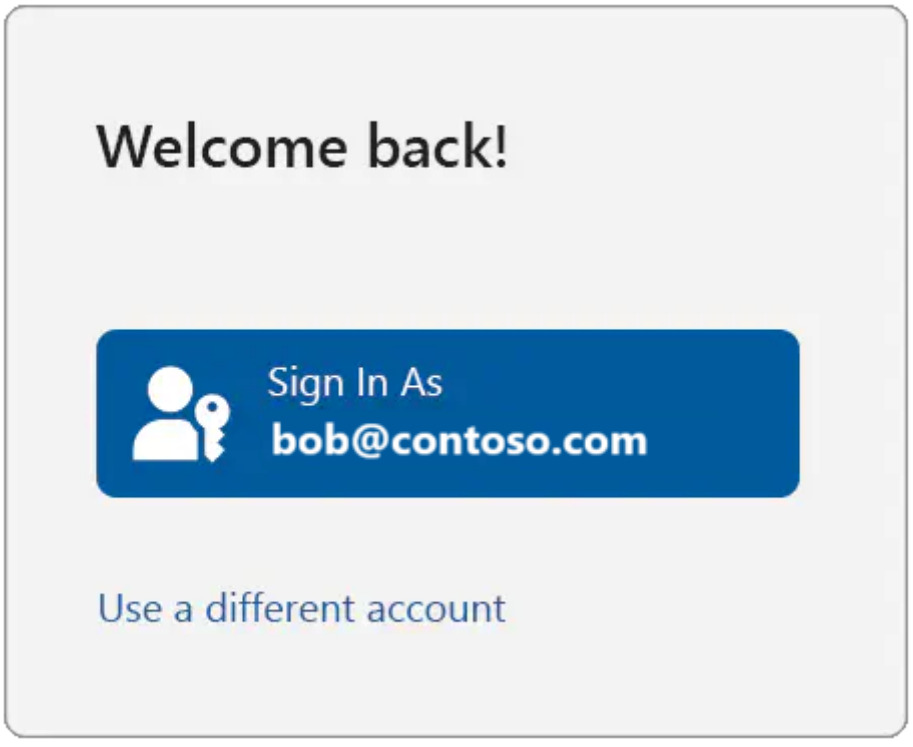
If the user instead clicks on "Try another way", then you should offer them other sign-in methods (password, etc.) to reauthenticate them (assuming that the user has such other sign-in methods available to them).

2.2: Expired sessions and logout

Now we'll examine the case where the reauthentication is triggered because the user logged themself out, or the relying party (RP) expired the user's session. To facilitate that, the RP would have to keep some form of user session state reminding them of the account that was formerly signed in, even when they consider the user signed-out (that could be achieved using browser artifacts such as cookies or local storage).

> ⓘ **Note**
>
> A relying party (RP) might choose to treat signing out as a comprehensive action, and thus delete all references to the user's identity. Such an RP should treat a subsequent sign-in like an account bootstrap, and repeat the steps explained previously.

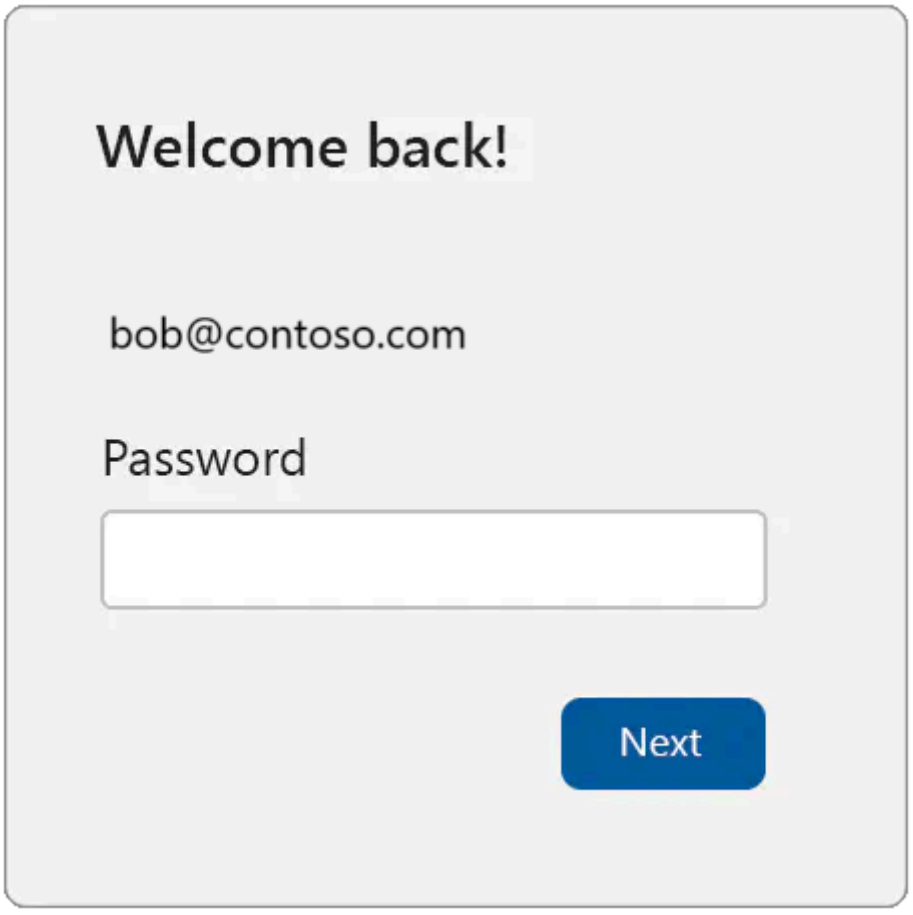
You, as the RP, might then serve a sign-in page like this:



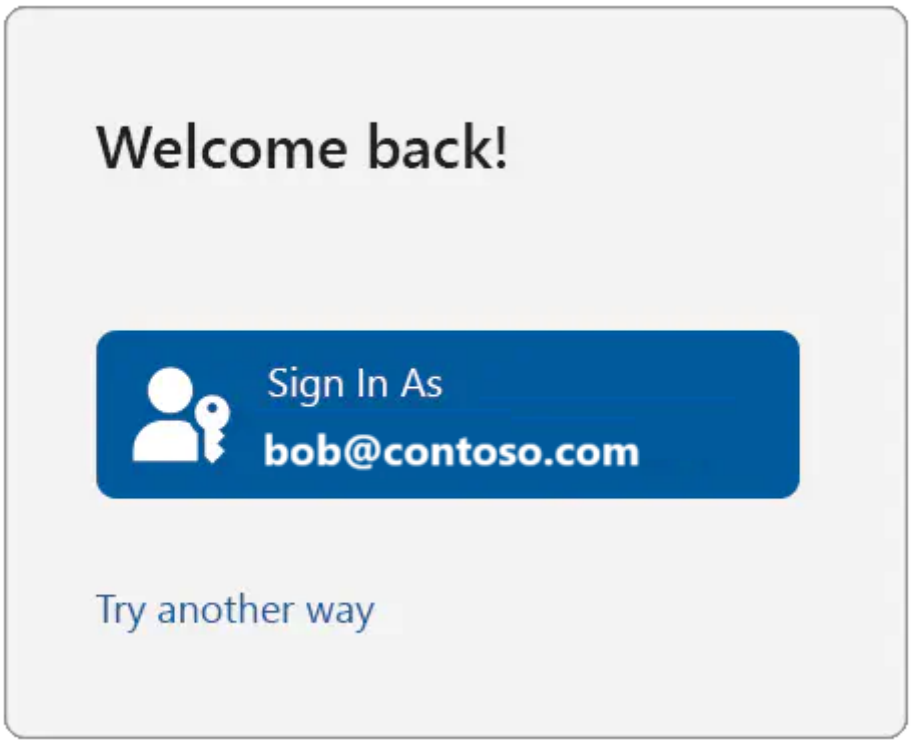
If the user clicks on "Use a different account", then you should enter an account bootstrap flow—as explained for the previous use case—repeating the steps there, where the platform will let the user select which account they want to use.

> ⓘ **Note**
>
> In that case, you should also give the user the ability to completely remove the suggested account from being listed on the sign-in page.

But if the user clicks the "Sign In As" button, then check whether you have at least one passkey credential ID associated with the user. If no credential ID is available, then serve a traditional login challenge suitable for reauthentication, for example:



If, on the other hand, you *do* find at least one passkey credential ID for the user, then you can use passkeys for reauthentication:



When the user is ready (in the above example, when they click on the "Go" button), call `navigator.credentials.get()`, exactly as already shown (that is, by passing in all the user's passkey credential IDs).

If the user instead clicks on "Try another way", then you should offer them other sign-in methods (password, etc.) to reauthenticate them (assuming that the user has such other sign-in methods available to them).

Next steps

Next, see Tools and libraries for passkeys.

Further info

- Intro to passkeys
- Passkeys.dev
- Get Started on Your Passwordless Journey on the FIDO Alliance website

Tools and libraries for passkeys

Article • 09/09/2024

This topic contains info about tools and libraries to help you implement passkeys.

Libraries

Selection criteria

If you wish to own passwordless authentication internally, or you're looking to implement a turnkey solution for passkeys, then you'll likely be looking for libraries or vendors. When selecting a library to implement passkeys, what should you as a developer at a relying party (RP) keep an eye on?

> **Note**
>
> A small set of these criteria are not specific to passkeys, but are useful to keep in mind when selecting an open-source solution.

WebAuthn versions and capabilities

- **Version**. Check which version of the spec the library supports (Level 2, Level 3, and so on).
- **Features and capabilities**. Check whether the library includes key features and capabilities for your use case.
  - Does the library help with generating registration and authentication options? Does it help with verification of the registration and authentication response? From a relying party (RP) perspective, those are the key steps of your implementation. So make sure that the library you select provides useful functions for those steps.
  - If you're considering using attestation features:
    - Does the library help leverage FIDO MDS in some way?
    - Can the library verify all attestation statement formats?

Verification steps

Check whether the library follows the necessary verification steps:

- During registration.
- During authentication.

User interface (UI) and user experience (UX)

If you're looking for a library that offers UI elements:

- **Visual consistency**. Check that the solution uses standardized icons.
- **Clear language**. Instructions using plain language are critical for broader user understanding. Prioritize solutions aligned with the FIDO UX guidelines.

Developer experience

- **Full-stack coverage**. A library that offers tightly-integrated frontend and backend components (for example, SimpleWebAuthn) can streamline your integration.
- **Developer documentation**. In order to ease the integration process, check that the library has a maintained documentation website .

Developer involvement and maintenance

- **Open-source maintenance**. For open-source options, investigate their community activity. A few active issues, or many issues with up-to-date labels (assuming that those require manual assignment), and comments by contributors, are all signals of an active community.
- **Patience**. Standards can be slow-moving. So a WebAuthn/passkey library can go a long time between updates if there aren't any real issues with it. But that doesn't mean the library is unmaintained.

Licensing

Review the solution's licensing model (for example, MIT, Apache, commercial) in the context of your project.

Updated for passkeys

- **Rust**. Webauthn-rs - Webauthn for Rust Server Applications (William Brown).
- **TypeScript**. SimpleWebAuthn (Matthew Miller).
- **Java**. java-webauthn-server (Yubico).

Other FIDO2/WebAuthn libraries

The WebAuthn Awesome GitHub repo is also regularly updated with libraries from the community.

- **.NET**. FIDO2 .NET Library (Anders Åberg, Alex Seigler).
- **Go**. WebAuthn Library.
- **Java**. WebAuthn4J (Yoshikazu Nojima).
- **Python**. py_webauthn (Duo Labs).
- **Ruby**. webauthn-ruby (Cedarcode), Devise::Passkeys (Ruby Passkeys, wrapper around webauthn-ruby), and Warden::WebAuthn (Ruby Passkeys, wrapper around webauthn-ruby).

Test sites and tools

In addition to the resources listed in the sections below, the WebAuthn Awesome GitHub repo is also regularly updated with tools and demos from the community.

Basic FIDO2/WebAuthn tools

- WebAuthn.io.
- yubico demo site.
- webauthn.

Advanced FIDO2/WebAuthn tools

- lbuchs/WebAuthn.
- WebAuthn Debugger.
- WebAuthn Developer Tool.

Next steps

Next, see Reference for passkeys.

Further info

- Intro to passkeys
- Passkeys.dev
- Get Started on Your Passwordless Journey on the FIDO Alliance website

Reference for passkeys on Windows

Article • 02/19/2025

This topic offers some reference info, demos, and examples for passkeys on Windows.

Overview

Passkeys can be used in all supported versions of Windows clients. As of the September 2023 moment, for Windows 11, version 22H2 with KB5030310 or later, native support for passkey management is available (see Support for passkeys in Windows). Windows native credential UX is displayed when a user saves or signs in with a passkey (even from the browser). Users can perform passkey management from Windows settings or Edge. Cross-device authentication through the browser, leveraging chromium-based credential UI, is still available.

Cross-Device authentication

Starting in Windows 11, version 23H2, FIDO Cross-Device Authentication (CDA) is supported globally at the operating system (OS) level, and is available for all apps and browsers. Persistent linking is available between Android devices (authenticator) and Windows 11, version 23H2, and later. iOS and iPadOS don't support persistent linking.

User-verification behavior

When a user tries to interact with a passkey on Windows 11, an available screen unlock method is used for user verification via Windows Hello. Setting up facial recognition or fingerprint recognition are optional.

Where those biometrics are not configured or available, both passkey creation and authentication fall back to requesting the Windows Hello PIN.

Demos and examples

See Windows Reference on passkeys.dev and Demo Sites.

Further info

- Intro to passkeys

- Passkeys.dev
- Get Started on Your Passwordless Journey on the FIDO Alliance website

Third-party passkey manager support

Article • 12/12/2024

This topic describes the third-party credential manager API plugin support for third-party passkey providers on Windows. It showcases a demo app named Contoso Passkey Manager.

Contoso Passkey Manager

Windows 11 supports a third-party plugin model for passkeys. Contoso Passkey Manager is a demo app that demonstrates this passkey provider support.

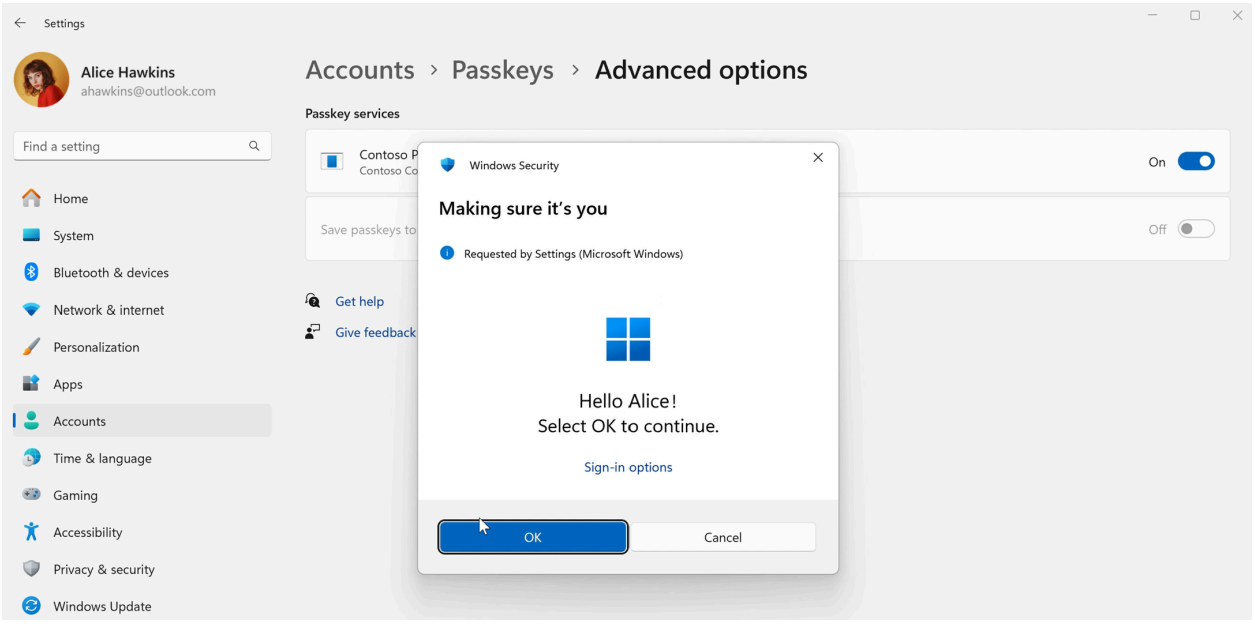
> ⚠ **Warning**
>
> Contoso Passkey Manager is designed for passkey creation and usage testing only. Don't use the app for production passkeys.

Step 1: Installation

1. Set up your development environment for WinUI 3 development (see Install tools for the Windows App SDK).
2. Clone the Windows-classic-samples GitHub repo for the Contoso Passkey Manager sample app.
3. In Visual Studio, open the Contoso Passkey Manager solution that you just cloned, carefully follow the instructions at the link above, build the sample, and then run it to confirm installation.

Step 2: Setup

1. Go to **Settings** > **Accounts** > **Passkeys**, and then **Advanced options**.
2. Switch the toggle to **On** for Contoso Passkey Manager.
3. Complete the Windows Hello user verification, and now you can save passkeys to the Contoso Passkey Manager.

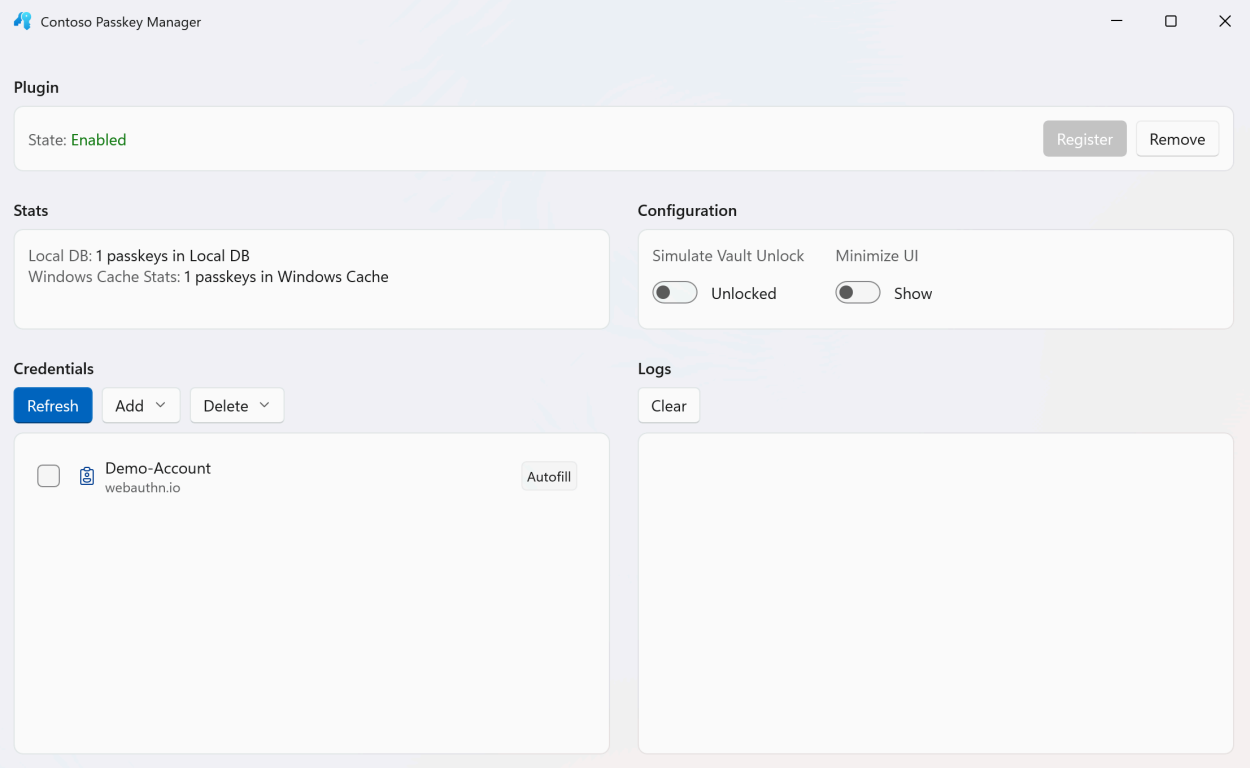


Step 3: Passkey creation

1. Navigate to a website such as webauthn.io.
2. Create a test user name, and click **Register**.
3. You will be prompted, on saving your passkey, with the Contoso Passkey Manager. Click **Continue**.

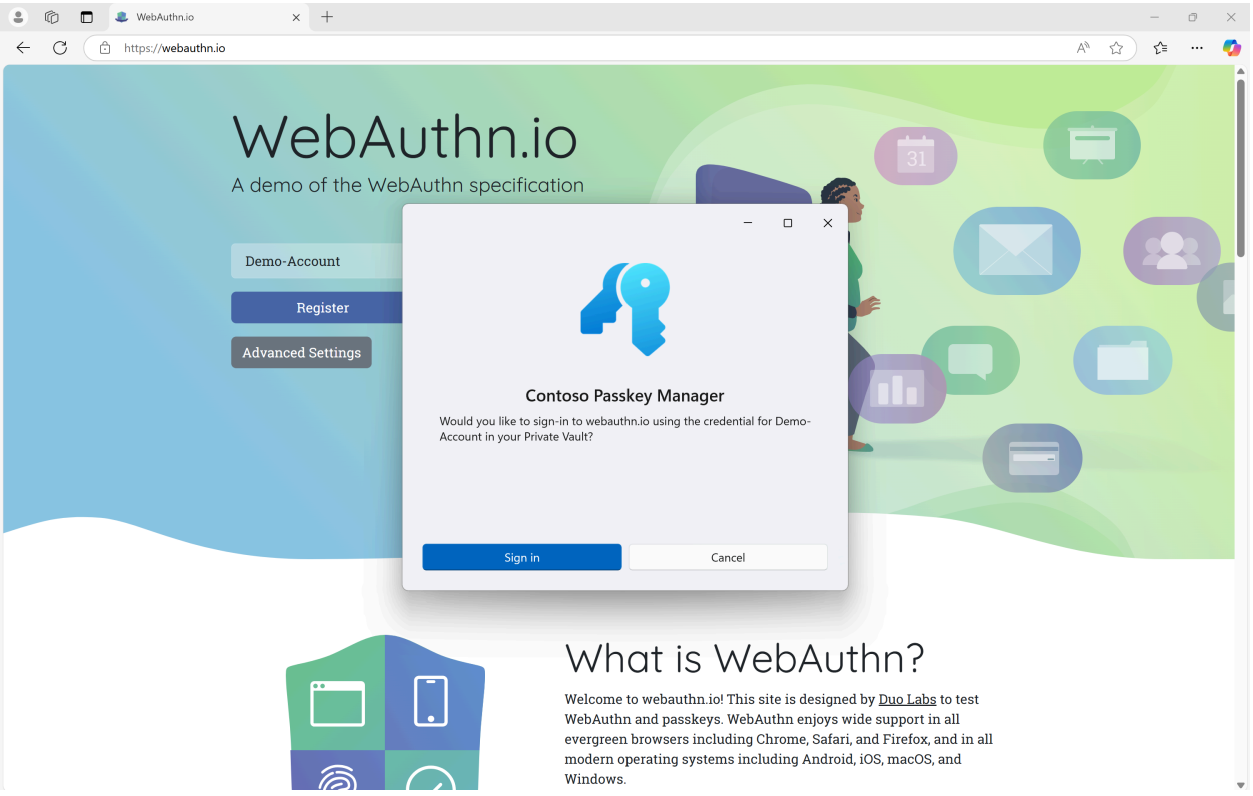
4. The Contoso Passkey Manager app will open. Click **Create** to create the credential in Contoso Passkey Manager.
5. Complete user verification with Windows Hello, and the passkey is created.
6. You can see your saved passkeys by opening the Contoso Passkey Manager.
7. If you want to test different experiences, then you can toggle **Simulate Vault Unlock** to test a vault unlock user experience when saving a passkey. The **Minimize**

UI toggle simplifies the step that opens the passkey manager, and the experience flow will instead just be with user verification on Windows Hello.



Step 4: Passkey authentication

1. Input your username, and click **Authenticate**.
2. Select to sign in with the passkey saved to the Contoso Passkey Manager.



3. Complete user verification with Windows Hello to authenticate.

4. You've successfully logged in.

Next steps

To integrate with the third-party plugin capability, go to webauthn to find the header files.

See also

- WebAuthNGetPlatformCredentialList
- webauthn.h header

Implement OAuth 2.0 functionality in Windows apps

Article • 03/19/2025

The OAuth2Manager in Windows App SDK enables desktop applications such as WinUI 3 to seamlessly perform OAuth 2.0 authorization on Windows. **OAuth2Manager** API intentionally doesn't provide APIs for the implicit request and resource owner password credential because of the security concerns that entails. It's recommended to use the authorization code grant type using Proof Key for Code Exchange (PKCE). For more information, see the PKCE RFC.

OAuth background

The Windows Runtime (WinRT) WebAuthenticationBroker, primarily designed for UWP apps, presents several challenges when used in desktop apps. Key issues include the dependency on ApplicationView, which isn't compatible with desktop app frameworks. As a result, developers are forced to resort to workarounds involving interop interfaces and additional code to implement OAuth 2.0 functionality into WinUI 3 and other desktop apps.

OAuth2Manager API in Windows App SDK

The **OAuth2Manager** API for Windows App SDK aims to provide a streamlined solution that meets the expectations of developers. It offers seamless OAuth 2.0 capabilities with full feature parity across all Windows platforms supported by Windows App SDK. The new API eliminates the need for cumbersome workarounds and simplifies the process of incorporating OAuth 2.0 functionality into desktop apps.

The **OAuth2Manager** is different than the **WebAuthenticationBroker** in WinRT. It follows OAuth 2.0 best practices more closely - e.g. using the user's default browser. The best practices for the API are taken from the IETF (Internet Engineering Task Force) OAuth 2.0 Authorization Framework RFC 6749, PKCE RFC 7636, and OAuth 2.0 for Native Apps RFC 8252.

Perform OAuth 2.0 examples

A full WinUI 3 sample app is available on GitHub. The following sections provide code snippets for the most common OAuth 2.0 flows using the **OAuth2Manager** API.

Authorization code request

The following example demonstrates how to perform an authorization code request using the **OAuth2Manager** in Windows App SDK:

C++

```
// Get the WindowId for the application window
Microsoft::UI::WindowId parentWindowId = this->AppWindow().Id();

AuthRequestParams authRequestParams =
AuthRequestParams::CreateForAuthorizationCodeRequest(L"my_client_id",
   Uri(L"my-app:/oauth-callback/"));
authRequestParams.Scope(L"user:email user:birthday");

AuthRequestResult authRequestResult = co_await
OAuth2Manager::RequestAuthWithParamsAsync(parentWindowId,
   Uri(L"https://my.server.com/oauth/authorize"), authRequestParams);
if (AuthResponse authResponse = authRequestResult.Response())
{
   //To obtain the authorization code
   //authResponse.Code();

   //To obtain the access token
   DoTokenExchange(authResponse);
}
else
{
   AuthFailure authFailure = authRequestResult.Failure();
   NotifyFailure(authFailure.Error(), authFailure.ErrorDescription());
}
```

Exchange authorization code for access token

The following example demonstrates how to exchange an authorization code for an access token using the **OAuth2Manager**:

C++

```
AuthResponse authResponse = authRequestResult.Response();
TokenRequestParams tokenRequestParams =
TokenRequestParams::CreateForAuthorizationCodeRequest(authResponse);
ClientAuthentication clientAuth =
ClientAuthentication::CreateForBasicAuthorization(L"my_client_id",
    L"my_client_secret");
```

```
TokenRequestResult tokenRequestResult = co_await
OAuth2Manager::RequestTokenAsync(
    Uri(L"https://my.server.com/oauth/token"), tokenRequestParams,
clientAuth);
if (TokenResponse tokenResponse = tokenRequestResult.Response())
{
    String accessToken = tokenResponse.AccessToken();
    String tokenType = tokenResponse.TokenType();

    // RefreshToken string null/empty when not present
    if (String refreshToken = tokenResponse.RefreshToken();
!refreshToken.empty())
    {
        // ExpiresIn is zero when not present
        DateTime expires = winrt::clock::now();
        if (String expiresIn = tokenResponse.ExpiresIn();
std::stoi(expiresIn) != 0)
        {
            expires += std::chrono::seconds(static_cast<int64_t>
(std::stoi(expiresIn)));
        }
        else
        {
            // Assume a duration of one hour
            expires += std::chrono::hours(1);
        }

        //Schedule a refresh of the access token
        myAppState.ScheduleRefreshAt(expires, refreshToken);
    }

    // Use the access token for resources
    DoRequestWithToken(accessToken, tokenType);
}
else
{
    TokenFailure tokenFailure = tokenRequestResult.Failure();
    NotifyFailure(tokenFailure.Error(),
tokenFailure.ErrorDescription());
}
```

Refresh an access token

The following example shows how to refresh an access token using the **OAuth2Manager**'s RefreshTokenAsync method:

C++

```
TokenRequestParams tokenRequestParams =
TokenRequestParams::CreateForRefreshToken(refreshToken);
ClientAuthentication clientAuth =
ClientAuthentication::CreateForBasicAuthorization(L"my_client_id",
    L"my_client_secret");
TokenRequestResult tokenRequestResult = co_await
OAuth2Manager::RequestTokenAsync(
    Uri(L"https://my.server.com/oauth/token"), tokenRequestParams,
clientAuth));
if (TokenResponse tokenResponse = tokenRequestResult.Response())
{
    UpdateToken(tokenResponse.AccessToken(), tokenResponse.TokenType(),
tokenResponse.ExpiresIn());

    //Store new refresh token if present
    if (String refreshToken = tokenResponse.RefreshToken();
!refreshToken.empty())
    {
        // ExpiresIn is zero when not present
        DateTime expires = winrt::clock::now();
        if (String expiresInStr = tokenResponse.ExpiresIn();
!expiresInStr.empty())
        {
            int expiresIn = std::stoi(expiresInStr);
            if (expiresIn != 0)
            {
                expires += std::chrono::seconds(static_cast<int64_t>
(expiresIn));
            }
        }
        else
        {
            // Assume a duration of one hour
            expires += std::chrono::hours(1);
        }

        //Schedule a refresh of the access token
        myAppState.ScheduleRefreshAt(expires, refreshToken);
    }
}
else
{
    TokenFailure tokenFailure = tokenRequestResult.Failure();
    NotifyFailure(tokenFailure.Error(),
tokenFailure.ErrorDescription());
}
```

Complete an authorization request

Finally, to complete an authorization request from a protocol activation, your app should handle the AppInstance.Activated event. This is required when having custom redirect logic. A full example is available on GitHub.

Use the following code:

C++

```
void App::OnActivated(const IActivatedEventArgs& args)
{
    if (args.Kind() == ActivationKind::Protocol)
    {
        auto protocolArgs = args.as<ProtocolActivatedEventArgs>();
        if (OAuth2Manager::CompleteAuthRequest(protocolArgs.Uri()))
        {
            TerminateCurrentProcess();
        }

        DisplayUnhandledMessageToUser();
    }
}
```

Related content

- WebAuthenticationBroker
- OAuth2Manager
- PKCE RFC 7636

Share certificates between Windows apps

Article • 11/22/2024

Windows apps that require secure authentication beyond a user Id and password combination can use certificates for authentication. Certificate authentication provides a high level of trust when authenticating a user. In some cases, a group of services will want to authenticate a user for multiple apps. This article shows how you can authenticate multiple Windows apps using the same certificate, and how you can provide a method for users to import a certificate that was provided for access to secured web services.

Apps can authenticate to a web service using a certificate, and multiple apps can use a single certificate from the certificate store to authenticate the same user. If a certificate does not exist in the store, you can add code to your app to import a certificate from a PFX file. The client app in this quickstart is a WinUI app, and the web service is an ASP.NET Core web API.

> **Tip**
>
> Microsoft Copilot is a great resource if you have questions about getting started writing Windows apps or ASP.NET Core web APIs. Copilot can help you write code, find examples, and learn more about best practices for creating secure apps.

Prerequisites

- Visual Studio with the **ASP.NET and web development** and **Windows application development** workloads installed.
- The latest Windows Software Development Kit (SDK) to use the Windows Runtime (WinRT) APIs in your WinUI app.
- PowerShell for working with self-signed certificates.

Create and publish a secured web service

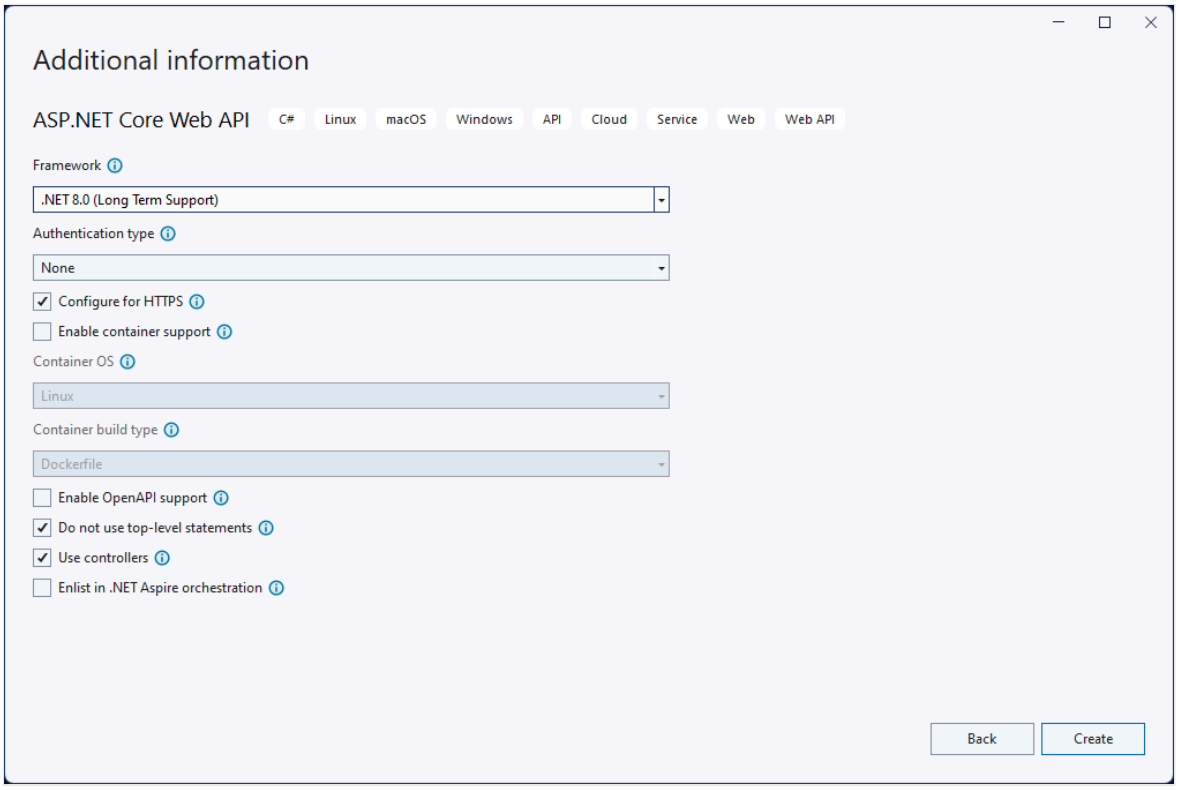
1. Open Microsoft Visual Studio and select **Create a new project** from the start screen.

2. In the Create a new project dialog, select **API** in the **Select a project type** dropdown list to filter the available project templates.

3. Select the **ASP.NET Core Web API** template and select **Next**.

4. Name the application "FirstContosoBank" and select **Next**.

5. Choose **.NET 8.0** or later as the **Framework**, set the **Authentication type** to **None**, ensure **Configure for HTTPS** is checked, uncheck **Enable OpenAPI support**, check **Do not use top-level statements** and **Use controllers**, and select **Create**.



6. Right-click the **WeatherForecastController.cs** file in the **Controllers** folder and select **Rename**. Change the name to **BankController.cs** and let Visual Studio rename the class and all references to the class.

7. In the **launchSettings.json** file, change the value of "launchUrl" from "weatherforecast" to "bank" for all three configuration what use the value.

8. In the **BankController.cs** file, add following "Login" method.

cs

```cs
[HttpGet]
[Route("login")]
public string Login()
{
    // Return any value you like here.
    // The client is just looking for a 200 OK response.
```

```
    return "true";
}
```

9. Open the NuGet Package Manager and search for and install latest stable version of the **Microsoft.AspNetCore.Authentication.Certificate** package. This package provides middleware for certificate authentication in ASP.NET Core.

10. Add a new class to the project named **SecureCertificateValidationService**. Add the following code to the class to configure the certificate authentication middleware.

cs

```
using System.Security.Cryptography.X509Certificates;

public class SecureCertificateValidationService
{
    public bool ValidateCertificate(X509Certificate2 clientCertificate)
    {
        // Values are hard-coded for this example.
        // You should load your valid thumbprints from a secure
location.
        string[] allowedThumbprints = {
"YOUR_CERTIFICATE_THUMBPRINT_1", "YOUR_CERTIFICATE_THUMBPRINT_2" };
        if (allowedThumbprints.Contains(clientCertificate.Thumbprint))
        {
            return true;
        }
    }
}
```

11. Open **Program.cs** and replace the code in the **Main** method with the following code:

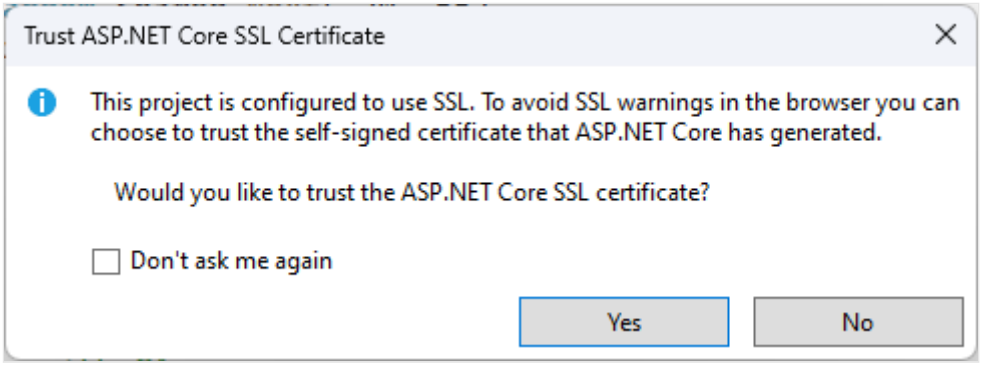
C#

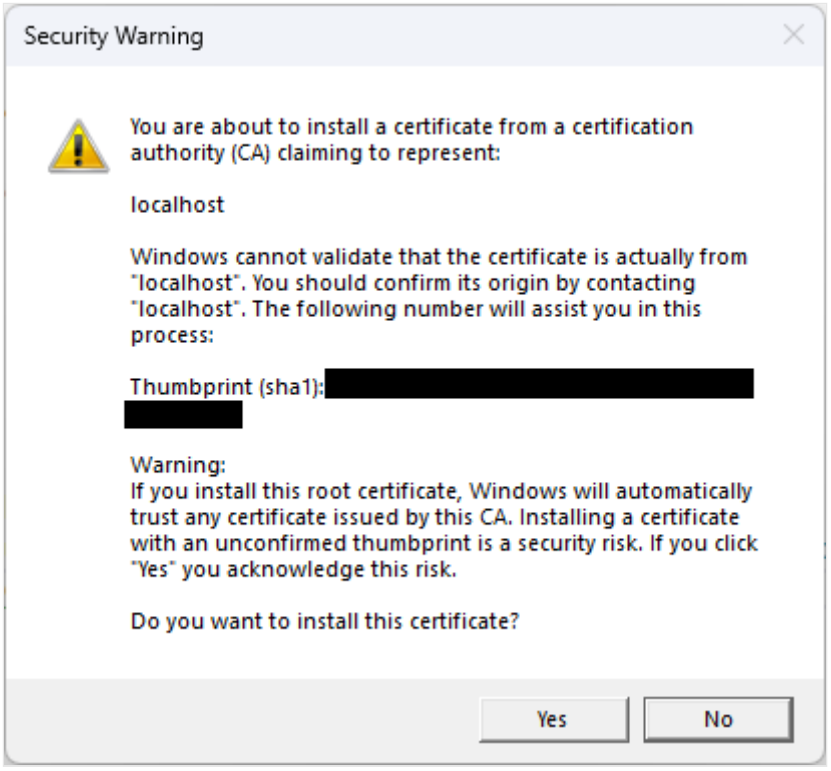
```
public static void Main(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);

    // Add our certificate validation service to the DI container.
    builder.Services.AddTransient<SecureCertificateValidationService>
();

    builder.Services.Configure<KestrelServerOptions>(options =>
    {
        // Configure Kestrel to require a client certificate.
        options.ConfigureHttpsDefaults(options =>
        {
            options.ClientCertificateMode =
ClientCertificateMode.RequireCertificate;
```

```
            options.AllowAnyClientCertificate();
        });
    });

    builder.Services.AddControllers();

    // Add certificate authentication middleware.
    builder.Services.AddAuthentication(
    CertificateAuthenticationDefaults.AuthenticationScheme)
       .AddCertificate(options =>
    {
        options.AllowedCertificateTypes = CertificateTypes.SelfSigned;
        options.Events = new CertificateAuthenticationEvents
        {
            // Validate the certificate with the validation service.
            OnCertificateValidated = context =>
            {
                var validationService =
context.HttpContext.RequestServices.GetService<SecureCertificateValidat
ionService>();

                if
(validationService.ValidateCertificate(context.ClientCertificate))
                {
                    context.Success();
                }
                else
                {
                    context.Fail("Invalid certificate");
                }

                return Task.CompletedTask;
            },
            OnAuthenticationFailed = context =>
            {
                context.Fail("Invalid certificate");
                return Task.CompletedTask;
            }
        };
     });

     var app = builder.Build();

     // Add authentication/authorization middleware.
     app.UseHttpsRedirection();
     app.UseAuthentication();
     app.UseAuthorization();

     app.MapControllers();
     app.Run();
 }
```

The code above configures the Kestrel server to require a client certificate and adds the certificate authentication middleware to the app. The middleware validates the client certificate using the `SecureCertificateValidationService` class. The `OnCertificateValidated` event is called when a certificate is validated. If the certificate is valid, the event calls the `Success` method. If the certificate is invalid, the event calls the `Fail` method with an error message, which will be returned to the client.

12. Start debugging the project to launch the web service. You may receive messages about trusting and installing an SSL certificate. Click **Yes** for each of these messages to trust the certificate and continue debugging the project.





13. The web service will be available at `https://localhost:7072/bank`. You can test the web service by opening a web browser and entering the web address. You will see the generated weather forecast data formatted as JSON. Keep the web service running while you create the client app.

For more information on working with ASP.NET Core controller-based web APIs, see Create a web API with ASP.NET Core.

Create a WinUI app that uses certificate authentication

Now that you have one or more secured web services, your apps can use certificates to authenticate to those web services. When you make a request to an authenticated web service using the HttpClient object from the WinRT APIs, the initial request will not contain a client certificate. The authenticated web service will respond with a request for client authentication. When this occurs, the Windows client will automatically query the certificate store for available client certificates. Your user can select from these certificates to authenticate to the web service. Some certificates are password protected, so you will need to provide the user with a way to input the password for a certificate.

> **Note**
>
> There are no Windows App SDK APIs for managing certificates yet. You must use the WinRT APIs to manage certificates in your app. We will also be using WinRT storage APIs to import a certificate from a PFX file. Many WinRT APIs can be used by any Windows app with package identity, including WinUI apps.
>
> The HTTP client code we'll implement uses .NET's HttpClient. The HttpClient included in the WinRT APIs doesn't support client certificates.

If there are no client certificates available, then the user will need to add a certificate to the certificate store. You can include code in your app that enables a user to select a PFX file that contains a client certificate and then import that certificate into the client certificate store.

> **Tip**
>
> You can use the PowerShell cmdlets **New-SelfSignedCertificate** and **Export-PfxCertificate** to create a self-signed certificate and export it to a PFX file to use with this quickstart. For information, see New-SelfSignedCertificate and Export-PfxCertificate.
>
> Note that when generating the certificate, you should save the thumbprint of the certificate to use in the web service for validation.

1. Open Visual Studio and create a new WinUI project from the start page. Name the new project "FirstContosoBankApp". Click **Create** to create the new project.

2. In the **MainWindow.xaml** file, add the following XAML to a **Grid** element, replacing the existing **StackPanel** element and its contents. This XAML includes a button to browse for a PFX file to import, a text box to enter a password for a password-protected PFX file, a button to import a selected PFX file, a button to log in to the secured web service, and a text block to display the status of the current action.

```xml
<Button x:Name="Import" Content="Import Certificate (PFX file)"
HorizontalAlignment="Left" Margin="352,305,0,0" VerticalAlignment="Top"
Height="77" Width="260" Click="Import_Click" FontSize="16"/>
<Button x:Name="Login" Content="Login" HorizontalAlignment="Left"
Margin="611,305,0,0" VerticalAlignment="Top" Height="75" Width="240"
Click="Login_Click" FontSize="16"/>
<TextBlock x:Name="Result" HorizontalAlignment="Left"
Margin="355,398,0,0" TextWrapping="Wrap" VerticalAlignment="Top"
Height="153" Width="560"/>
<PasswordBox x:Name="PfxPassword" HorizontalAlignment="Left"
Margin="483,271,0,0" VerticalAlignment="Top" Width="229"/>
<TextBlock HorizontalAlignment="Left" Margin="355,271,0,0"
TextWrapping="Wrap" Text="PFX password" VerticalAlignment="Top"
FontSize="18" Height="32" Width="123"/>
<Button x:Name="Browse" Content="Browse for PFX file"
HorizontalAlignment="Left" Margin="352,189,0,0" VerticalAlignment="Top"
Click="Browse_Click" Width="499" Height="68" FontSize="16"/>
<TextBlock HorizontalAlignment="Left" Margin="717,271,0,0"
TextWrapping="Wrap" Text="(Optional)" VerticalAlignment="Top"
Height="32" Width="83" FontSize="16"/>
```

3. Save the **MainWindow** changes.

4. Open the **MainWindow.xaml.cs** file, and add the following `using` statements.

```cs
using System;
using System.Security.Cryptography.X509Certificates;
using System.Diagnostics;
using System.Net.Http;
using System.Net;
using System.Text;
using Microsoft.UI.Xaml;
using Windows.Security.Cryptography.Certificates;
using Windows.Storage.Pickers;
using Windows.Storage;
using Windows.Storage.Streams;
```

5. In the MainWindow.xaml.cs file, add the following variables to the **MainWindow** class. They specify the address for the secured **login** service endpoint of your "FirstContosoBank" web service, and a global variable that holds a PFX certificate to import into the certificate store. Update the `<server-name>` to `localhost:7072` or whichever port is specified in the "https" configuration in your API project's launchSettings.json file.

```cs
private Uri requestUri = new Uri("https://<server-name>/bank/login");
private string pfxCert = null;
```

6. In the **MainWindow.xaml.cs** file, add the following click handler for the login button and method to access the secured web service.

```cs
private void Login_Click(object sender, RoutedEventArgs e)
{
    MakeHttpsCall();
}

private async void MakeHttpsCall()
{
    var result = new StringBuilder("Login ");

    // Load the certificate
    var certificate = new
X509Certificate2(Convert.FromBase64String(pfxCert),
                                            PfxPassword.Password);

    // Create HttpClientHandler and add the certificate
    var handler = new HttpClientHandler();
    handler.ClientCertificates.Add(certificate);
    handler.ClientCertificateOptions =
ClientCertificateOption.Automatic;

    // Create HttpClient with the handler
    var client = new HttpClient(handler);

    try
    {
        // Make a request
        var response = await client.GetAsync(requestUri);

        if (response.StatusCode == HttpStatusCode.OK)
        {
            result.Append("successful");
        }
        else
```

```
        {
            result = result.Append("failed with ");
            result = result.Append(response.StatusCode);
        }
    }
    catch (Exception ex)
    {
        result = result.Append("failed with ");
        result = result.Append(ex.Message);
    }

    Result.Text = result.ToString();
}
```

7. Next, add the following click handlers for the button to browse for a PFX file and the button to import a selected PFX file into the certificate store.

cs

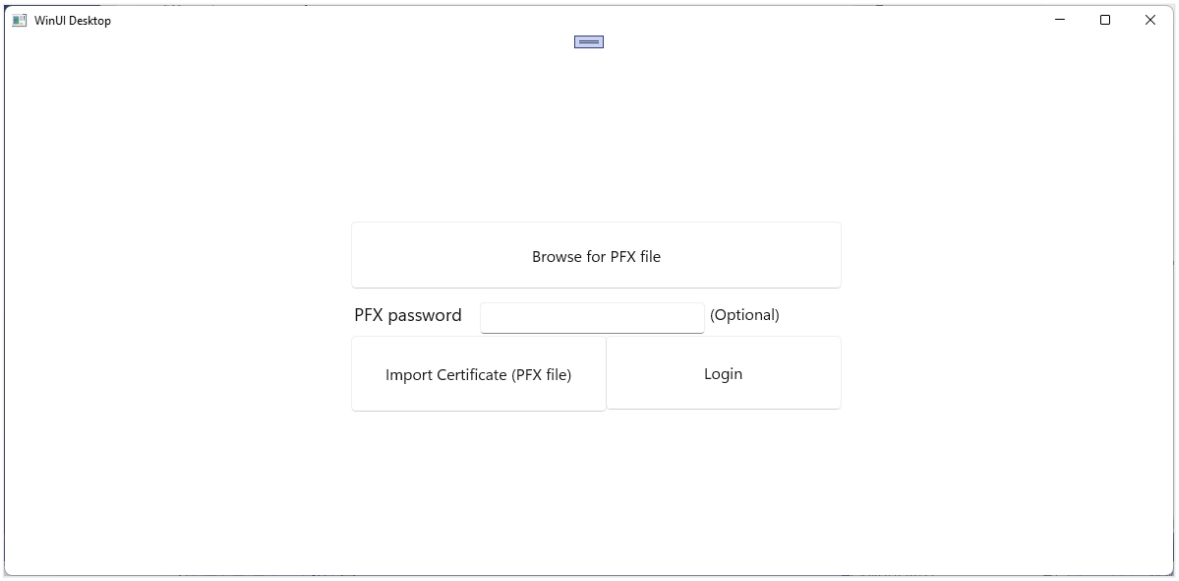
```
private async void Import_Click(object sender, RoutedEventArgs e)
{
    try
    {
        Result.Text = "Importing selected certificate into user
certificate store....";
        await
CertificateEnrollmentManager.UserCertificateEnrollmentManager.ImportPfx
DataAsync(
              pfxCert,
              PfxPassword.Password,
              ExportOption.Exportable,
              KeyProtectionLevel.NoConsent,
              InstallOptions.DeleteExpired,
              "Import Pfx");

        Result.Text = "Certificate import succeeded";
    }
    catch (Exception ex)
    {
        Result.Text = "Certificate import failed with " + ex.Message;
    }
}

private async void Browse_Click(object sender, RoutedEventArgs e)
{
    var result = new StringBuilder("Pfx file selection ");
    var pfxFilePicker = new FileOpenPicker();
    IntPtr hwnd = WinRT.Interop.WindowNative.GetWindowHandle(this);
    WinRT.Interop.InitializeWithWindow.Initialize(pfxFilePicker, hwnd);
    pfxFilePicker.FileTypeFilter.Add(".pfx");
    pfxFilePicker.CommitButtonText = "Open";
    try
    {
```

```
        StorageFile pfxFile = await
pfxFilePicker.PickSingleFileAsync();
        if (pfxFile != null)
        {
            IBuffer buffer = await FileIO.ReadBufferAsync(pfxFile);
            using (DataReader dataReader =
DataReader.FromBuffer(buffer))
            {
                byte[] bytes = new byte[buffer.Length];
                dataReader.ReadBytes(bytes);
                pfxCert = System.Convert.ToBase64String(bytes);
                PfxPassword.Password = string.Empty;
                result.Append("succeeded");
            }
        }
        else
        {
            result.Append("failed");
        }
    }
    catch (Exception ex)
    {
        result.Append("failed with ");
        result.Append(ex.Message); ;
    }

    Result.Text = result.ToString();
}
```

8. Open the **Package.appxmanifest** file and add the following capabilities to the **Capabilities** tab.

    - **EnterpriseAuthentication**
    - **SharedUserCertificates**

9. Run your app and log in to your secured web service as well as import a PFX file into the local certificate store.



You can use these steps to create multiple apps that use the same user certificate to access the same or different secured web services.

Related content

Windows Hello

Security and identity

Create a web API with ASP.NET Core

Smart cards

Article • 08/09/2024

This topic explains how Windows apps can use smart cards to connect users to secure network services, including how to access physical smart card readers, create virtual smart cards, communicate with smart cards, authenticate users, reset user PINs, and remove or disconnect smart cards.

The Windows Runtime (WinRT) APIs for smart cards are part of the Windows Software Development Kit (SDK). These APIs were created for use in Universal Windows Platform (UWP) apps, but they can also be used in WinUI apps or in packaged desktop apps, including WPF and Windows Forms. For more information about using WinRT APIs in your Windows desktop app, see Call Windows Runtime APIs in desktop apps.

Configure the app manifest

Before your app can authenticate users using smart cards or virtual smart cards, you must set the **Shared User Certificates** capability in the project Package.appxmanifest file of your WinUI project or packaging project.

Access connected card readers and smart cards

You can query for readers and attached smart cards by passing the device ID (specified in DeviceInformation) to the SmartCardReader.FromIdAsync method. To access the smart cards currently attached to the returned reader device, call SmartCardReader.FindAllCardsAsync.

cs

```
string selector = SmartCardReader.GetDeviceSelector();
DeviceInformationCollection devices =
    await DeviceInformation.FindAllAsync(selector);

foreach (DeviceInformation device in devices)
{
    SmartCardReader reader =
        await SmartCardReader.FromIdAsync(device.Id);

    // For each reader, we want to find all the cards associated
    // with it. Then we will create a SmartCardListItem for
    // each (reader, card) pair.
    IReadOnlyList<SmartCard> cards =
```

```
        await reader.FindAllCardsAsync();
}
```

You should also enable your app to observe for CardAdded events by implementing a method to handle app behavior on card insertion.

cs

```
private void reader_CardAdded(SmartCardReader sender, CardAddedEventArgs
args)
{
  // A card has been inserted into the sender SmartCardReader.
}
```

You can then pass each returned SmartCard object to SmartCardProvisioning to access the methods that allow your app to access and customize its configuration.

Create a virtual smart card

To create a virtual smart card using SmartCardProvisioning, your app will first need to provide a friendly name, an admin key, and a SmartCardPinPolicy. The friendly name is generally something provided to the app, but your app will still need to provide an admin key and generate an instance of the current **SmartCardPinPolicy** before passing all three values to RequestVirtualSmartCardCreationAsync.

1. Create a new instance of a SmartCardPinPolicy
2. Generate the admin key value by calling CryptographicBuffer.GenerateRandom on the admin key value provided by the service or management tool.
3. Pass these values along with the *FriendlyNameText* string to RequestVirtualSmartCardCreationAsync.

cs

```
var pinPolicy = new SmartCardPinPolicy
    {
        MinLength = 6
    };

IBuffer adminkey = CryptographicBuffer.GenerateRandom(24);

SmartCardProvisioning provisioning = await
     SmartCardProvisioning.RequestVirtualSmartCardCreationAsync(
          "Card friendly name",
          adminkey,
          pinPolicy);
```

Once RequestVirtualSmartCardCreationAsync has returned the associated SmartCardProvisioning object, the virtual smart card is provisioned and ready for use.

> **Note**
>
> In order to create a virtual smart card using a packaged Windows app, the user running the app must be a member of the administrators group. If the user is not a member of the administrators group, virtual smart card creation will fail.

Handle authentication challenges

To authenticate with smart cards or virtual smart cards, your app must provide the behavior to complete challenges between the admin key data stored on the card, and the admin key data maintained by the authentication server or management tool.

The following code shows how to support smart card authentication for services or modification of physical or virtual card details. If the data generated using the admin key on the card ("challenge") is the same as the admin key data provided by the server or management tool ("adminkey"), authentication is successful.

cs

```
static class ChallengeResponseAlgorithm
{
    public static IBuffer CalculateResponse(IBuffer challenge, IBuffer
adminkey)
    {
        if (challenge == null)
            throw new ArgumentNullException("challenge");
        if (adminkey == null)
            throw new ArgumentNullException("adminkey");

        SymmetricKeyAlgorithmProvider objAlg =
SymmetricKeyAlgorithmProvider.OpenAlgorithm(SymmetricAlgorithmNames.TripleDe
sCbc);
        var symmetricKey = objAlg.CreateSymmetricKey(adminkey);
        var buffEncrypted = CryptographicEngine.Encrypt(symmetricKey,
challenge, null);
        return buffEncrypted;
    }
}
```

You will see this code referenced throughout the remainder of this topic was we review how to complete an authentication action, and how to apply changes to smart card and virtual smart card information.

Verify smart card or virtual smart card authentication response

Now that we have the logic for authentication challenges defined, we can communicate with the reader to access the smart card, or alternatively, access a virtual smart card for authentication.

1. To begin the challenge, call GetChallengeContextAsync from the SmartCardProvisioning object associated with the smart card. This will generate an instance of SmartCardChallengeContext, which contains the card's Challenge value.
2. Next, pass the card's challenge value and the admin key provided by the service or management tool to the **ChallengeResponseAlgorithm** that we defined in the previous example.
3. VerifyResponseAsync will return **true** if authentication is successful.

cs

```
bool verifyResult = false;
SmartCard card = await rootPage.GetSmartCard();
SmartCardProvisioning provisioning =
    await SmartCardProvisioning.FromSmartCardAsync(card);

SmartCardChallengeContext context =
    await provisioning.GetChallengeContextAsync();

IBuffer response = ChallengeResponseAlgorithm.CalculateResponse(
    context.Challenge,
    rootPage.AdminKey);

verifyResult = await context.VerifyResponseAsync(response);
```

Change or reset a user PIN

To change the PIN associated with a smart card:

1. Access the card and generate the associated SmartCardProvisioning object.
2. Call RequestPinChangeAsync to display a UI to the user to complete this operation.
3. If the PIN was successfully changed the call will return **true**.

cs

```
SmartCardProvisioning provisioning =
    await SmartCardProvisioning.FromSmartCardAsync(card);
```

```
bool result = await provisioning.RequestPinChangeAsync();
```

To request a PIN reset:

1. Call RequestPinResetAsync to initiate the operation. This call includes a SmartCardPinResetHandler method that represents the smart card and the pin reset request.
2. SmartCardPinResetHandler provides information that our **ChallengeResponseAlgorithm**, wrapped in a SmartCardPinResetDeferral call, uses to compare the card's challenge value and the admin key provided by the service or management tool to authenticate the request.
3. If the challenge is successful, the RequestPinResetAsync call is completed; returning **true** if the PIN was successfully reset.

cs

```
SmartCardProvisioning provisioning =
    await SmartCardProvisioning.FromSmartCardAsync(card);

bool result = await provisioning.RequestPinResetAsync(
    (pinResetSender, request) =>
    {
        SmartCardPinResetDeferral deferral =
            request.GetDeferral();

        try
        {
            IBuffer response =
                ChallengeResponseAlgorithm.CalculateResponse(
                    request.Challenge,
                    rootPage.AdminKey);
            request.SetResponse(response);
        }
        finally
        {
            deferral.Complete();
        }
    });
}
```

Remove a smart card or virtual smart card

When a physical smart card is removed a CardRemoved event will fire when the card is deleted.

Associate the firing of this event with the card reader with the method that defines your app's behavior on card or reader removal as an event handler. This behavior can be something as simply as providing notification to the user that the card was removed.

```cs
reader = card.Reader;
reader.CardRemoved += HandleCardRemoved;
```

The removal of a virtual smart card is handled programmatically by first retrieving the card and then calling RequestVirtualSmartCardDeletionAsync from the SmartCardProvisioning returned object.

```cs
bool result = await SmartCardProvisioning
    .RequestVirtualSmartCardDeletionAsync(card);
```

Related content

- Call Windows Runtime APIs in desktop apps
- SmartCard

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Create a Windows Hello login app

Article • 07/11/2024

This is the first part of a complete walkthrough on how to create a packaged Windows app that uses Windows Hello as an alternative to traditional username and password authentication systems. In this case, the app is a WinUI app, but the same approach can be used with any packaged Windows app, including WPF and Windows Forms apps. The app uses a username for sign-in and creates a Hello key for each account. These accounts will be protected by the PIN that is set up in Windows Settings on configuration of Windows Hello.

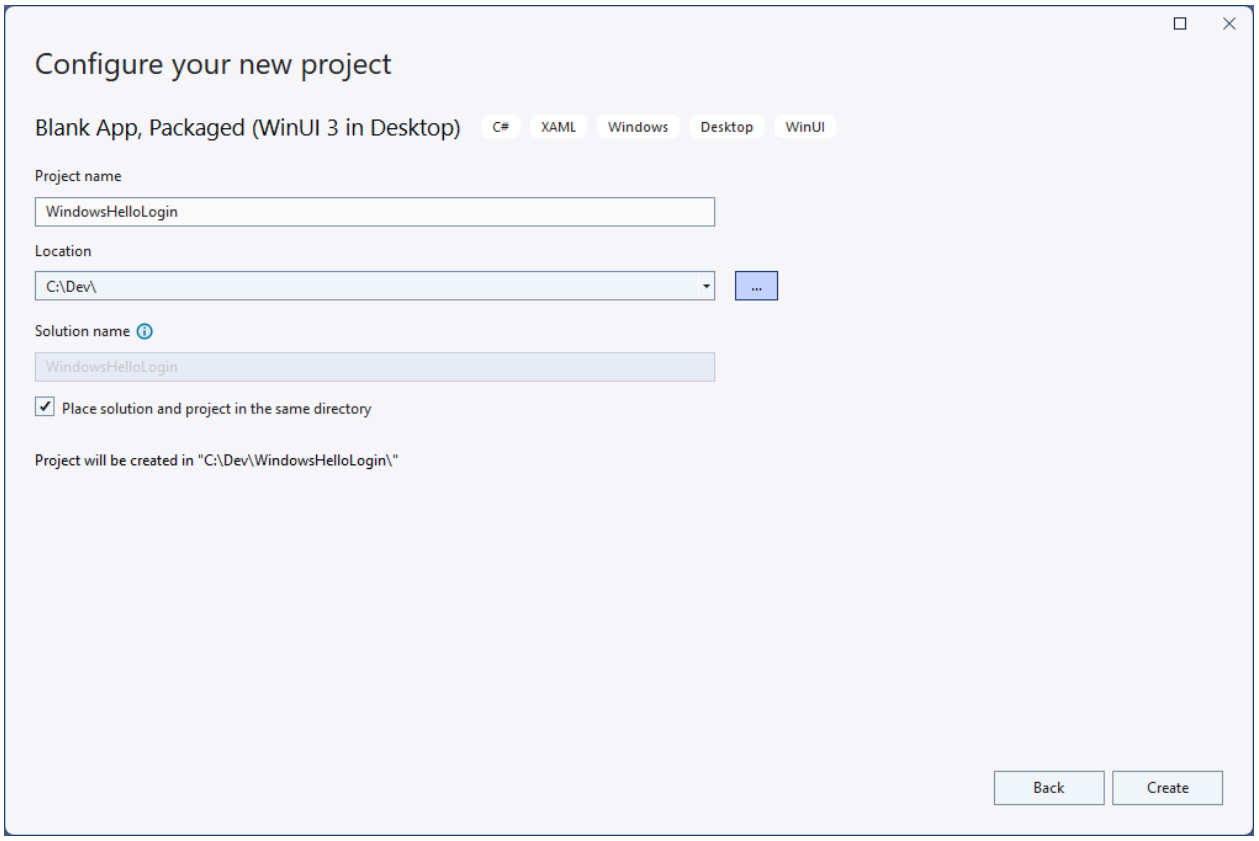
This walkthrough is split into two parts: building the app and connecting the backend service. When you're finished with this article, continue on to Part 2: Windows Hello login service.

Before you begin, you should read the Windows Hello overview for a general understanding of how Windows Hello works.

Get started

In order to build this project, you'll need some experience with C#, and XAML. You'll also need to be using Visual Studio 2022 on a Windows 10 or Windows 11 machine. See Start developing Windows apps for complete instructions on setting up your development environment.

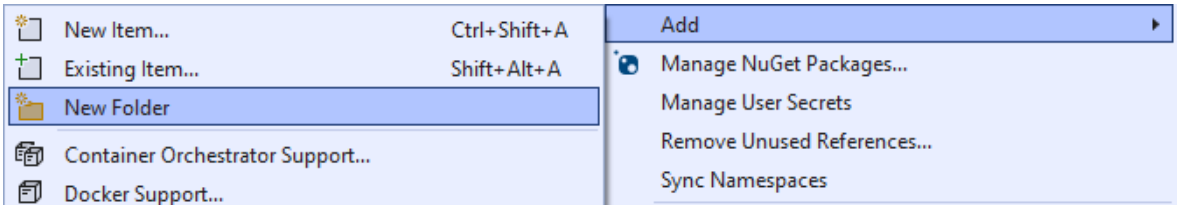
- In Visual Studio, select **File** > **New** > **Project**.
- In the **New Project** dialog's drop-down filters, select **C#/C++**, **Windows**, and **WinUI**, respectively.
- Choose **Blank App, Packaged (WinUI 3 in Desktop)** and name your application "WindowsHelloLogin".
- Build and Run the new application (F5), you should see a blank window shown on the screen. Close the application.



Exercise 1: Login with Windows Hello

In this exercise you will learn how to check if Windows Hello is setup on the machine, and how to sign into an account using Windows Hello.

- In the new project create a new folder in the solution called "Views". This folder will contain the pages that will be navigated to in this sample. Right-click the project in **Solution Explorer**, select **Add** > **New Folder**, then rename the folder to **Views**.

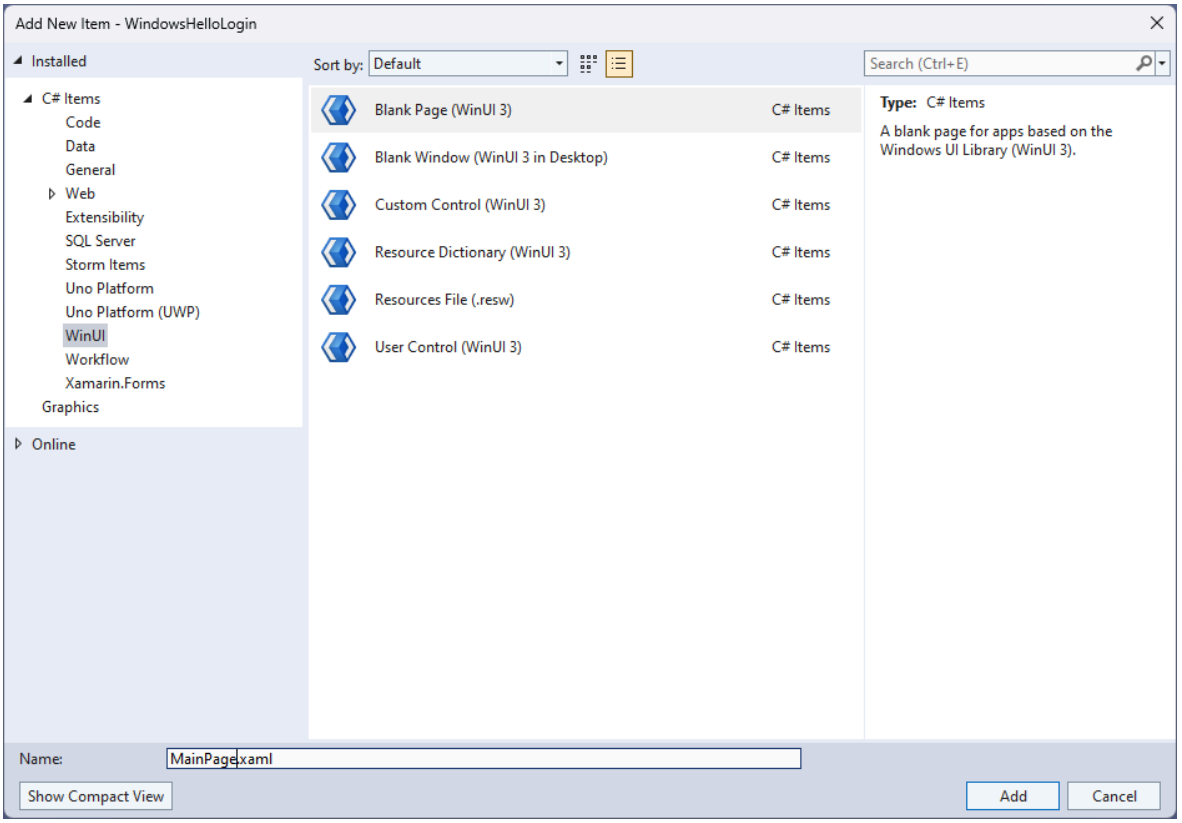


- Open MainWindow.xaml and replace the `Window` contents with an empty `StackPanel` or `Grid` control. We will be implementing page navigation and navigating to a new page when the **MainWindow** is loaded, so we don't need any content in the **MainWindow**.

- Add a `Title` property to **MainWindow** in the XAML. The attribute should look like this: `Title="Windows Hello Login"`.

- Remove the myButton_Click event handler from MainWindow.xaml.cs to avoid any compilation errors. This event handler is not needed for this sample.

- Right-click the new **Views** folder, select **Add** > **New Item** and select the **Blank Page** template. Name this page "MainPage.xaml".



- Open the App.xaml.cs file and update the **OnLaunched** handler to implement page navigation for the app. You'll also need to add a **RootFrame_NavigationFailed** handler method to deal with any errors that occur while loading pages.

```cs
protected override void
OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
{
    m_window = new MainWindow();
    var rootFrame = new Frame();
    rootFrame.NavigationFailed += RootFrame_NavigationFailed;
    rootFrame.Navigate(typeof(MainPage), args);
    m_window.Content = rootFrame;
    m_window.Activate();
}

private void RootFrame_NavigationFailed(object sender,
NavigationFailedEventArgs e)
{
    throw new Exception($"Error loading page
{e.SourcePageType.FullName}");
}
```

- You'll also need to add four using statements to the top of the App.xaml.cs file to resolve compilation errors in the code.

cs

```
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml.Navigation;
using System;
using WindowsHelloLogin.Views;
```

- Right-click the new **Views** folder, select **Add** > **New Item** and select the **Blank Page** template. Name this page "Login.xaml".
- To define the user interface for the new login page, add the following XAML. This XAML defines a `StackPanel` to align the following children:
  - A `TextBlock` that will contain a title.
  - A `TextBlock` for error messages.
  - A `TextBox` for the username to be input.
  - A `Button` to navigate to a register page.
  - A `TextBlock` to contain the status of Windows Hello.
  - A `TextBlock` to explain the Login page, as there is no backend or configured users yet.

XML

```
<Grid>
  <StackPanel>
    <TextBlock Text="Login" FontSize="36" Margin="4"
TextAlignment="Center"/>
    <TextBlock x:Name="ErrorMessage" Text="" FontSize="20"
Margin="4" Foreground="Red" TextAlignment="Center"/>
    <TextBlock Text="Enter your username below" Margin="0,0,0,20"
               TextWrapping="Wrap" Width="300"
               TextAlignment="Center" VerticalAlignment="Center"
FontSize="16"/>
    <TextBox x:Name="UsernameTextBox" Margin="4" Width="250"/>
    <Button x:Name="LoginButton" Content="Login"
Background="DodgerBlue" Foreground="White"
            Click="LoginButton_Click" Width="80"
HorizontalAlignment="Center" Margin="0,20"/>
    <TextBlock Text="Don't have an account?"
               TextAlignment="Center" VerticalAlignment="Center"
FontSize="16"/>
    <TextBlock x:Name="RegisterButtonTextBlock" Text="Register now"

PointerPressed="RegisterButtonTextBlock_OnPointerPressed"
               Foreground="DodgerBlue"
```

```
                TextAlignment="Center" VerticalAlignment="Center"
  FontSize="16"/>
      <Border x:Name="WindowsHelloStatus" Background="#22B14C"
              Margin="0,20" Height="100" >
        <TextBlock x:Name="WindowsHelloStatusText" Text="Windows Hello
  is ready to use!"
                   Margin="4" TextAlignment="Center"
  VerticalAlignment="Center" FontSize="20"/>
      </Border>
      <TextBlock x:Name="LoginExplanation" FontSize="24"
  TextAlignment="Center" TextWrapping="Wrap"
                 Text="Please Note: To demonstrate a login, validation
  will only occur using the default username 'sampleUsername'"/>
    </StackPanel>
  </Grid>
```

- A few methods need to be added to the code-behind file to get the solution building. Either press F7 or use the **Solution Explorer** to edit the Login.xaml.cs file. Add in the following two event methods to handle the **Login** and **Register** events. For now these methods will set the `ErrorMessage.Text` to an empty string. Be sure to include the following using statements. They will be needed for the next steps.

cs

```
using Microsoft.UI.Xaml;
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml.Input;
using Microsoft.UI.Xaml.Media;
using Microsoft.UI.Xaml.Navigation;
using System;

namespace WindowsHelloLogin.Views
{
    public sealed partial class Login : Page
    {
        public Login()
        {
            this.InitializeComponent();
        }
        private void LoginButton_Click(object sender, RoutedEventArgs
e)
        {
            ErrorMessage.Text = "";
        }
        private void RegisterButtonTextBlock_OnPointerPressed(object
sender, PointerRoutedEventArgs e)
        {
            ErrorMessage.Text = "";
        }
    }
}
```

- In order to render the **Login** page, edit the **MainPage** code to navigate to the **Login** page when the **MainPage** is loaded. Open the MainPage.xaml.cs file. In **Solution Explorer**, double-click on MainPage.xaml.cs. If you can't find this click the little arrow next to MainPage.xaml to show the code-behind file. Create a **Loaded** event handler method that will navigate to the **Login** page.

```cs
namespace WindowsHelloLogin.Views
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            Loaded += MainPage_Loaded;
        }

        private void MainPage_Loaded(object sender, RoutedEventArgs e)
        {
            Frame.Navigate(typeof(Login));
        }
    }
}
```

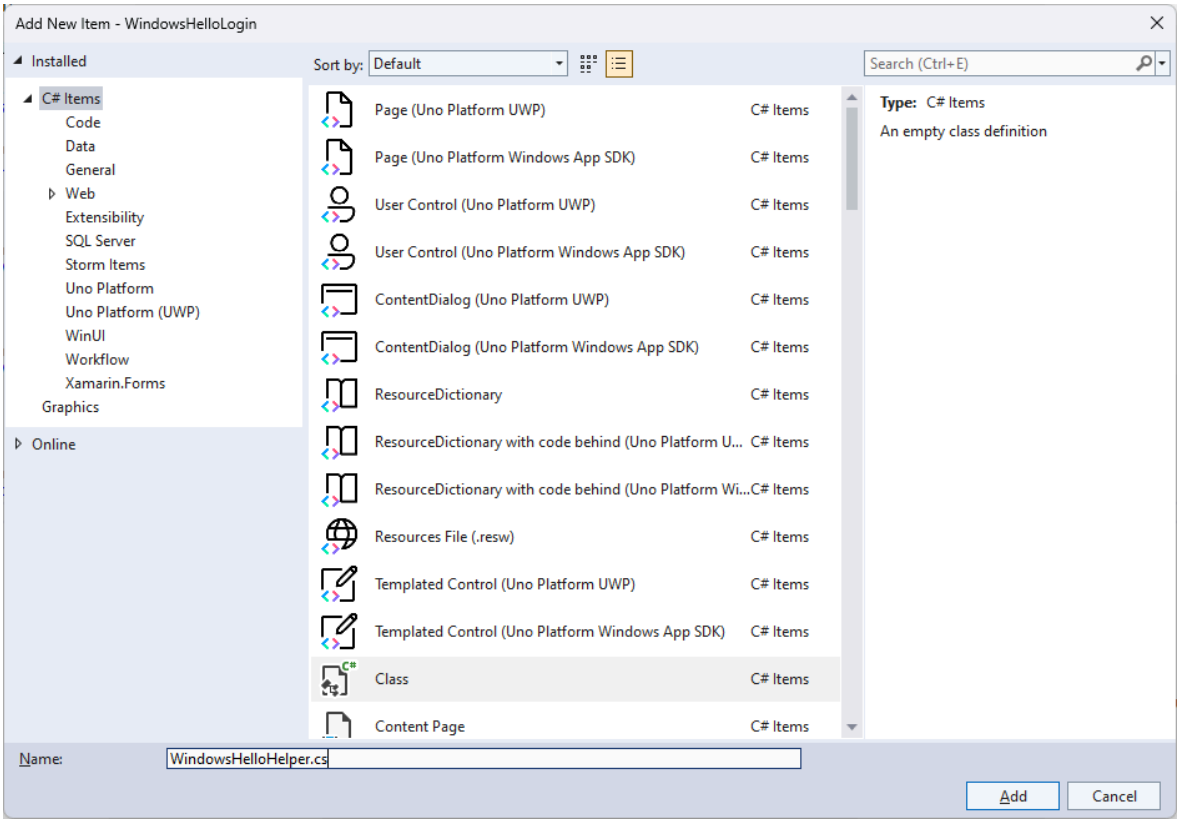
- In the **Login** page, you need to handle the `OnNavigatedTo` event to validate if Windows Hello is available on the current machine. In Login.xaml.cs, implement the following code. You'll notice that the WindowsHelloHelper object indicates that there's an error. That's because we haven't created this helper class yet.

```cs
public sealed partial class Login : Page
{
    public Login()
    {
        this.InitializeComponent();
    }

    protected override async void OnNavigatedTo(NavigationEventArgs e)
    {
        // Check if Windows Hello is set up and available on this
machine
        if (await WindowsHelloHelper.WindowsHelloAvailableCheckAsync())
        {
        }
        else
        {
            // Windows Hello isn't set up, so inform the user
            WindowsHelloStatus.Background = new
```

```
SolidColorBrush(Windows.UI.Color.FromArgb(255, 50, 170, 207));
            WindowsHelloStatusText.Text = $"Windows Hello is not set
up!{Environment.NewLine}Please go to Windows Settings and set up a PIN
to use it.";
            LoginButton.IsEnabled = false;
        }
    }
}
```

- To create the WindowsHelloHelper class, right-click the **WindowsHelloLogin** project and click **Add** > **New Folder**. Name this folder **Utils**.

- Right click the **Utils** folder and select **Add** > **Class**. Name this new class "WindowsHelloHelper.cs".



- Change the scope of the **WindowsHelloHelper** class to be `public static`, then add the following method that to inform the user if Windows Hello is ready to be used or not. You will need to add the required namespaces.

cs

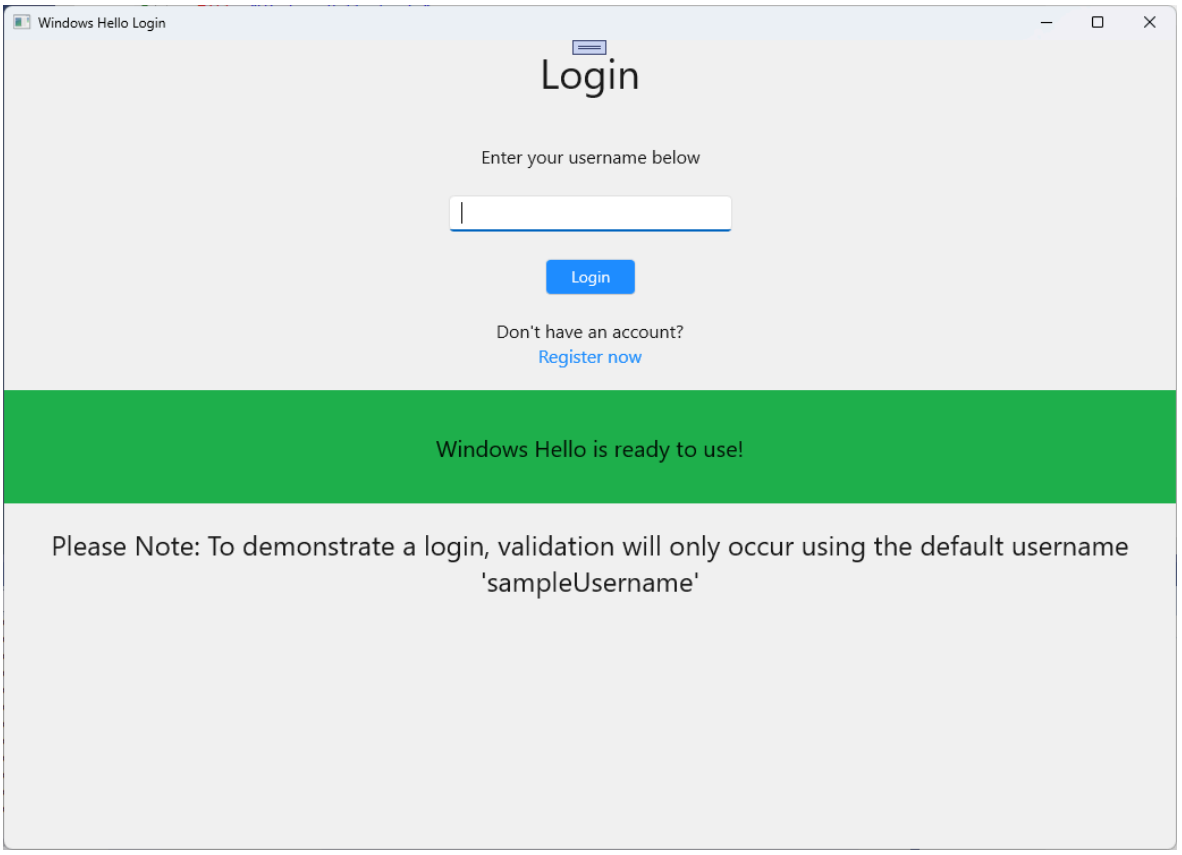
```
using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.Security.Credentials;

namespace WindowsHelloLogin.Utils
{
    public static class WindowsHelloHelper
```

```
    {
        /// <summary>
        /// Checks to see if Windows Hello is ready to be used.
        ///
        /// Windows Hello has dependencies on:
        ///     1. Having a connected Microsoft Account
        ///     2. Having a Windows PIN set up for that account on the
local machine
        /// </summary>
        public static async Task<bool>
WindowsHelloAvailableCheckAsync()
        {
            bool keyCredentialAvailable = await
KeyCredentialManager.IsSupportedAsync();
            if (keyCredentialAvailable == false)
            {
                // Key credential is not enabled yet as user
                // needs to connect to a Microsoft Account and select a
PIN in the connecting flow.
                Debug.WriteLine("Windows Hello is not set up!\nPlease
go to Windows Settings and set up a PIN to use it.");
                return false;
            }

            return true;
        }
    }
}
```

- In Login.xaml.cs, add a reference to the `WindowsHelloLogin.Utils` namespace. This will resolve the error in the `OnNavigatedTo` method.

```cs
using WindowsHelloLogin.Utils;
```

- Build and run the application. You will be navigated to the login page and the Windows Hello banner will indicate to you if Windows Hello is ready to be used. You should see either the green or blue banner indicating the Windows Hello status on your machine.



- The next thing you need to do is build the logic for signing in. Create a new folder in the project named "Models".
- In the **Models** folder, create a new class called "Account.cs". This class will act as your account model. As this is a sample project, it will only contain a username. Change the class scope to `public` and add the `Username` property.

cs

```cs
namespace WindowsHelloLogin.Models
{
    public class Account
    {
        public string Username { get; set; }
    }
}
```

- The app needs a way to handle accounts. For this hands on lab, as there is no server or database, a list of users are saved and loaded locally. Right-click the **Utils** folder and add a new class named "AccountHelper.cs". Change the class scope to be `public static`. The **AccountHelper** is a static class that contains all the necessary methods to save and load the list of accounts locally. Saving and loading works by using an **XmlSerializer**. You also need to remember the file that was saved and where you saved it.

cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;
using Windows.Storage;
using WindowsHelloLogin.Models;

namespace WindowsHelloLogin.Utils
{
    public static class AccountHelper
    {
        // In the real world this would not be needed as there would be
a server implemented that would host a user account database.
        // For this tutorial we will just be storing accounts locally.
        private const string USER_ACCOUNT_LIST_FILE_NAME =
"accountlist.txt";
        private static string _accountListPath =
Path.Combine(ApplicationData.Current.LocalFolder.Path,
USER_ACCOUNT_LIST_FILE_NAME);
        public static List<Account> AccountList = [];

        /// <summary>
        /// Create and save a useraccount list file. (Updating the old
one)
        /// </summary>
        private static async void SaveAccountListAsync()
        {
            string accountsXml = SerializeAccountListToXml();

            if (File.Exists(_accountListPath))
            {
                StorageFile accountsFile = await
StorageFile.GetFileFromPathAsync(_accountListPath);
                await FileIO.WriteTextAsync(accountsFile, accountsXml);
            }
            else
            {
                StorageFile accountsFile = await
ApplicationData.Current.LocalFolder.CreateFileAsync(USER_ACCOUNT_LIST_F
ILE_NAME);
                await FileIO.WriteTextAsync(accountsFile, accountsXml);
            }
        }

        /// <summary>
        /// Gets the useraccount list file and deserializes it from XML
to a list of useraccount objects.
        /// </summary>
        /// <returns>List of useraccount objects</returns>
        public static async Task<List<Account>> LoadAccountListAsync()
        {
            if (File.Exists(_accountListPath))
```

```
            {
                StorageFile accountsFile = await
StorageFile.GetFileFromPathAsync(_accountListPath);

                string accountsXml = await
FileIO.ReadTextAsync(accountsFile);
                DeserializeXmlToAccountList(accountsXml);
            }

            return AccountList;
        }

        /// <summary>
        /// Uses the local list of accounts and returns an XML
formatted string representing the list
        /// </summary>
        /// <returns>XML formatted list of accounts</returns>
        public static string SerializeAccountListToXml()
        {
            var xmlizer = new XmlSerializer(typeof(List<Account>));
            var writer = new StringWriter();
            xmlizer.Serialize(writer, AccountList);

            return writer.ToString();
        }

        /// <summary>
        /// Takes an XML formatted string representing a list of
accounts and returns a list object of accounts
        /// </summary>
        /// <param name="listAsXml">XML formatted list of
accounts</param>
        /// <returns>List object of accounts</returns>
        public static List<Account> DeserializeXmlToAccountList(string
listAsXml)
        {
            var xmlizer = new XmlSerializer(typeof(List<Account>));
            TextReader textreader = new StreamReader(new
MemoryStream(Encoding.UTF8.GetBytes(listAsXml)));

            return AccountList = (xmlizer.Deserialize(textreader)) as
List<Account>;
        }
    }
}
```

- Next, implement a way to add and remove an account from the local list of accounts. These actions will each save the list. The final method that you'll need for this hands-on lab is a validation method. As there is no authorization server or database of users, this will validate against a single user which is hard-coded. These methods should be added to the **AccountHelper** class.

cs

```cs
public static Account AddAccount(string username)
{
    // Create a new account with the username
    var account = new Account() { Username = username };
    // Add it to the local list of accounts
    AccountList.Add(account);
    // SaveAccountList and return the account
    SaveAccountListAsync();
    return account;
}

public static void RemoveAccount(Account account)
{
    // Remove the account from the accounts list
    AccountList.Remove(account);
    // Re save the updated list
    SaveAccountListAsync();
}

public static bool ValidateAccountCredentials(string username)
{
    // In the real world, this method would call the server to
authenticate that the account exists and is valid.
    // However, for this tutorial, we'll just have an existing sample
user that's named "sampleUsername".
    // If the username is null or does not match "sampleUsername"
validation will fail.
    // In this case, the user should register a new Windows Hello user.

    if (string.IsNullOrEmpty(username))
    {
        return false;
    }

    if (!string.Equals(username, "sampleUsername"))
    {
        return false;
    }

    return true;
}
```

- The next thing you need to do is handle a sign in request from the user. In Login.xaml.cs, create a new private variable that will hold the current account logging in. Then add a new method named **SignInWindowsHelloAsync**. This will validate the account credentials using the **AccountHelper.ValidateAccountCredentials** method. This method will return a **Boolean** value if the entered user name is the same as the hard coded string value

you configured in the previous step. The hard-coded value for this sample is "sampleUsername".

cs

```
using WindowsHelloLogin.Models;
using WindowsHelloLogin.Utils;
using System.Diagnostics;
using System.Threading.Tasks;

namespace WindowsHelloLogin.Views
{
    public sealed partial class Login : Page
    {
        private Account _account;

        public Login()
        {
            this.InitializeComponent();
        }

        protected override async void OnNavigatedTo(NavigationEventArgs
e)
        {
            // Check if Windows Hello is set up and available on this
machine
            if (await
WindowsHelloHelper.WindowsHelloAvailableCheckAsync())
            {
            }
            else
            {
                // Windows Hello is not set up, so inform the user
                WindowsHelloStatus.Background = new
SolidColorBrush(Windows.UI.Color.FromArgb(255, 50, 170, 207));
                WindowsHelloStatusText.Text = "Windows Hello is not set
up!\nPlease go to Windows Settings and set up a PIN to use it.";
                LoginButton.IsEnabled = false;
            }
        }

        private async void LoginButton_Click(object sender,
RoutedEventArgs e)
        {
            ErrorMessage.Text = "";
            await SignInWindowsHelloAsync();
        }

        private void RegisterButtonTextBlock_OnPointerPressed(object
sender, PointerRoutedEventArgs e)
        {
            ErrorMessage.Text = "";
        }
```

```
        private async Task SignInWindowsHelloAsync()
        {
            if
(AccountHelper.ValidateAccountCredentials(UsernameTextBox.Text))
            {
                // Create and add a new local account
                _account =
AccountHelper.AddAccount(UsernameTextBox.Text);
                Debug.WriteLine("Successfully signed in with
traditional credentials and created local account instance!");

                //if (await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(UsernameTextBox.Text))
                //{
                //    Debug.WriteLine("Successfully signed in with
Windows Hello!");
                //}
            }
            else
            {
                ErrorMessage.Text = "Invalid Credentials";
            }
        }
    }
}
```

- You may have noticed the commented code that was referencing a method in **WindowsHelloHelper**. In WindowsHelloHelper.cs, add a new method named **CreateWindowsHelloKeyAsync**. This method uses the Windows Hello API in the KeyCredentialManager. Calling RequestCreateAsync will create a Windows Hello key that is specific to the *accountId* and the local machine. Please note the comments in the switch statement if you are interested in implementing this in a real world scenario.

cs

```
/// <summary>
/// Creates a Windows Hello key on the machine using the account ID
provided.
/// </summary>
/// <param name="accountId">The account ID associated with the account
that we are enrolling into Windows Hello</param>
/// <returns>Boolean indicating if creating the Windows Hello key
succeeded</returns>
public static async Task<bool> CreateWindowsHelloKeyAsync(string
accountId)
{
    KeyCredentialRetrievalResult keyCreationResult = await
KeyCredentialManager.RequestCreateAsync(accountId,
KeyCredentialCreationOption.ReplaceExisting);
```

```
    switch (keyCreationResult.Status)
    {
        case KeyCredentialStatus.Success:
            Debug.WriteLine("Successfully created key");

            // In the real world, authentication would take place on a
server.
            // So, every time a user migrates or creates a new Windows
Hello
            // account, details should be pushed to the server.
            // The details that would be pushed to the server include:
            // The public key, keyAttestation (if available),
            // certificate chain for attestation endorsement key (if
available),
            // status code of key attestation result:
keyAttestationIncluded or
            // keyAttestationCanBeRetrievedLater and
keyAttestationRetryType.
            // As this sample has no concept of a server, it will be
skipped for now.
            // For information on how to do this, refer to the second
sample.

            // For this sample, just return true
            return true;
        case KeyCredentialStatus.UserCanceled:
            Debug.WriteLine("User cancelled sign-in process.");
            break;
        case KeyCredentialStatus.NotFound:
            // User needs to set up Windows Hello
            Debug.WriteLine("Windows Hello is not set up!\nPlease go to
Windows Settings and set up a PIN to use it.");
            break;
        default:
            break;
    }

    return false;
}
```

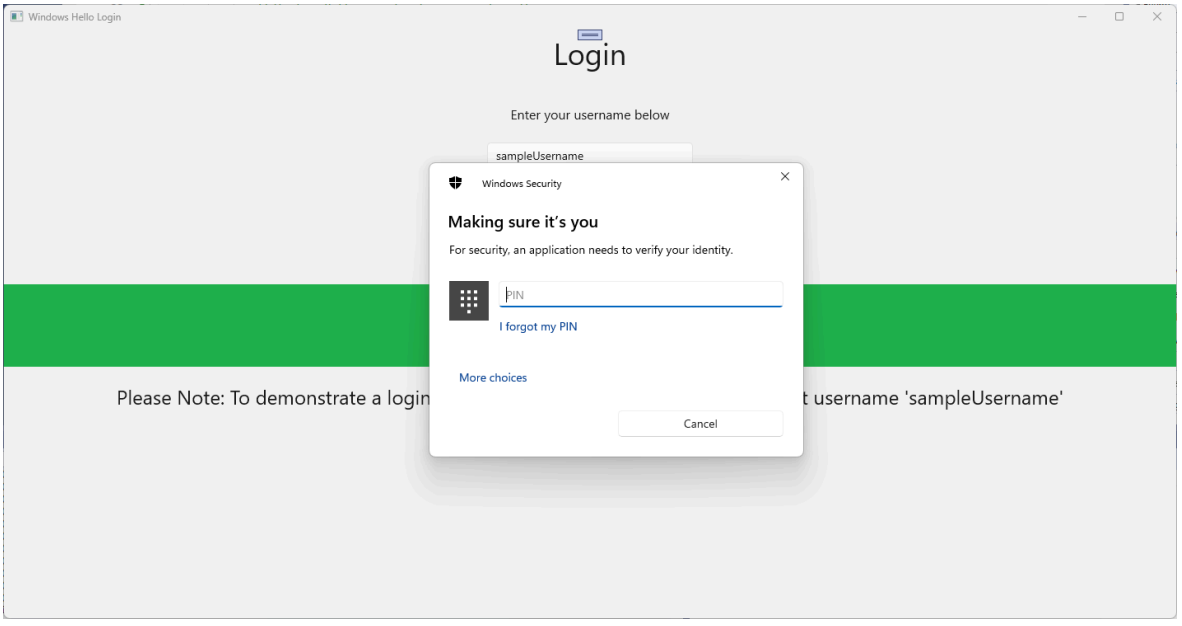
- Now that you have created the **CreateWindowsHelloKeyAsync** method, return to the Login.xaml.cs file and uncomment the code inside the SignInWindowsHelloAsync method.

cs

```
private async void SignInWindowsHelloAsync()
{
    if (AccountHelper.ValidateAccountCredentials(UsernameTextBox.Text))
    {
        //Create and add a new local account
        _account = AccountHelper.AddAccount(UsernameTextBox.Text);
        Debug.WriteLine("Successfully signed in with traditional
```

```
credentials and created local account instance!");

        if (await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(UsernameTextBox.Text))
        {
            Debug.WriteLine("Successfully signed in with Windows
Hello!");
        }
    }
    else
    {
        ErrorMessage.Text = "Invalid Credentials";
    }
}
```

- Build and run the application. You will be taken to the Login page. Enter the user name as "sampleUsername" and click login. You will be prompted with a Windows Hello prompt asking you to enter your PIN. Upon entering your PIN correctly, the **CreateWindowsHelloKeyAsync** method will be able to create a Windows Hello key. Monitor the output windows to see if the messages indicating success are shown.



Exercise 2: Welcome and User Selection Pages

In this exercise, you will continue from the previous exercise. When a user successfully logs in, they should be taken to a welcome page where they can sign out or delete their account. As Windows Hello creates a key for every machine, a user selection screen can be created, which displays all users that have been signed in on that machine. A user can then select one of these accounts and go directly to the welcome screen without needed to re-enter a password as they have already authenticated to access the machine.

- In the **Views** folder add a new blank page named "Welcome.xaml". Add the following XAML to complete the user interface for the page. This will display a title, the logged in username, and two buttons. One of the buttons will navigate back to a user list (that you will create later), and the other button will handle forgetting this user.

XML

```
<Grid>
  <StackPanel>
    <TextBlock x:Name="Title" Text="Welcome" FontSize="40"
TextAlignment="Center"/>
    <TextBlock x:Name="UserNameText" FontSize="28"
TextAlignment="Center"/>

    <Button x:Name="BackToUserListButton" Content="Back to User List"
Click="Button_Restart_Click"
            HorizontalAlignment="Center" Margin="0,20"
Foreground="White" Background="DodgerBlue"/>

    <Button x:Name="ForgetButton" Content="Forget Me"
Click="Button_Forget_User_Click"
            Foreground="White"
            Background="Gray"
            HorizontalAlignment="Center"/>
  </StackPanel>
</Grid>
```

- In the Welcome.xaml.cs code-behind file, add a new private variable that will hold the account that is logged in. You will need to implement a method to override the `OnNavigateTo` event, this will store the account passed to the **Welcome** page. You will also need to implement the `Click` event for the two buttons defined in the XAML. You will need to add using statements for the `WindowsHelloLogin.Models` and `WindowsHelloLogin.Utils` namespaces.

cs

```
using WindowsHelloLogin.Models;
using WindowsHelloLogin.Utils;
using System.Diagnostics;

namespace WindowsHelloLogin.Views
{
    public sealed partial class Welcome : Page
    {
        private Account _activeAccount;

        public Welcome()
        {
```

```
            InitializeComponent();
        }

        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            _activeAccount = (Account)e.Parameter;
            if (_activeAccount != null)
            {
                UserNameText.Text = _activeAccount.Username;
            }
        }

        private void Button_Restart_Click(object sender,
RoutedEventArgs e)
        {
        }

        private void Button_Forget_User_Click(object sender,
RoutedEventArgs e)
        {
            // Remove the account from Windows Hello
            //
WindowsHelloHelper.RemoveWindowsHelloAccountAsync(_activeAccount);

            // Remove it from the local accounts list and re-save the
updated list
            AccountHelper.RemoveAccount(_activeAccount);

            Debug.WriteLine($"User {_activeAccount.Username}
deleted.");
        }
    }
}
```

- You may have noticed a line commented out in the `Button_Forget_User_Click` event handler. The account is being removed from your local list but currently there is no way to be removed from Windows Hello. You need to implement a new method in WindowsHelloHelper.cs that will handle removing a Windows Hello user. This method will use other Windows Hello APIs to open and delete the account. In the real world, when you delete an account the server or database should be notified so the user database remains valid. You will need a using statement referencing the `WindowsHelloLogin.Models` namespace.

cs

```
using WindowsHelloLogin.Models;

/// <summary>
/// Function to be called when user requests deleting their account.
/// Checks the KeyCredentialManager to see if there is a Windows Hello
/// account for the current user.
```

```
/// It then deletes the local key associated with the account.
/// </summary>
public static async void RemoveWindowsHelloAccountAsync(Account
account)
{
    // Open the account with Windows Hello
    KeyCredentialRetrievalResult keyOpenResult = await
KeyCredentialManager.OpenAsync(account.Username);

    if (keyOpenResult.Status == KeyCredentialStatus.Success)
    {
        // In the real world you would send key information to server
to unregister
        //for example, RemoveWindowsHelloAccountOnServer(account);
    }

    // Then delete the account from the machine's list of Windows Hello
accounts
    await KeyCredentialManager.DeleteAsync(account.Username);
}
```

- Back in Welcome.xaml.cs, uncomment the line that calls **RemoveWindowsHelloAccountAsync**.

cs

```
private void Button_Forget_User_Click(object sender, RoutedEventArgs e)
{
    // Remove it from Windows Hello
    WindowsHelloHelper.RemoveWindowsHelloAccountAsync(_activeAccount);

    // Remove it from the local accounts list and re-save the updated
list
    AccountHelper.RemoveAccount(_activeAccount);

    Debug.WriteLine($"User {_activeAccount.Username} deleted.");
}
```

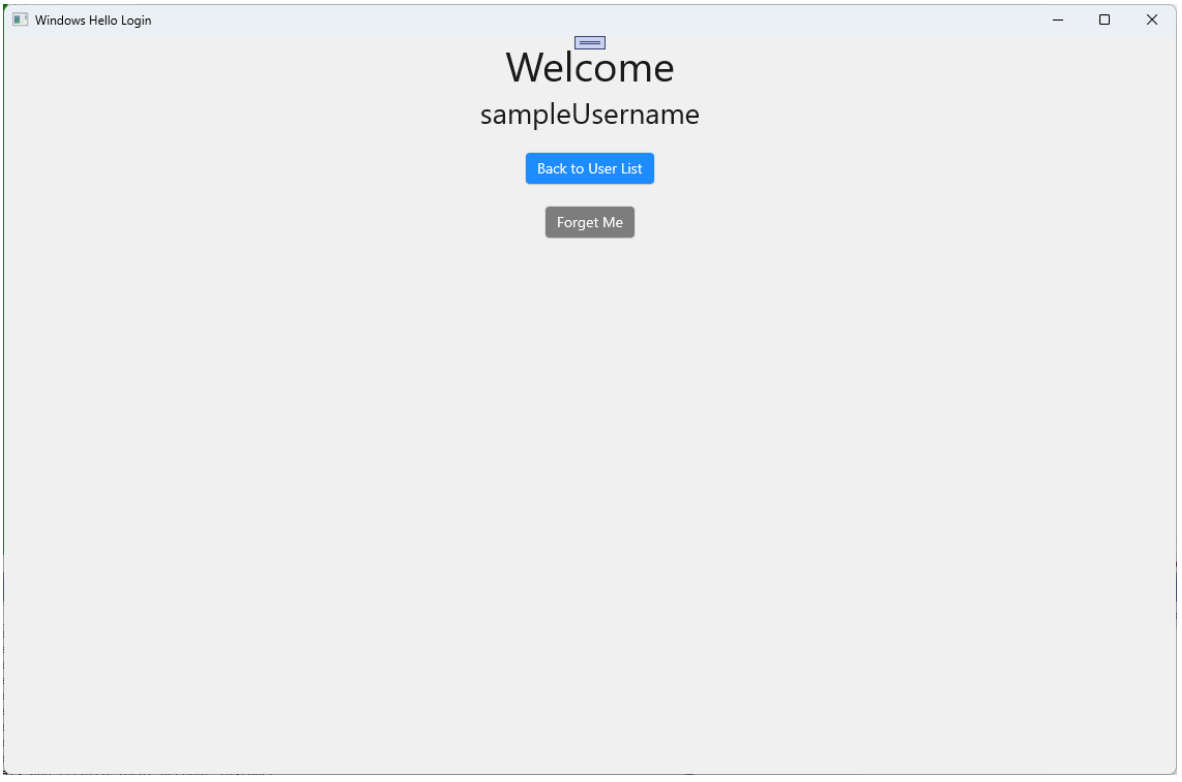
- In the **SignInWindowsHelloAsync** method (in Login.xaml.cs), once the **CreateWindowsHelloKeyAsync** is successful, it should navigate to the **Welcome** page and pass the **Account**.

cs

```
private async void SignInWindowsHelloAsync()
{
    if (AccountHelper.ValidateAccountCredentials(UsernameTextBox.Text))
    {
        // Create and add a new local account
        _account = AccountHelper.AddAccount(UsernameTextBox.Text);
        Debug.WriteLine("Successfully signed in with traditional
```

```
credentials and created local account instance!");

        if (await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(UsernameTextBox.Text))
        {
            Debug.WriteLine("Successfully signed in with Windows
Hello!");
            Frame.Navigate(typeof(Welcome), _account);
        }
    }
    else
    {
        ErrorMessage.Text = "Invalid Credentials";
    }
}
```

- Build and run the application. Login with "sampleUsername" and click **Login**. Enter your PIN and if successful you should be navigated to the **Welcome** screen. Try clicking **Forget User** and monitor Visual Studio's **Output** window to see if the user was deleted. Notice that when the user is deleted, you remain on the **Welcome** page. You'll need to create a user selection page to which the app can navigate.



- In the **Views** folder, create a new blank page named "UserSelection.xaml" and add the following XAML to define the user interface. This page will contain a ListView that displays all the users in the local accounts list, and a `Button` that will navigate to the **Login** page to allow the user to add another account.

```
XML
```

```
<Grid>
  <StackPanel>
    <TextBlock x:Name="Title" Text="Select a User" FontSize="36"
Margin="4" TextAlignment="Center" HorizontalAlignment="Center"/>

    <ListView x:Name="UserListView" Margin="4" MaxHeight="200"
MinWidth="250" Width="250" HorizontalAlignment="Center">
      <ListView.ItemTemplate>
        <DataTemplate>
          <Grid Background="DodgerBlue" Height="50" Width="250"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
            <TextBlock Text="{Binding Username}"
HorizontalAlignment="Center" TextAlignment="Center"
VerticalAlignment="Center" Foreground="White"/>
          </Grid>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>

    <Button x:Name="AddUserButton" Content="+" FontSize="36" Width="60"
Click="AddUserButton_Click" HorizontalAlignment="Center"/>
  </StackPanel>
</Grid>
```

- In UserSelection.xaml.cs, implement the `Loaded` method that will navigate to the **Login** page if there are no accounts in the local list. Also implement the `SelectionChanged` event for the `ListView` and a `Click` event for the `Button`.

cs

```
using System.Diagnostics;
using WindowsHelloLogin.Models;
using WindowsHelloLogin.Utils;

namespace WindowsHelloLogin.Views
{
    public sealed partial class UserSelection : Page
    {
        public UserSelection()
        {
            InitializeComponent();
            Loaded += UserSelection_Loaded;
        }

        private void UserSelection_Loaded(object sender,
RoutedEventArgs e)
        {
            if (AccountHelper.AccountList.Count == 0)
            {
                // If there are no accounts, navigate to the Login page
                Frame.Navigate(typeof(Login));
```

```
            }


            UserListView.ItemsSource = AccountHelper.AccountList;
            UserListView.SelectionChanged += UserSelectionChanged;
        }

        /// <summary>
        /// Function called when an account is selected in the list of
accounts
        /// Navigates to the Login page and passes the chosen account
        /// </summary>
        private void UserSelectionChanged(object sender,
RoutedEventArgs e)
        {
            if (((ListView)sender).SelectedValue != null)
            {
                Account account = (Account)
((ListView)sender).SelectedValue;
                if (account != null)
                {
                    Debug.WriteLine($"Account {account.Username}
selected!");
                }
                Frame.Navigate(typeof(Login), account);
            }
        }

        /// <summary>
        /// Function called when the "+" button is clicked to add a new
user.
        /// Navigates to the Login page with nothing filled out
        /// </summary>
        private void AddUserButton_Click(object sender, RoutedEventArgs
e)
        {
            Frame.Navigate(typeof(Login));
        }
    }
}
```

- There are a few places in the app where you want to navigate to the **UserSelection** page. In MainPage.xaml.cs, you should navigate to the **UserSelection** page instead of the **Login** page. While you are in the loaded event in **MainPage**, you will need to load the accounts list so the **UserSelection** page can check if there are any accounts. This will require changing the `Loaded` method to be async and also adding a using statement for the `WindowsHelloLogin.Utils` namespace.

```
cs
```

```
using WindowsHelloLogin.Utils;

private async void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    // Load the local account list before navigating to the
UserSelection page
    await AccountHelper.LoadAccountListAsync();
    Frame.Navigate(typeof(UserSelection));
}
```

- Next, the app will need navigate to the **UserSelection** page from the **Welcome** page. In both `Click` events, you should navigate back to the **UserSelection** page.

cs

```
private void Button_Restart_Click(object sender, RoutedEventArgs e)
{
    Frame.Navigate(typeof(UserSelection));
}

private void Button_Forget_User_Click(object sender, RoutedEventArgs e)
{
    // Remove it from Windows Hello
    WindowsHelloHelper.RemoveWindowsHelloAccountAsync(_activeAccount);

    // Remove it from the local accounts list and re-save the updated
list
    AccountHelper.RemoveAccount(_activeAccount);

    Debug.WriteLine($"User {_activeAccount.Username} deleted.");

    // Navigate back to UserSelection page.
    Frame.Navigate(typeof(UserSelection));
}
```

- In the **Login** page, you need code to log in to the account selected from the list in the **UserSelection** page. In the `OnNavigatedTo` event, store the account passed during navigation. Start by adding a new private variable that will identify if the account is an existing account. Then handle the `OnNavigatedTo` event.

cs

```
namespace WindowsHelloLogin.Views
{
    public sealed partial class Login : Page
    {
        private Account _account;
        private bool _isExistingAccount;
```

```
        public Login()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Function called when this frame is navigated to.
        /// Checks to see if Windows Hello is available and if an
account was passed in.
        /// If an account was passed in set the "_isExistingAccount"
flag to true and set the _account.
        /// </summary>
        protected override async void OnNavigatedTo(NavigationEventArgs
e)
        {
            // Check Windows Hello is set up and available on this
machine
            if (await
WindowsHelloHelper.WindowsHelloAvailableCheckAsync())
            {
                if (e.Parameter != null)
                {
                    _isExistingAccount = true;
                    // Set the account to the existing account being
passed in
                    _account = (Account)e.Parameter;
                    UsernameTextBox.Text = _account.Username;
                    await SignInWindowsHelloAsync();
                }
            }
            else
            {
                // Windows Hello is not set up, so inform the user
                WindowsHelloStatus.Background = new
SolidColorBrush(Windows.UI.Color.FromArgb(255, 50, 170, 207));
                WindowsHelloStatusText.Text = $"Windows Hello is not
set up!{Environment.NewLine}Please go to Windows Settings and set up a
PIN to use it.";
                LoginButton.IsEnabled = false;
            }
        }
    }
}
```

- The **SignInWindowsHelloAsync** method will need to be updated to sign in to the selected account. The **WindowsHelloHelper** will need another method to open the account with Windows Hello, as the account already has an account key created for it. Implement the new method in WindowsHelloHelper.cs to sign in an existing user with Windows Hello. For information on each part of the code, please read through the code comments.

```
cs
```

```
/// <summary>
/// Attempts to sign a message using the account key on the system for
the accountId passed.
/// </summary>
/// <returns>Boolean representing if creating the Windows Hello
authentication message succeeded</returns>
public static async Task<bool>
GetWindowsHelloAuthenticationMessageAsync(Account account)
{
    KeyCredentialRetrievalResult openKeyResult = await
KeyCredentialManager.OpenAsync(account.Username);
    // Calling OpenAsync will allow the user access to what is
available in the app and will not require user credentials again.
    // If you wanted to force the user to sign in again you can use the
following:
    // var consentResult = await
Windows.Security.Credentials.UI.UserConsentVerifier.RequestVerification
Async(account.Username);
    // This will ask for the either the password of the currently
signed in Microsoft Account or the PIN used for Windows Hello.

    if (openKeyResult.Status == KeyCredentialStatus.Success)
    {
        // If OpenAsync has succeeded, the next thing to think about is
whether the client application requires access to backend services.
        // If it does here you would request a challenge from the
server. The client would sign this challenge and the server
        // would check the signed challenge. If it is correct, it would
allow the user access to the backend.
        // You would likely make a new method called RequestSignAsync
to handle all this.
        // For example, RequestSignAsync(openKeyResult);
        // Refer to the second Windows Hello sample for information on
how to do this.

        // For this sample, there is not concept of a server
implemented so just return true.
        return true;
    }
    else if (openKeyResult.Status == KeyCredentialStatus.NotFound)
    {
        // If the account is not found at this stage. It could be one
of two errors.
        // 1. Windows Hello has been disabled
        // 2. Windows Hello has been disabled and re-enabled cause the
Windows Hello Key to change.
        // Calling CreateWindowsHelloKeyAsync and passing through the
account will attempt to replace the existing Windows Hello Key for that
account.
        // If the error really is that Windows Hello is disabled then
the CreateWindowsHelloKeyAsync method will output that error.
        if (await CreateWindowsHelloKeyAsync(account.Username))
        {
            // If the Hello Key was again successfully created, Windows
```

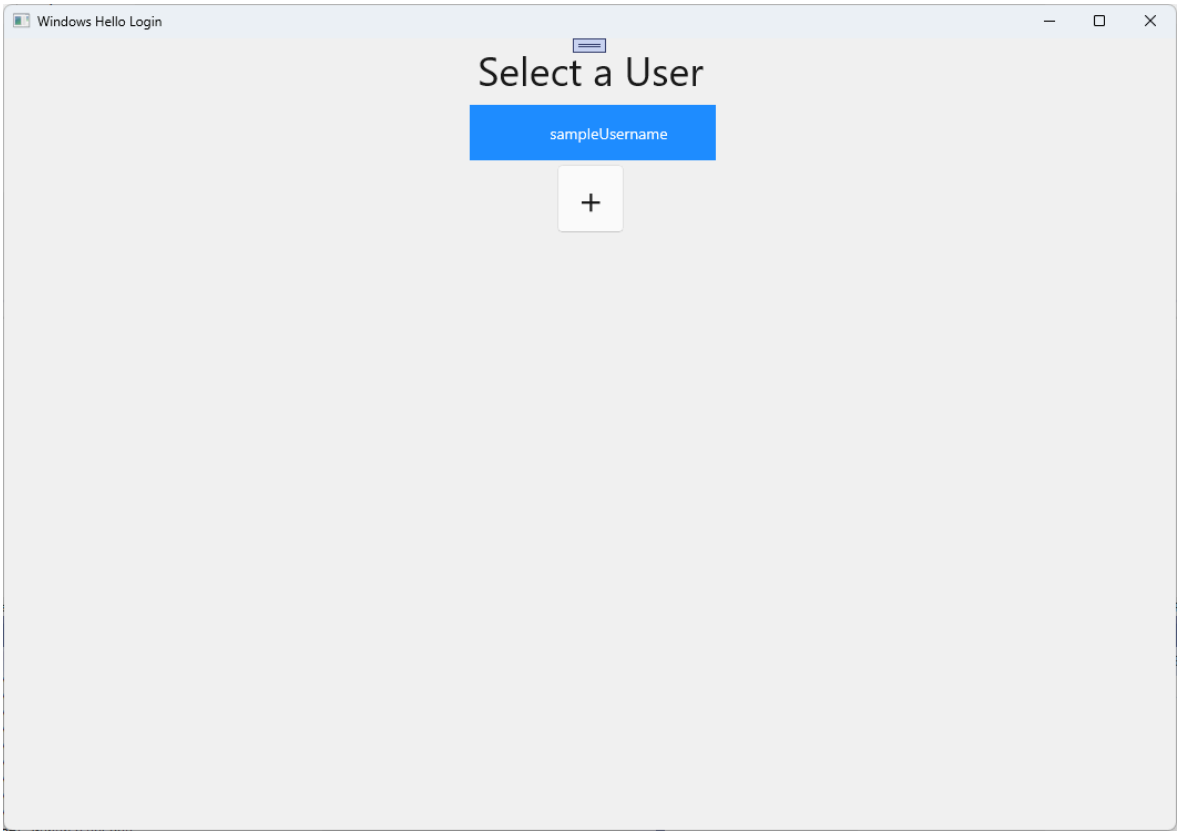
```
Hello has just been reset.
            // Now that the Hello Key has been reset for the account
retry sign in.
            return await
GetWindowsHelloAuthenticationMessageAsync(account);
        }
    }

    // Can't use Windows Hello right now, try again later
    return false;
}
```

- Update the **SignInWindowsHelloAsync** method in Login.xaml.cs to handle the existing account. This will use the new method in the WindowsHelloHelper.cs. If successful the account will be signed in and the user navigated to the **Welcome** page.

cs

```
private async Task SignInWindowsHelloAsync()
{
    if (_isExistingAccount)
    {
        if (await
WindowsHelloHelper.GetWindowsHelloAuthenticationMessageAsync(_account))
        {
            Frame.Navigate(typeof(Welcome), _account);
        }
    }
    else if
(AccountHelper.ValidateAccountCredentials(UsernameTextBox.Text))
    {
        //Create and add a new local account
        _account = AccountHelper.AddAccount(UsernameTextBox.Text);
        Debug.WriteLine("Successfully signed in with traditional
credentials and created local account instance!");

        if (await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(UsernameTextBox.Text))
        {
            Debug.WriteLine("Successfully signed in with Windows
Hello!");
            Frame.Navigate(typeof(Welcome), _account);
        }
    }
    else
    {
        ErrorMessage.Text = "Invalid Credentials";
    }
}
```

- Build and run the application. Login with "sampleUsername". Type in your PIN and if successful you will be navigated to the **Welcome** page. Click back to user list. You should now see a user in the list. If you click on this, WindowsHello enables you to sign back in without having to re-enter any passwords etc.



Exercise 3: Registering a new Windows Hello user

In this exercise, you create a new page that can create a new account with Windows Hello. This works similarly to how the **Login** page works. The **Login** page is implemented for an existing user that is migrating to use Windows Hello. A **WindowsHelloRegister** page will create Windows Hello registration for a new user.

- In the **Views** folder, create a new blank page named "WindowsHelloRegister.xaml". In the XAML add in the following to setup the user interface. The interface on this page is similar to the **Login** page.

XML

```
<Grid>
  <StackPanel>
    <TextBlock x:Name="Title" Text="Register New Windows Hello User"
FontSize="24" Margin="4" TextAlignment="Center"/>

    <TextBlock x:Name="ErrorMessage" Text="" FontSize="20" Margin="4"
```

```
Foreground="Red" TextAlignment="Center"/>

    <TextBlock Text="Enter your new username below" Margin="0,0,0,20"
               TextWrapping="Wrap" Width="300"
               TextAlignment="Center" VerticalAlignment="Center"
FontSize="16"/>

    <TextBox x:Name="UsernameTextBox" Margin="4" Width="250"/>

    <Button x:Name="RegisterButton" Content="Register"
Background="DodgerBlue" Foreground="White"
            Click="RegisterButton_Click_Async" Width="80"
HorizontalAlignment="Center" Margin="0,20"/>

    <Border x:Name="WindowsHelloStatus" Background="#22B14C" Margin="4"
Height="100">
      <TextBlock x:Name="WindowsHelloStatusText" Text="Windows Hello is
ready to use!" FontSize="20"
                 Margin="4" TextAlignment="Center"
VerticalAlignment="Center"/>
    </Border>
  </StackPanel>
</Grid>
```

- In the WindowsHelloRegister.xaml.cs code-behind file, implement a private `Account` variable and a `Click` event for the register button. This will add a new local account and create a Windows Hello key.

cs

```
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml;
using WindowsHelloLogin.Models;
using WindowsHelloLogin.Utils;

namespace WindowsHelloLogin.Views
{
    public sealed partial class WindowsHelloRegister : Page
    {
        private Account _account;

        public WindowsHelloRegister()
        {
            InitializeComponent();
        }

        private async void RegisterButton_Click_Async(object sender,
RoutedEventArgs e)
        {
            ErrorMessage.Text = "";

            // In the real world, you would validate the entered
```

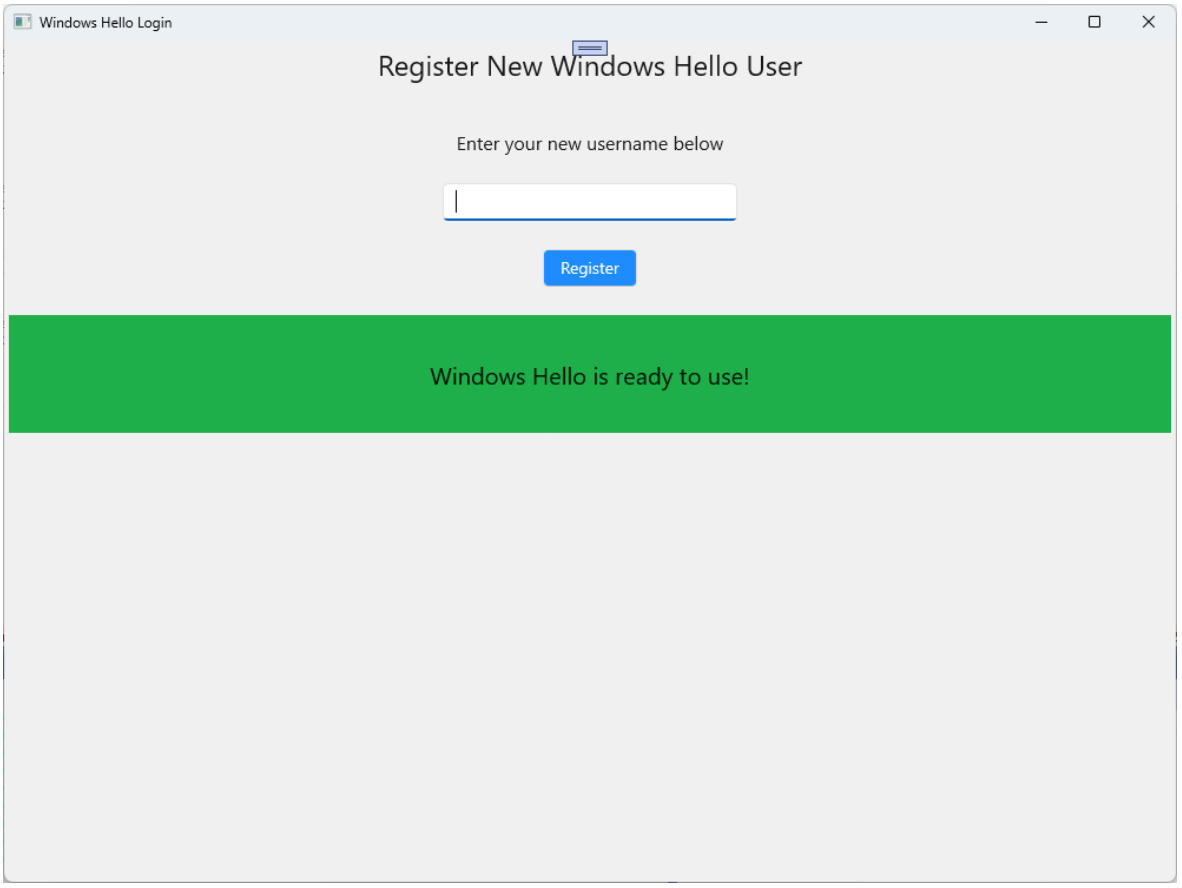
```
credentials and information before
            // allowing a user to register a new account.
            // For this sample, we'll skip that step and just register
an account if the username is not null.

            if (!string.IsNullOrEmpty(UsernameTextBox.Text))
            {
                // Register a new account
                _account =
AccountHelper.AddAccount(UsernameTextBox.Text);
                // Register new account with Windows Hello
                await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(_account.Username);
                // Navigate to the Welcome page.
                Frame.Navigate(typeof(Welcome), _account);
            }
            else
            {
                ErrorMessage.Text = "Please enter a username";
            }
        }
    }
}
```

- You need to navigate to this page from the **Login** page when register is clicked.

cs

```
private void RegisterButtonTextBlock_OnPointerPressed(object sender,
PointerRoutedEventArgs e)
{
    ErrorMessage.Text = "";
    Frame.Navigate(typeof(WindowsHelloRegister));
}
```

- Build and run the application. Try to register a new user. Then return to the user list and validate that you can select that user and login.



In this lab, you have learned the essential skills needed to use the new Windows Hello API to authenticate existing users and create accounts for new users. With this new knowledge, you can start removing the need for users to remember passwords for your application, yet remain confident that your applications remain protected by user authentication. Windows uses Windows Hello's new authentication technology to support its biometrics login options.

Related topics

- Windows Hello
- Windows Hello login service

Create a Windows Hello login app

Article • 07/11/2024

This is the first part of a complete walkthrough on how to create a packaged Windows app that uses Windows Hello as an alternative to traditional username and password authentication systems. In this case, the app is a WinUI app, but the same approach can be used with any packaged Windows app, including WPF and Windows Forms apps. The app uses a username for sign-in and creates a Hello key for each account. These accounts will be protected by the PIN that is set up in Windows Settings on configuration of Windows Hello.

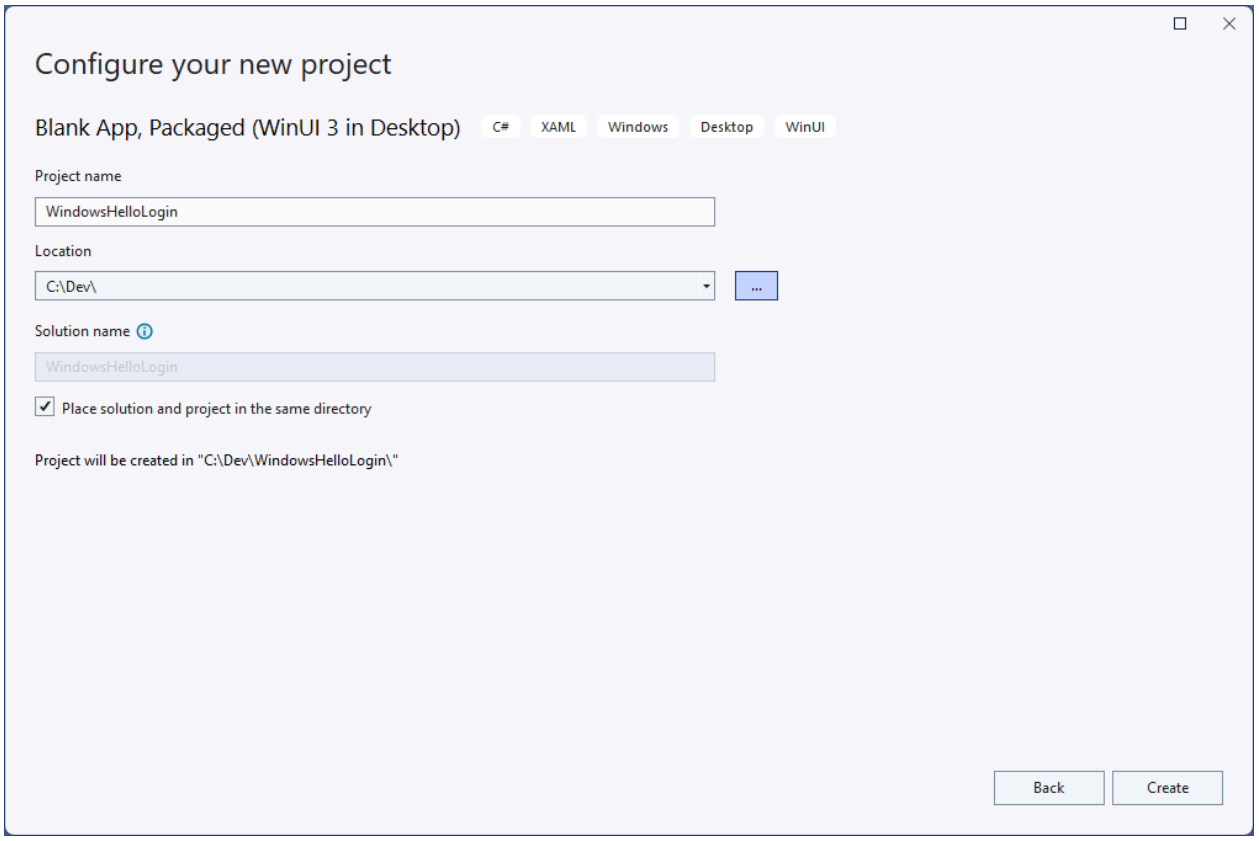
This walkthrough is split into two parts: building the app and connecting the backend service. When you're finished with this article, continue on to Part 2: Windows Hello login service.

Before you begin, you should read the Windows Hello overview for a general understanding of how Windows Hello works.

Get started

In order to build this project, you'll need some experience with C#, and XAML. You'll also need to be using Visual Studio 2022 on a Windows 10 or Windows 11 machine. See Start developing Windows apps for complete instructions on setting up your development environment.

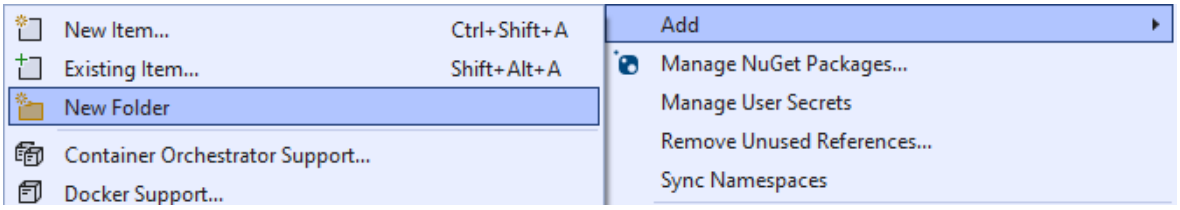
- In Visual Studio, select **File** > **New** > **Project**.
- In the **New Project** dialog's drop-down filters, select **C#/C++**, **Windows**, and **WinUI**, respectively.
- Choose **Blank App, Packaged (WinUI 3 in Desktop)** and name your application "WindowsHelloLogin".
- Build and Run the new application (F5), you should see a blank window shown on the screen. Close the application.



Exercise 1: Login with Windows Hello

In this exercise you will learn how to check if Windows Hello is setup on the machine, and how to sign into an account using Windows Hello.

- In the new project create a new folder in the solution called "Views". This folder will contain the pages that will be navigated to in this sample. Right-click the project in **Solution Explorer**, select **Add** > **New Folder**, then rename the folder to **Views**.

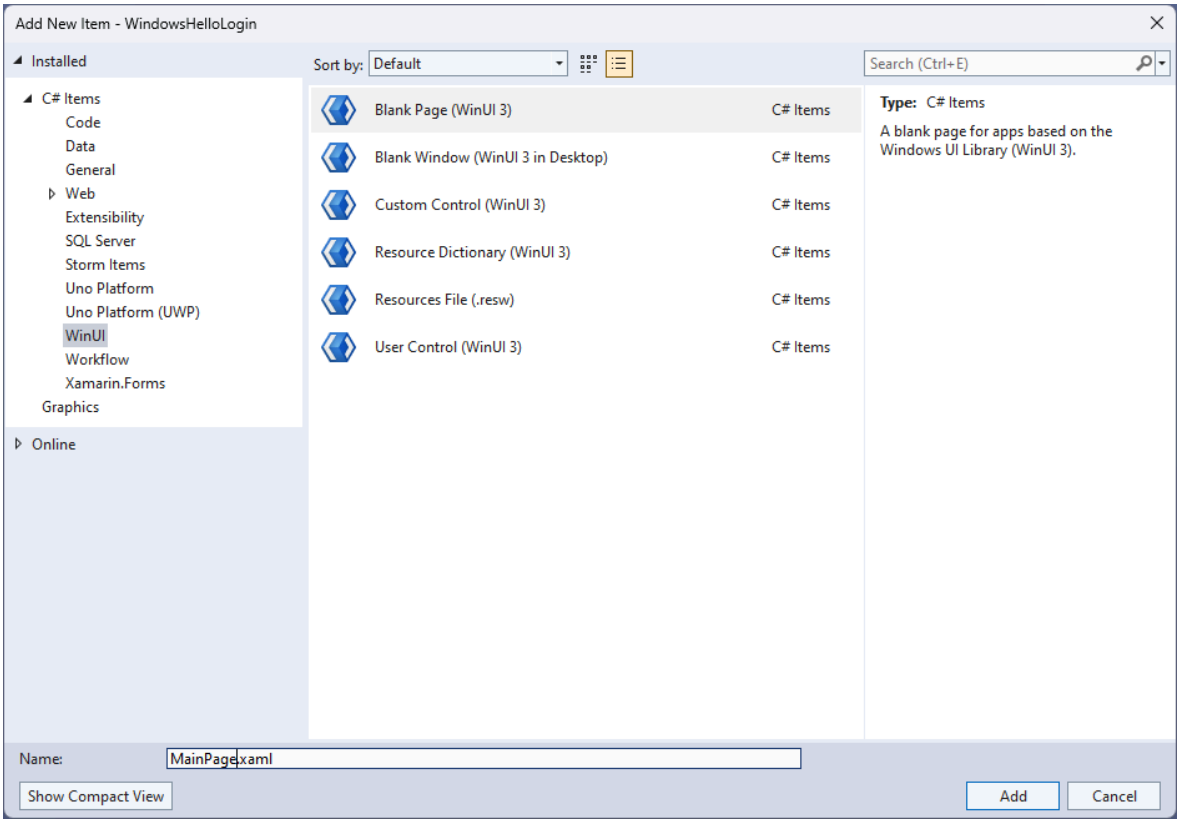


- Open MainWindow.xaml and replace the `Window` contents with an empty `StackPanel` or `Grid` control. We will be implementing page navigation and navigating to a new page when the **MainWindow** is loaded, so we don't need any content in the **MainWindow**.

- Add a `Title` property to **MainWindow** in the XAML. The attribute should look like this: `Title="Windows Hello Login"`.

- Remove the myButton_Click event handler from MainWindow.xaml.cs to avoid any compilation errors. This event handler is not needed for this sample.

- Right-click the new **Views** folder, select **Add** > **New Item** and select the **Blank Page** template. Name this page "MainPage.xaml".



- Open the App.xaml.cs file and update the **OnLaunched** handler to implement page navigation for the app. You'll also need to add a **RootFrame_NavigationFailed** handler method to deal with any errors that occur while loading pages.

```cs
protected override void
OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
{
    m_window = new MainWindow();
    var rootFrame = new Frame();
    rootFrame.NavigationFailed += RootFrame_NavigationFailed;
    rootFrame.Navigate(typeof(MainPage), args);
    m_window.Content = rootFrame;
    m_window.Activate();
}

private void RootFrame_NavigationFailed(object sender,
NavigationFailedEventArgs e)
{
    throw new Exception($"Error loading page
{e.SourcePageType.FullName}");
}
```

- You'll also need to add four using statements to the top of the App.xaml.cs file to resolve compilation errors in the code.

cs

```
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml.Navigation;
using System;
using WindowsHelloLogin.Views;
```

- Right-click the new **Views** folder, select **Add** > **New Item** and select the **Blank Page** template. Name this page "Login.xaml".
- To define the user interface for the new login page, add the following XAML. This XAML defines a `StackPanel` to align the following children:
  - A `TextBlock` that will contain a title.
  - A `TextBlock` for error messages.
  - A `TextBox` for the username to be input.
  - A `Button` to navigate to a register page.
  - A `TextBlock` to contain the status of Windows Hello.
  - A `TextBlock` to explain the Login page, as there is no backend or configured users yet.

XML

```
<Grid>
  <StackPanel>
    <TextBlock Text="Login" FontSize="36" Margin="4"
TextAlignment="Center"/>
    <TextBlock x:Name="ErrorMessage" Text="" FontSize="20"
Margin="4" Foreground="Red" TextAlignment="Center"/>
    <TextBlock Text="Enter your username below" Margin="0,0,0,20"
               TextWrapping="Wrap" Width="300"
               TextAlignment="Center" VerticalAlignment="Center"
FontSize="16"/>
    <TextBox x:Name="UsernameTextBox" Margin="4" Width="250"/>
    <Button x:Name="LoginButton" Content="Login"
Background="DodgerBlue" Foreground="White"
            Click="LoginButton_Click" Width="80"
HorizontalAlignment="Center" Margin="0,20"/>
    <TextBlock Text="Don't have an account?"
               TextAlignment="Center" VerticalAlignment="Center"
FontSize="16"/>
    <TextBlock x:Name="RegisterButtonTextBlock" Text="Register now"

PointerPressed="RegisterButtonTextBlock_OnPointerPressed"
               Foreground="DodgerBlue"
```

```
                TextAlignment="Center" VerticalAlignment="Center"
  FontSize="16"/>
      <Border x:Name="WindowsHelloStatus" Background="#22B14C"
              Margin="0,20" Height="100" >
        <TextBlock x:Name="WindowsHelloStatusText" Text="Windows Hello
  is ready to use!"
                   Margin="4" TextAlignment="Center"
  VerticalAlignment="Center" FontSize="20"/>
      </Border>
      <TextBlock x:Name="LoginExplanation" FontSize="24"
  TextAlignment="Center" TextWrapping="Wrap"
                 Text="Please Note: To demonstrate a login, validation
  will only occur using the default username 'sampleUsername'"/>
    </StackPanel>
  </Grid>
```

- A few methods need to be added to the code-behind file to get the solution building. Either press F7 or use the **Solution Explorer** to edit the Login.xaml.cs file. Add in the following two event methods to handle the **Login** and **Register** events. For now these methods will set the `ErrorMessage.Text` to an empty string. Be sure to include the following using statements. They will be needed for the next steps.

cs

```
using Microsoft.UI.Xaml;
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml.Input;
using Microsoft.UI.Xaml.Media;
using Microsoft.UI.Xaml.Navigation;
using System;

namespace WindowsHelloLogin.Views
{
    public sealed partial class Login : Page
    {
        public Login()
        {
            this.InitializeComponent();
        }
        private void LoginButton_Click(object sender, RoutedEventArgs
e)
        {
            ErrorMessage.Text = "";
        }
        private void RegisterButtonTextBlock_OnPointerPressed(object
sender, PointerRoutedEventArgs e)
        {
            ErrorMessage.Text = "";
        }
    }
}
```

- In order to render the **Login** page, edit the **MainPage** code to navigate to the **Login** page when the **MainPage** is loaded. Open the MainPage.xaml.cs file. In **Solution Explorer**, double-click on MainPage.xaml.cs. If you can't find this click the little arrow next to MainPage.xaml to show the code-behind file. Create a **Loaded** event handler method that will navigate to the **Login** page.

```cs
namespace WindowsHelloLogin.Views
{
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            Loaded += MainPage_Loaded;
        }

        private void MainPage_Loaded(object sender, RoutedEventArgs e)
        {
            Frame.Navigate(typeof(Login));
        }
    }
}
```

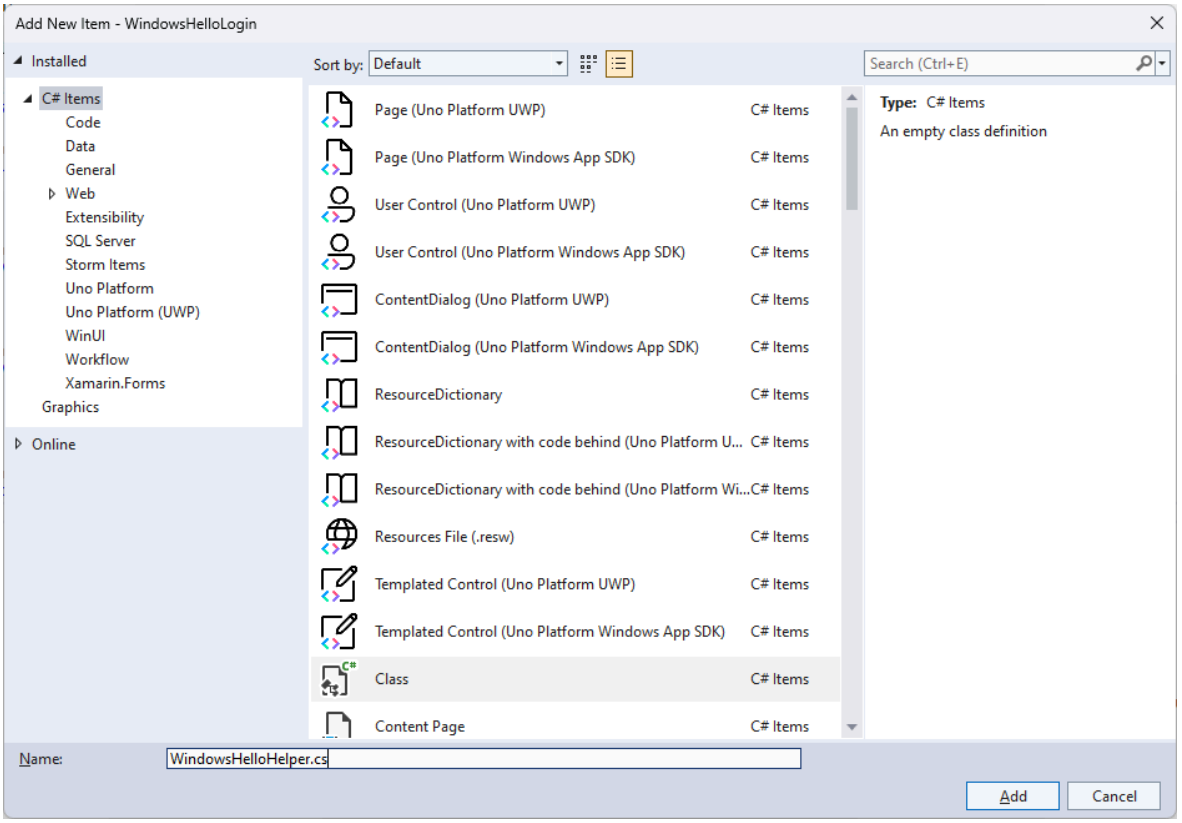
- In the **Login** page, you need to handle the `OnNavigatedTo` event to validate if Windows Hello is available on the current machine. In Login.xaml.cs, implement the following code. You'll notice that the WindowsHelloHelper object indicates that there's an error. That's because we haven't created this helper class yet.

```cs
public sealed partial class Login : Page
{
    public Login()
    {
        this.InitializeComponent();
    }

    protected override async void OnNavigatedTo(NavigationEventArgs e)
    {
        // Check if Windows Hello is set up and available on this
machine
        if (await WindowsHelloHelper.WindowsHelloAvailableCheckAsync())
        {
        }
        else
        {
            // Windows Hello isn't set up, so inform the user
            WindowsHelloStatus.Background = new
```

```
SolidColorBrush(Windows.UI.Color.FromArgb(255, 50, 170, 207));
            WindowsHelloStatusText.Text = $"Windows Hello is not set
up!{Environment.NewLine}Please go to Windows Settings and set up a PIN
to use it.";
            LoginButton.IsEnabled = false;
        }
    }
}
```

- To create the WindowsHelloHelper class, right-click the **WindowsHelloLogin** project and click **Add** > **New Folder**. Name this folder **Utils**.

- Right click the **Utils** folder and select **Add** > **Class**. Name this new class "WindowsHelloHelper.cs".



- Change the scope of the **WindowsHelloHelper** class to be `public static`, then add the following method that to inform the user if Windows Hello is ready to be used or not. You will need to add the required namespaces.

cs

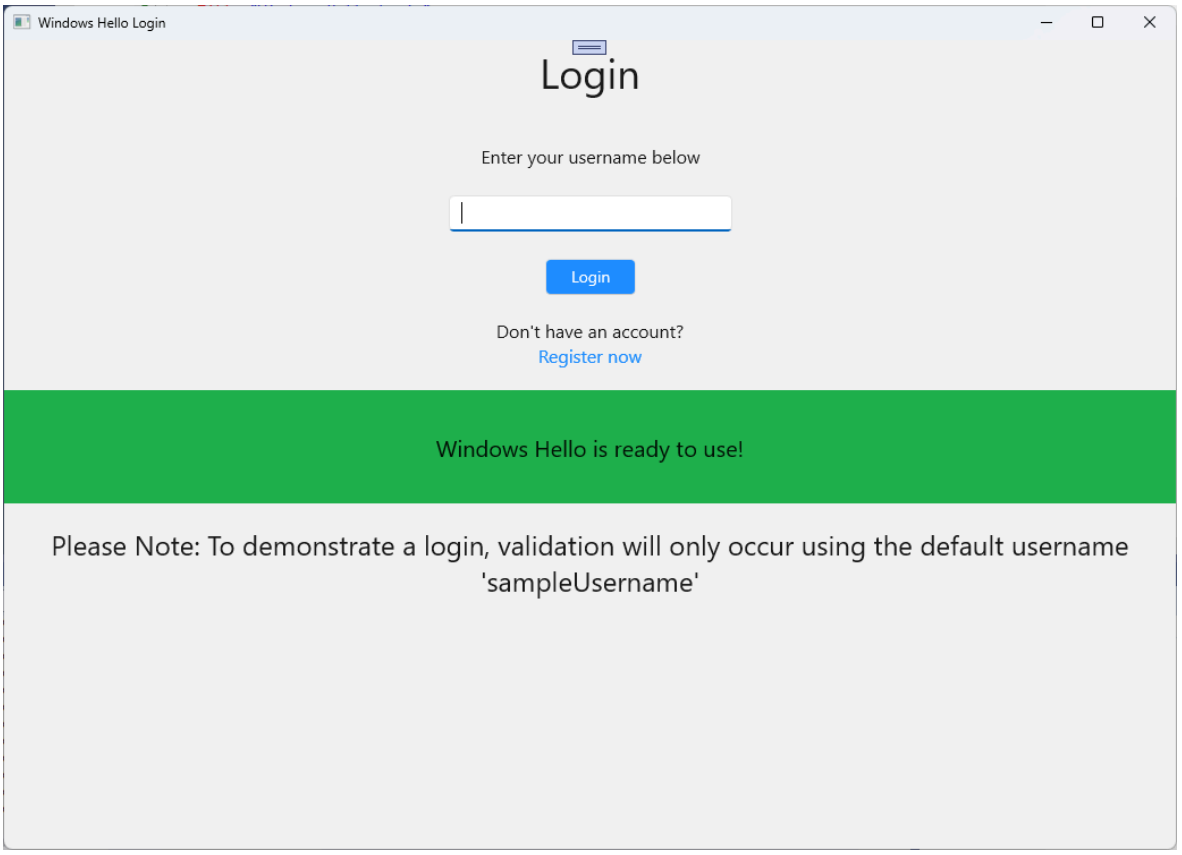
```
using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.Security.Credentials;

namespace WindowsHelloLogin.Utils
{
    public static class WindowsHelloHelper
```

```
    {
        /// <summary>
        /// Checks to see if Windows Hello is ready to be used.
        ///
        /// Windows Hello has dependencies on:
        ///     1. Having a connected Microsoft Account
        ///     2. Having a Windows PIN set up for that account on the
local machine
        /// </summary>
        public static async Task<bool>
WindowsHelloAvailableCheckAsync()
        {
            bool keyCredentialAvailable = await
KeyCredentialManager.IsSupportedAsync();
            if (keyCredentialAvailable == false)
            {
                // Key credential is not enabled yet as user
                // needs to connect to a Microsoft Account and select a
PIN in the connecting flow.
                Debug.WriteLine("Windows Hello is not set up!\nPlease
go to Windows Settings and set up a PIN to use it.");
                return false;
            }

            return true;
        }
    }
}
```

- In Login.xaml.cs, add a reference to the `WindowsHelloLogin.Utils` namespace. This will resolve the error in the `OnNavigatedTo` method.

```cs
using WindowsHelloLogin.Utils;
```

- Build and run the application. You will be navigated to the login page and the Windows Hello banner will indicate to you if Windows Hello is ready to be used. You should see either the green or blue banner indicating the Windows Hello status on your machine.



- The next thing you need to do is build the logic for signing in. Create a new folder in the project named "Models".
- In the **Models** folder, create a new class called "Account.cs". This class will act as your account model. As this is a sample project, it will only contain a username. Change the class scope to `public` and add the `Username` property.

cs

```cs
namespace WindowsHelloLogin.Models
{
    public class Account
    {
        public string Username { get; set; }
    }
}
```

- The app needs a way to handle accounts. For this hands on lab, as there is no server or database, a list of users are saved and loaded locally. Right-click the **Utils** folder and add a new class named "AccountHelper.cs". Change the class scope to be `public static`. The **AccountHelper** is a static class that contains all the necessary methods to save and load the list of accounts locally. Saving and loading works by using an **XmlSerializer**. You also need to remember the file that was saved and where you saved it.

cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;
using Windows.Storage;
using WindowsHelloLogin.Models;

namespace WindowsHelloLogin.Utils
{
    public static class AccountHelper
    {
        // In the real world this would not be needed as there would be
a server implemented that would host a user account database.
        // For this tutorial we will just be storing accounts locally.
        private const string USER_ACCOUNT_LIST_FILE_NAME =
"accountlist.txt";
        private static string _accountListPath =
Path.Combine(ApplicationData.Current.LocalFolder.Path,
USER_ACCOUNT_LIST_FILE_NAME);
        public static List<Account> AccountList = [];

        /// <summary>
        /// Create and save a useraccount list file. (Updating the old
one)
        /// </summary>
        private static async void SaveAccountListAsync()
        {
            string accountsXml = SerializeAccountListToXml();

            if (File.Exists(_accountListPath))
            {
                StorageFile accountsFile = await
StorageFile.GetFileFromPathAsync(_accountListPath);
                await FileIO.WriteTextAsync(accountsFile, accountsXml);
            }
            else
            {
                StorageFile accountsFile = await
ApplicationData.Current.LocalFolder.CreateFileAsync(USER_ACCOUNT_LIST_F
ILE_NAME);
                await FileIO.WriteTextAsync(accountsFile, accountsXml);
            }
        }

        /// <summary>
        /// Gets the useraccount list file and deserializes it from XML
to a list of useraccount objects.
        /// </summary>
        /// <returns>List of useraccount objects</returns>
        public static async Task<List<Account>> LoadAccountListAsync()
        {
            if (File.Exists(_accountListPath))
```

```
            {
                StorageFile accountsFile = await
StorageFile.GetFileFromPathAsync(_accountListPath);

                string accountsXml = await
FileIO.ReadTextAsync(accountsFile);
                DeserializeXmlToAccountList(accountsXml);
            }

            return AccountList;
        }

        /// <summary>
        /// Uses the local list of accounts and returns an XML
formatted string representing the list
        /// </summary>
        /// <returns>XML formatted list of accounts</returns>
        public static string SerializeAccountListToXml()
        {
            var xmlizer = new XmlSerializer(typeof(List<Account>));
            var writer = new StringWriter();
            xmlizer.Serialize(writer, AccountList);

            return writer.ToString();
        }

        /// <summary>
        /// Takes an XML formatted string representing a list of
accounts and returns a list object of accounts
        /// </summary>
        /// <param name="listAsXml">XML formatted list of
accounts</param>
        /// <returns>List object of accounts</returns>
        public static List<Account> DeserializeXmlToAccountList(string
listAsXml)
        {
            var xmlizer = new XmlSerializer(typeof(List<Account>));
            TextReader textreader = new StreamReader(new
MemoryStream(Encoding.UTF8.GetBytes(listAsXml)));

            return AccountList = (xmlizer.Deserialize(textreader)) as
List<Account>;
        }
    }
}
```

- Next, implement a way to add and remove an account from the local list of accounts. These actions will each save the list. The final method that you'll need for this hands-on lab is a validation method. As there is no authorization server or database of users, this will validate against a single user which is hard-coded. These methods should be added to the **AccountHelper** class.

cs

```cs
public static Account AddAccount(string username)
{
    // Create a new account with the username
    var account = new Account() { Username = username };
    // Add it to the local list of accounts
    AccountList.Add(account);
    // SaveAccountList and return the account
    SaveAccountListAsync();
    return account;
}

public static void RemoveAccount(Account account)
{
    // Remove the account from the accounts list
    AccountList.Remove(account);
    // Re save the updated list
    SaveAccountListAsync();
}

public static bool ValidateAccountCredentials(string username)
{
    // In the real world, this method would call the server to
authenticate that the account exists and is valid.
    // However, for this tutorial, we'll just have an existing sample
user that's named "sampleUsername".
    // If the username is null or does not match "sampleUsername"
validation will fail.
    // In this case, the user should register a new Windows Hello user.

    if (string.IsNullOrEmpty(username))
    {
        return false;
    }

    if (!string.Equals(username, "sampleUsername"))
    {
        return false;
    }

    return true;
}
```

- The next thing you need to do is handle a sign in request from the user. In Login.xaml.cs, create a new private variable that will hold the current account logging in. Then add a new method named **SignInWindowsHelloAsync**. This will validate the account credentials using the **AccountHelper.ValidateAccountCredentials** method. This method will return a **Boolean** value if the entered user name is the same as the hard coded string value

you configured in the previous step. The hard-coded value for this sample is "sampleUsername".

cs

```
using WindowsHelloLogin.Models;
using WindowsHelloLogin.Utils;
using System.Diagnostics;
using System.Threading.Tasks;

namespace WindowsHelloLogin.Views
{
    public sealed partial class Login : Page
    {
        private Account _account;

        public Login()
        {
            this.InitializeComponent();
        }

        protected override async void OnNavigatedTo(NavigationEventArgs
e)
        {
            // Check if Windows Hello is set up and available on this
machine
            if (await
WindowsHelloHelper.WindowsHelloAvailableCheckAsync())
            {
            }
            else
            {
                // Windows Hello is not set up, so inform the user
                WindowsHelloStatus.Background = new
SolidColorBrush(Windows.UI.Color.FromArgb(255, 50, 170, 207));
                WindowsHelloStatusText.Text = "Windows Hello is not set
up!\nPlease go to Windows Settings and set up a PIN to use it.";
                LoginButton.IsEnabled = false;
            }
        }

        private async void LoginButton_Click(object sender,
RoutedEventArgs e)
        {
            ErrorMessage.Text = "";
            await SignInWindowsHelloAsync();
        }

        private void RegisterButtonTextBlock_OnPointerPressed(object
sender, PointerRoutedEventArgs e)
        {
            ErrorMessage.Text = "";
        }
```

```
        private async Task SignInWindowsHelloAsync()
        {
            if
(AccountHelper.ValidateAccountCredentials(UsernameTextBox.Text))
            {
                // Create and add a new local account
                _account =
AccountHelper.AddAccount(UsernameTextBox.Text);
                Debug.WriteLine("Successfully signed in with
traditional credentials and created local account instance!");

                //if (await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(UsernameTextBox.Text))
                //{
                //    Debug.WriteLine("Successfully signed in with
Windows Hello!");
                //}
            }
            else
            {
                ErrorMessage.Text = "Invalid Credentials";
            }
        }
    }
}
```

- You may have noticed the commented code that was referencing a method in **WindowsHelloHelper**. In WindowsHelloHelper.cs, add a new method named **CreateWindowsHelloKeyAsync**. This method uses the Windows Hello API in the KeyCredentialManager. Calling RequestCreateAsync will create a Windows Hello key that is specific to the *accountId* and the local machine. Please note the comments in the switch statement if you are interested in implementing this in a real world scenario.

cs

```
/// <summary>
/// Creates a Windows Hello key on the machine using the account ID
provided.
/// </summary>
/// <param name="accountId">The account ID associated with the account
that we are enrolling into Windows Hello</param>
/// <returns>Boolean indicating if creating the Windows Hello key
succeeded</returns>
public static async Task<bool> CreateWindowsHelloKeyAsync(string
accountId)
{
    KeyCredentialRetrievalResult keyCreationResult = await
KeyCredentialManager.RequestCreateAsync(accountId,
KeyCredentialCreationOption.ReplaceExisting);
```

```
    switch (keyCreationResult.Status)
    {
        case KeyCredentialStatus.Success:
            Debug.WriteLine("Successfully created key");

            // In the real world, authentication would take place on a
server.
            // So, every time a user migrates or creates a new Windows
Hello
            // account, details should be pushed to the server.
            // The details that would be pushed to the server include:
            // The public key, keyAttestation (if available),
            // certificate chain for attestation endorsement key (if
available),
            // status code of key attestation result:
keyAttestationIncluded or
            // keyAttestationCanBeRetrievedLater and
keyAttestationRetryType.
            // As this sample has no concept of a server, it will be
skipped for now.
            // For information on how to do this, refer to the second
sample.

            // For this sample, just return true
            return true;
        case KeyCredentialStatus.UserCanceled:
            Debug.WriteLine("User cancelled sign-in process.");
            break;
        case KeyCredentialStatus.NotFound:
            // User needs to set up Windows Hello
            Debug.WriteLine("Windows Hello is not set up!\nPlease go to
Windows Settings and set up a PIN to use it.");
            break;
        default:
            break;
    }

    return false;
}
```

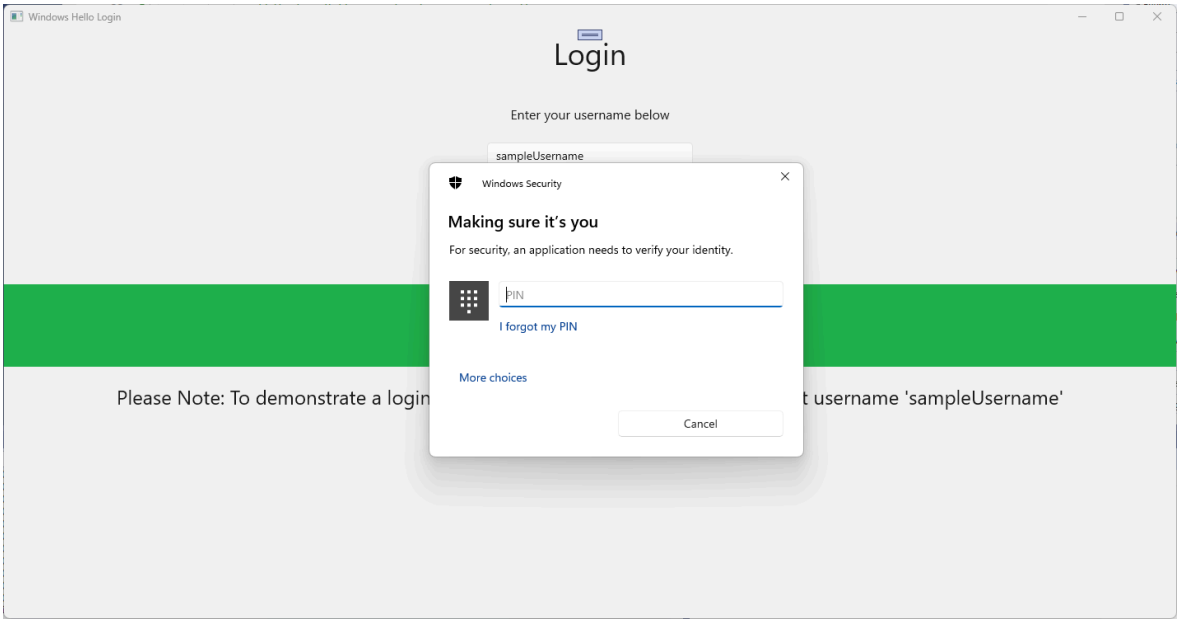
- Now that you have created the **CreateWindowsHelloKeyAsync** method, return to the Login.xaml.cs file and uncomment the code inside the SignInWindowsHelloAsync method.

cs

```
private async void SignInWindowsHelloAsync()
{
    if (AccountHelper.ValidateAccountCredentials(UsernameTextBox.Text))
    {
        //Create and add a new local account
        _account = AccountHelper.AddAccount(UsernameTextBox.Text);
        Debug.WriteLine("Successfully signed in with traditional
```

```
credentials and created local account instance!");

        if (await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(UsernameTextBox.Text))
        {
            Debug.WriteLine("Successfully signed in with Windows
Hello!");
        }
    }
    else
    {
        ErrorMessage.Text = "Invalid Credentials";
    }
}
```

- Build and run the application. You will be taken to the Login page. Enter the user name as "sampleUsername" and click login. You will be prompted with a Windows Hello prompt asking you to enter your PIN. Upon entering your PIN correctly, the **CreateWindowsHelloKeyAsync** method will be able to create a Windows Hello key. Monitor the output windows to see if the messages indicating success are shown.



Exercise 2: Welcome and User Selection Pages

In this exercise, you will continue from the previous exercise. When a user successfully logs in, they should be taken to a welcome page where they can sign out or delete their account. As Windows Hello creates a key for every machine, a user selection screen can be created, which displays all users that have been signed in on that machine. A user can then select one of these accounts and go directly to the welcome screen without needed to re-enter a password as they have already authenticated to access the machine.

- In the **Views** folder add a new blank page named "Welcome.xaml". Add the following XAML to complete the user interface for the page. This will display a title, the logged in username, and two buttons. One of the buttons will navigate back to a user list (that you will create later), and the other button will handle forgetting this user.

XML

```
<Grid>
  <StackPanel>
    <TextBlock x:Name="Title" Text="Welcome" FontSize="40"
TextAlignment="Center"/>
    <TextBlock x:Name="UserNameText" FontSize="28"
TextAlignment="Center"/>

    <Button x:Name="BackToUserListButton" Content="Back to User List"
Click="Button_Restart_Click"
            HorizontalAlignment="Center" Margin="0,20"
Foreground="White" Background="DodgerBlue"/>

    <Button x:Name="ForgetButton" Content="Forget Me"
Click="Button_Forget_User_Click"
            Foreground="White"
            Background="Gray"
            HorizontalAlignment="Center"/>
  </StackPanel>
</Grid>
```

- In the Welcome.xaml.cs code-behind file, add a new private variable that will hold the account that is logged in. You will need to implement a method to override the `OnNavigateTo` event, this will store the account passed to the **Welcome** page. You will also need to implement the `Click` event for the two buttons defined in the XAML. You will need to add using statements for the `WindowsHelloLogin.Models` and `WindowsHelloLogin.Utils` namespaces.

cs

```
using WindowsHelloLogin.Models;
using WindowsHelloLogin.Utils;
using System.Diagnostics;

namespace WindowsHelloLogin.Views
{
    public sealed partial class Welcome : Page
    {
        private Account _activeAccount;

        public Welcome()
        {
```

```
            InitializeComponent();
        }

        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            _activeAccount = (Account)e.Parameter;
            if (_activeAccount != null)
            {
                UserNameText.Text = _activeAccount.Username;
            }
        }

        private void Button_Restart_Click(object sender,
RoutedEventArgs e)
        {
        }

        private void Button_Forget_User_Click(object sender,
RoutedEventArgs e)
        {
            // Remove the account from Windows Hello
            //
WindowsHelloHelper.RemoveWindowsHelloAccountAsync(_activeAccount);

            // Remove it from the local accounts list and re-save the
updated list
            AccountHelper.RemoveAccount(_activeAccount);

            Debug.WriteLine($"User {_activeAccount.Username}
deleted.");
        }
    }
}
```

- You may have noticed a line commented out in the `Button_Forget_User_Click` event handler. The account is being removed from your local list but currently there is no way to be removed from Windows Hello. You need to implement a new method in WindowsHelloHelper.cs that will handle removing a Windows Hello user. This method will use other Windows Hello APIs to open and delete the account. In the real world, when you delete an account the server or database should be notified so the user database remains valid. You will need a using statement referencing the `WindowsHelloLogin.Models` namespace.

cs

```
using WindowsHelloLogin.Models;

/// <summary>
/// Function to be called when user requests deleting their account.
/// Checks the KeyCredentialManager to see if there is a Windows Hello
/// account for the current user.
```

```
/// It then deletes the local key associated with the account.
/// </summary>
public static async void RemoveWindowsHelloAccountAsync(Account
account)
{
    // Open the account with Windows Hello
    KeyCredentialRetrievalResult keyOpenResult = await
KeyCredentialManager.OpenAsync(account.Username);

    if (keyOpenResult.Status == KeyCredentialStatus.Success)
    {
        // In the real world you would send key information to server
to unregister
        //for example, RemoveWindowsHelloAccountOnServer(account);
    }

    // Then delete the account from the machine's list of Windows Hello
accounts
    await KeyCredentialManager.DeleteAsync(account.Username);
}
```

- Back in Welcome.xaml.cs, uncomment the line that calls **RemoveWindowsHelloAccountAsync**.

cs

```
private void Button_Forget_User_Click(object sender, RoutedEventArgs e)
{
    // Remove it from Windows Hello
    WindowsHelloHelper.RemoveWindowsHelloAccountAsync(_activeAccount);

    // Remove it from the local accounts list and re-save the updated
list
    AccountHelper.RemoveAccount(_activeAccount);

    Debug.WriteLine($"User {_activeAccount.Username} deleted.");
}
```

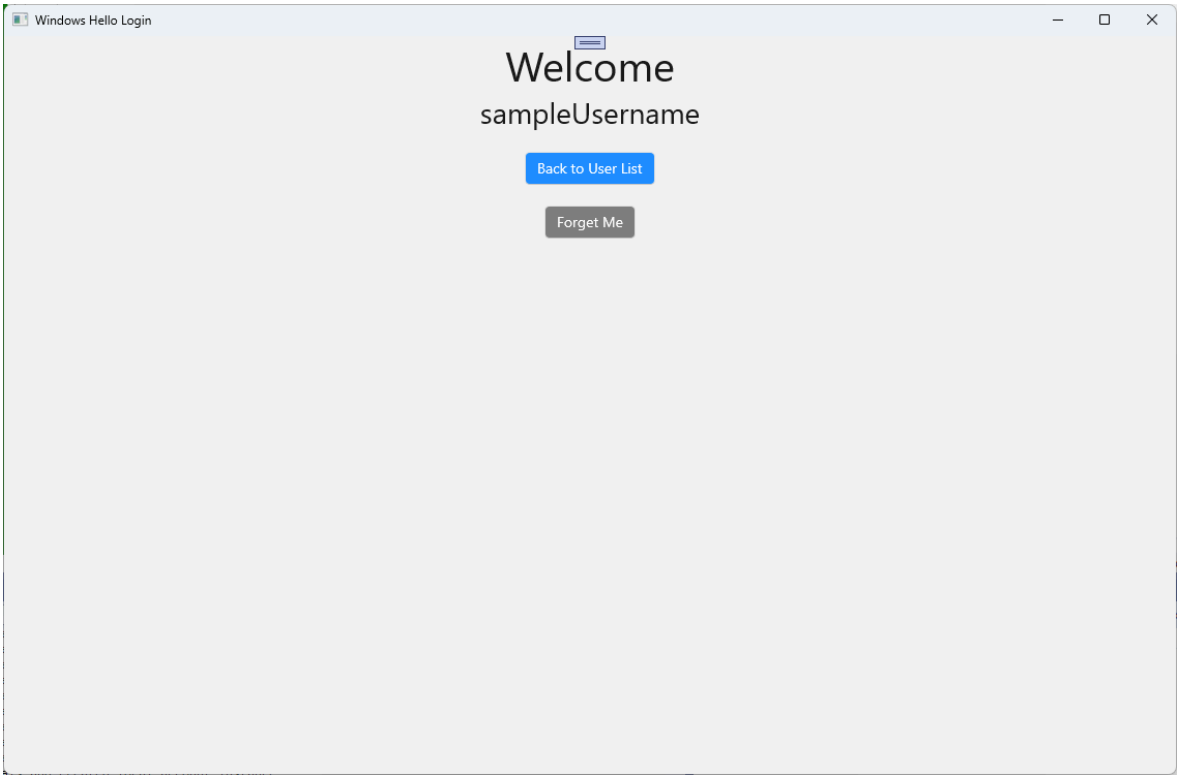
- In the **SignInWindowsHelloAsync** method (in Login.xaml.cs), once the **CreateWindowsHelloKeyAsync** is successful, it should navigate to the **Welcome** page and pass the **Account**.

cs

```
private async void SignInWindowsHelloAsync()
{
    if (AccountHelper.ValidateAccountCredentials(UsernameTextBox.Text))
    {
        // Create and add a new local account
        _account = AccountHelper.AddAccount(UsernameTextBox.Text);
        Debug.WriteLine("Successfully signed in with traditional
```

```
credentials and created local account instance!");

        if (await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(UsernameTextBox.Text))
        {
            Debug.WriteLine("Successfully signed in with Windows
Hello!");
            Frame.Navigate(typeof(Welcome), _account);
        }
    }
    else
    {
        ErrorMessage.Text = "Invalid Credentials";
    }
}
```

- Build and run the application. Login with "sampleUsername" and click **Login**. Enter your PIN and if successful you should be navigated to the **Welcome** screen. Try clicking **Forget User** and monitor Visual Studio's **Output** window to see if the user was deleted. Notice that when the user is deleted, you remain on the **Welcome** page. You'll need to create a user selection page to which the app can navigate.



- In the **Views** folder, create a new blank page named "UserSelection.xaml" and add the following XAML to define the user interface. This page will contain a ListView that displays all the users in the local accounts list, and a `Button` that will navigate to the **Login** page to allow the user to add another account.

```
XML
```

```
<Grid>
  <StackPanel>
    <TextBlock x:Name="Title" Text="Select a User" FontSize="36"
Margin="4" TextAlignment="Center" HorizontalAlignment="Center"/>

    <ListView x:Name="UserListView" Margin="4" MaxHeight="200"
MinWidth="250" Width="250" HorizontalAlignment="Center">
      <ListView.ItemTemplate>
        <DataTemplate>
          <Grid Background="DodgerBlue" Height="50" Width="250"
HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
            <TextBlock Text="{Binding Username}"
HorizontalAlignment="Center" TextAlignment="Center"
VerticalAlignment="Center" Foreground="White"/>
          </Grid>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>

    <Button x:Name="AddUserButton" Content="+" FontSize="36" Width="60"
Click="AddUserButton_Click" HorizontalAlignment="Center"/>
  </StackPanel>
</Grid>
```

- In UserSelection.xaml.cs, implement the `Loaded` method that will navigate to the **Login** page if there are no accounts in the local list. Also implement the `SelectionChanged` event for the `ListView` and a `Click` event for the `Button`.

cs

```
using System.Diagnostics;
using WindowsHelloLogin.Models;
using WindowsHelloLogin.Utils;

namespace WindowsHelloLogin.Views
{
    public sealed partial class UserSelection : Page
    {
        public UserSelection()
        {
            InitializeComponent();
            Loaded += UserSelection_Loaded;
        }

        private void UserSelection_Loaded(object sender,
RoutedEventArgs e)
        {
            if (AccountHelper.AccountList.Count == 0)
            {
                // If there are no accounts, navigate to the Login page
                Frame.Navigate(typeof(Login));
```

```
            }


            UserListView.ItemsSource = AccountHelper.AccountList;
            UserListView.SelectionChanged += UserSelectionChanged;
        }

        /// <summary>
        /// Function called when an account is selected in the list of
accounts
        /// Navigates to the Login page and passes the chosen account
        /// </summary>
        private void UserSelectionChanged(object sender,
RoutedEventArgs e)
        {
            if (((ListView)sender).SelectedValue != null)
            {
                Account account = (Account)
((ListView)sender).SelectedValue;
                if (account != null)
                {
                    Debug.WriteLine($"Account {account.Username}
selected!");
                }
                Frame.Navigate(typeof(Login), account);
            }
        }

        /// <summary>
        /// Function called when the "+" button is clicked to add a new
user.
        /// Navigates to the Login page with nothing filled out
        /// </summary>
        private void AddUserButton_Click(object sender, RoutedEventArgs
e)
        {
            Frame.Navigate(typeof(Login));
        }
    }
}
```

- There are a few places in the app where you want to navigate to the **UserSelection** page. In MainPage.xaml.cs, you should navigate to the **UserSelection** page instead of the **Login** page. While you are in the loaded event in **MainPage**, you will need to load the accounts list so the **UserSelection** page can check if there are any accounts. This will require changing the `Loaded` method to be async and also adding a using statement for the `WindowsHelloLogin.Utils` namespace.

```
cs
```

```
using WindowsHelloLogin.Utils;

private async void MainPage_Loaded(object sender, RoutedEventArgs e)
{
    // Load the local account list before navigating to the
UserSelection page
    await AccountHelper.LoadAccountListAsync();
    Frame.Navigate(typeof(UserSelection));
}
```

- Next, the app will need navigate to the **UserSelection** page from the **Welcome** page. In both `Click` events, you should navigate back to the **UserSelection** page.

cs

```
private void Button_Restart_Click(object sender, RoutedEventArgs e)
{
    Frame.Navigate(typeof(UserSelection));
}

private void Button_Forget_User_Click(object sender, RoutedEventArgs e)
{
    // Remove it from Windows Hello
    WindowsHelloHelper.RemoveWindowsHelloAccountAsync(_activeAccount);

    // Remove it from the local accounts list and re-save the updated
list
    AccountHelper.RemoveAccount(_activeAccount);

    Debug.WriteLine($"User {_activeAccount.Username} deleted.");

    // Navigate back to UserSelection page.
    Frame.Navigate(typeof(UserSelection));
}
```

- In the **Login** page, you need code to log in to the account selected from the list in the **UserSelection** page. In the `OnNavigatedTo` event, store the account passed during navigation. Start by adding a new private variable that will identify if the account is an existing account. Then handle the `OnNavigatedTo` event.

cs

```
namespace WindowsHelloLogin.Views
{
    public sealed partial class Login : Page
    {
        private Account _account;
        private bool _isExistingAccount;
```

```
        public Login()
        {
            InitializeComponent();
        }

        /// <summary>
        /// Function called when this frame is navigated to.
        /// Checks to see if Windows Hello is available and if an
account was passed in.
        /// If an account was passed in set the "_isExistingAccount"
flag to true and set the _account.
        /// </summary>
        protected override async void OnNavigatedTo(NavigationEventArgs
e)
        {
            // Check Windows Hello is set up and available on this
machine
            if (await
WindowsHelloHelper.WindowsHelloAvailableCheckAsync())
            {
                if (e.Parameter != null)
                {
                    _isExistingAccount = true;
                    // Set the account to the existing account being
passed in
                    _account = (Account)e.Parameter;
                    UsernameTextBox.Text = _account.Username;
                    await SignInWindowsHelloAsync();
                }
            }
            else
            {
                // Windows Hello is not set up, so inform the user
                WindowsHelloStatus.Background = new
SolidColorBrush(Windows.UI.Color.FromArgb(255, 50, 170, 207));
                WindowsHelloStatusText.Text = $"Windows Hello is not
set up!{Environment.NewLine}Please go to Windows Settings and set up a
PIN to use it.";
                LoginButton.IsEnabled = false;
            }
        }
    }
}
```

- The **SignInWindowsHelloAsync** method will need to be updated to sign in to the selected account. The **WindowsHelloHelper** will need another method to open the account with Windows Hello, as the account already has an account key created for it. Implement the new method in WindowsHelloHelper.cs to sign in an existing user with Windows Hello. For information on each part of the code, please read through the code comments.

```
cs
```

```
/// <summary>
/// Attempts to sign a message using the account key on the system for
the accountId passed.
/// </summary>
/// <returns>Boolean representing if creating the Windows Hello
authentication message succeeded</returns>
public static async Task<bool>
GetWindowsHelloAuthenticationMessageAsync(Account account)
{
    KeyCredentialRetrievalResult openKeyResult = await
KeyCredentialManager.OpenAsync(account.Username);
    // Calling OpenAsync will allow the user access to what is
available in the app and will not require user credentials again.
    // If you wanted to force the user to sign in again you can use the
following:
    // var consentResult = await
Windows.Security.Credentials.UI.UserConsentVerifier.RequestVerification
Async(account.Username);
    // This will ask for the either the password of the currently
signed in Microsoft Account or the PIN used for Windows Hello.

    if (openKeyResult.Status == KeyCredentialStatus.Success)
    {
        // If OpenAsync has succeeded, the next thing to think about is
whether the client application requires access to backend services.
        // If it does here you would request a challenge from the
server. The client would sign this challenge and the server
        // would check the signed challenge. If it is correct, it would
allow the user access to the backend.
        // You would likely make a new method called RequestSignAsync
to handle all this.
        // For example, RequestSignAsync(openKeyResult);
        // Refer to the second Windows Hello sample for information on
how to do this.

        // For this sample, there is not concept of a server
implemented so just return true.
        return true;
    }
    else if (openKeyResult.Status == KeyCredentialStatus.NotFound)
    {
        // If the account is not found at this stage. It could be one
of two errors.
        // 1. Windows Hello has been disabled
        // 2. Windows Hello has been disabled and re-enabled cause the
Windows Hello Key to change.
        // Calling CreateWindowsHelloKeyAsync and passing through the
account will attempt to replace the existing Windows Hello Key for that
account.
        // If the error really is that Windows Hello is disabled then
the CreateWindowsHelloKeyAsync method will output that error.
        if (await CreateWindowsHelloKeyAsync(account.Username))
        {
            // If the Hello Key was again successfully created, Windows
```

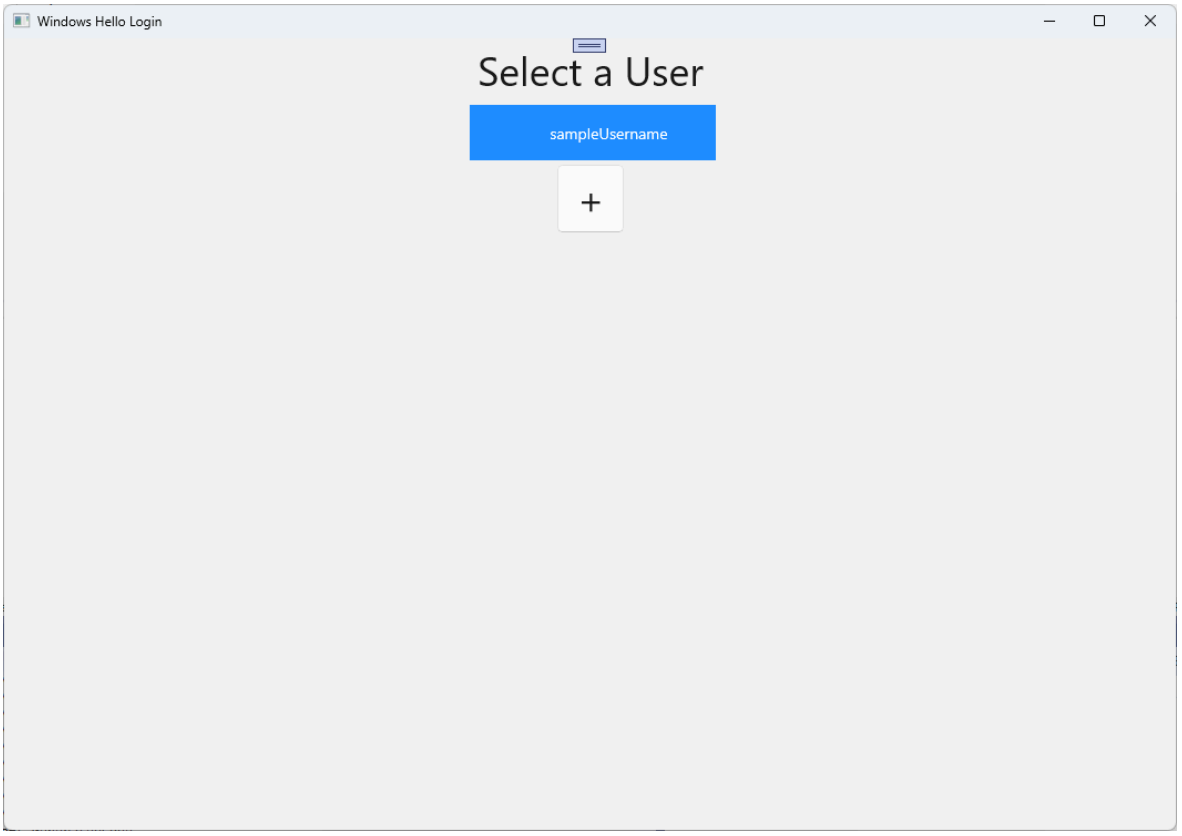
```
Hello has just been reset.
            // Now that the Hello Key has been reset for the account
retry sign in.
            return await
GetWindowsHelloAuthenticationMessageAsync(account);
        }
    }

    // Can't use Windows Hello right now, try again later
    return false;
}
```

- Update the **SignInWindowsHelloAsync** method in Login.xaml.cs to handle the existing account. This will use the new method in the WindowsHelloHelper.cs. If successful the account will be signed in and the user navigated to the **Welcome** page.

cs

```
private async Task SignInWindowsHelloAsync()
{
    if (_isExistingAccount)
    {
        if (await
WindowsHelloHelper.GetWindowsHelloAuthenticationMessageAsync(_account))
        {
            Frame.Navigate(typeof(Welcome), _account);
        }
    }
    else if
(AccountHelper.ValidateAccountCredentials(UsernameTextBox.Text))
    {
        //Create and add a new local account
        _account = AccountHelper.AddAccount(UsernameTextBox.Text);
        Debug.WriteLine("Successfully signed in with traditional
credentials and created local account instance!");

        if (await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(UsernameTextBox.Text))
        {
            Debug.WriteLine("Successfully signed in with Windows
Hello!");
            Frame.Navigate(typeof(Welcome), _account);
        }
    }
    else
    {
        ErrorMessage.Text = "Invalid Credentials";
    }
}
```

- Build and run the application. Login with "sampleUsername". Type in your PIN and if successful you will be navigated to the **Welcome** page. Click back to user list. You should now see a user in the list. If you click on this, WindowsHello enables you to sign back in without having to re-enter any passwords etc.



Exercise 3: Registering a new Windows Hello user

In this exercise, you create a new page that can create a new account with Windows Hello. This works similarly to how the **Login** page works. The **Login** page is implemented for an existing user that is migrating to use Windows Hello. A **WindowsHelloRegister** page will create Windows Hello registration for a new user.

- In the **Views** folder, create a new blank page named "WindowsHelloRegister.xaml". In the XAML add in the following to setup the user interface. The interface on this page is similar to the **Login** page.

XML

```
<Grid>
  <StackPanel>
    <TextBlock x:Name="Title" Text="Register New Windows Hello User"
FontSize="24" Margin="4" TextAlignment="Center"/>

    <TextBlock x:Name="ErrorMessage" Text="" FontSize="20" Margin="4"
```

```
Foreground="Red" TextAlignment="Center"/>

    <TextBlock Text="Enter your new username below" Margin="0,0,0,20"
               TextWrapping="Wrap" Width="300"
               TextAlignment="Center" VerticalAlignment="Center"
FontSize="16"/>

    <TextBox x:Name="UsernameTextBox" Margin="4" Width="250"/>

    <Button x:Name="RegisterButton" Content="Register"
Background="DodgerBlue" Foreground="White"
            Click="RegisterButton_Click_Async" Width="80"
HorizontalAlignment="Center" Margin="0,20"/>

    <Border x:Name="WindowsHelloStatus" Background="#22B14C" Margin="4"
Height="100">
      <TextBlock x:Name="WindowsHelloStatusText" Text="Windows Hello is
ready to use!" FontSize="20"
                 Margin="4" TextAlignment="Center"
VerticalAlignment="Center"/>
    </Border>
  </StackPanel>
</Grid>
```

- In the WindowsHelloRegister.xaml.cs code-behind file, implement a private `Account` variable and a `Click` event for the register button. This will add a new local account and create a Windows Hello key.

cs

```
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml;
using WindowsHelloLogin.Models;
using WindowsHelloLogin.Utils;

namespace WindowsHelloLogin.Views
{
    public sealed partial class WindowsHelloRegister : Page
    {
        private Account _account;

        public WindowsHelloRegister()
        {
            InitializeComponent();
        }

        private async void RegisterButton_Click_Async(object sender,
RoutedEventArgs e)
        {
            ErrorMessage.Text = "";

            // In the real world, you would validate the entered
```

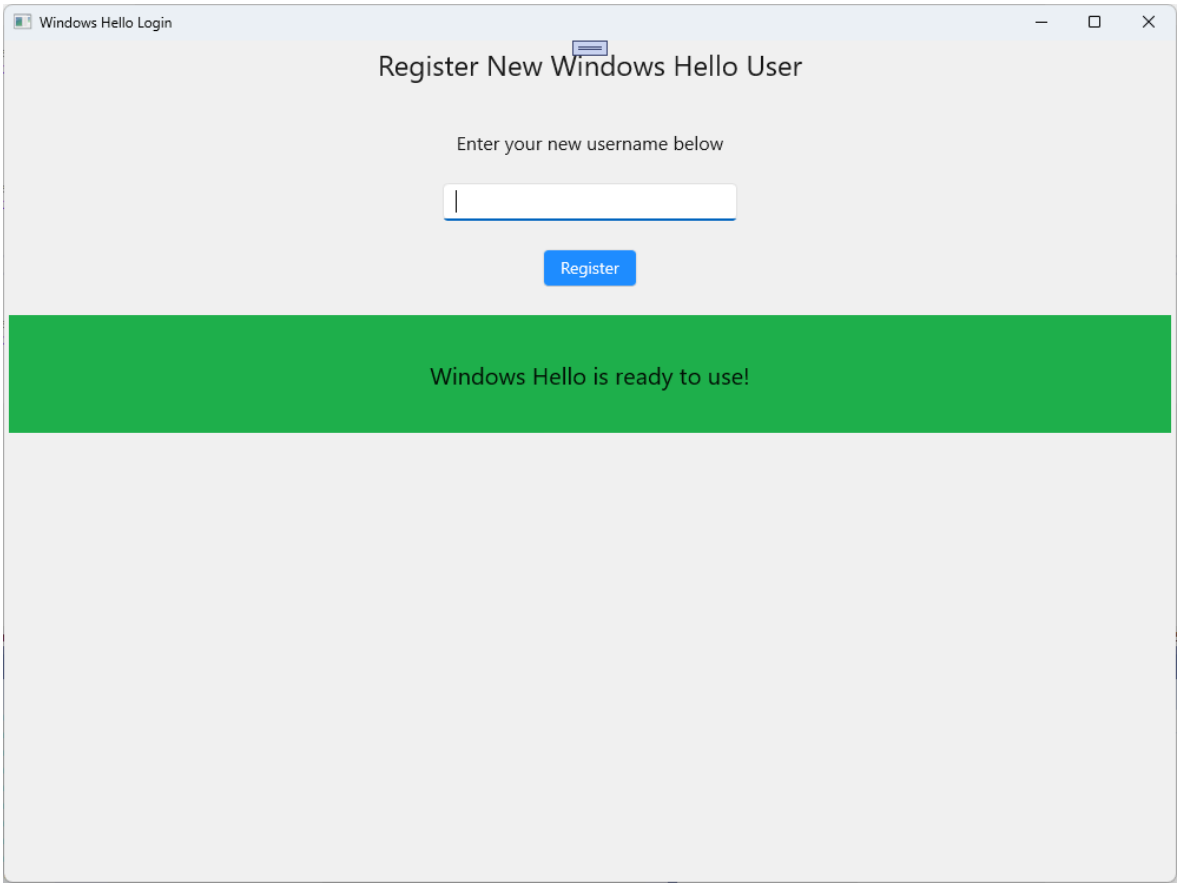
```
credentials and information before
            // allowing a user to register a new account.
            // For this sample, we'll skip that step and just register
an account if the username is not null.

            if (!string.IsNullOrEmpty(UsernameTextBox.Text))
            {
                // Register a new account
                _account =
AccountHelper.AddAccount(UsernameTextBox.Text);
                // Register new account with Windows Hello
                await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(_account.Username);
                // Navigate to the Welcome page.
                Frame.Navigate(typeof(Welcome), _account);
            }
            else
            {
                ErrorMessage.Text = "Please enter a username";
            }
        }
    }
}
```

- You need to navigate to this page from the **Login** page when register is clicked.

cs

```
private void RegisterButtonTextBlock_OnPointerPressed(object sender,
PointerRoutedEventArgs e)
{
    ErrorMessage.Text = "";
    Frame.Navigate(typeof(WindowsHelloRegister));
}
```

- Build and run the application. Try to register a new user. Then return to the user list and validate that you can select that user and login.



In this lab, you have learned the essential skills needed to use the new Windows Hello API to authenticate existing users and create accounts for new users. With this new knowledge, you can start removing the need for users to remember passwords for your application, yet remain confident that your applications remain protected by user authentication. Windows uses Windows Hello's new authentication technology to support its biometrics login options.

Related topics

- Windows Hello
- Windows Hello login service

Create a Windows Hello login service

Article • 07/11/2024

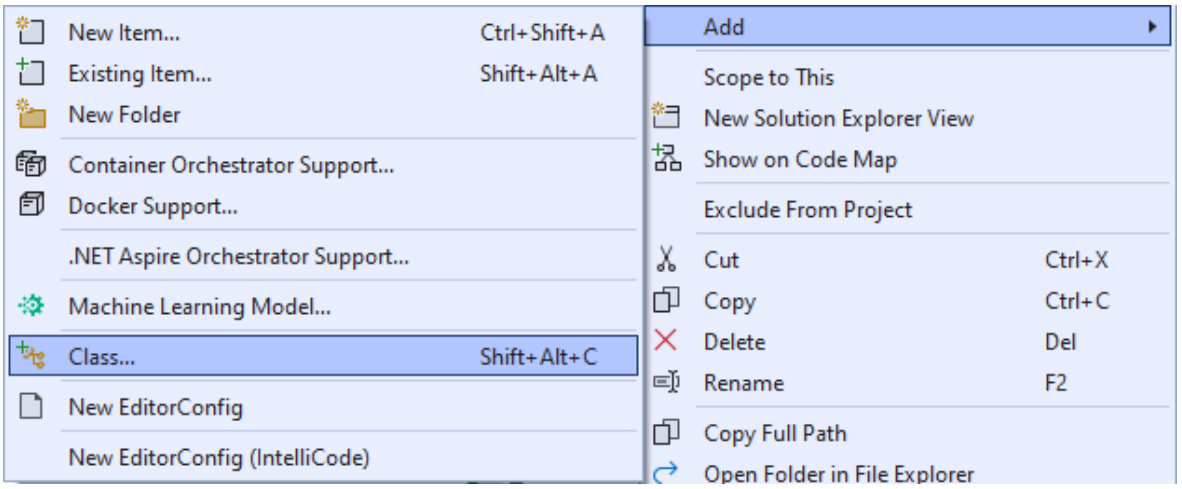
This is the second part of a complete walkthrough on how to use Windows Hello as an alternative to traditional username and password authentication systems in packaged Windows apps. This article picks up where Part 1, Windows Hello login app, left off and extends the functionality to demonstrate how you can integrate Windows Hello into your existing application.

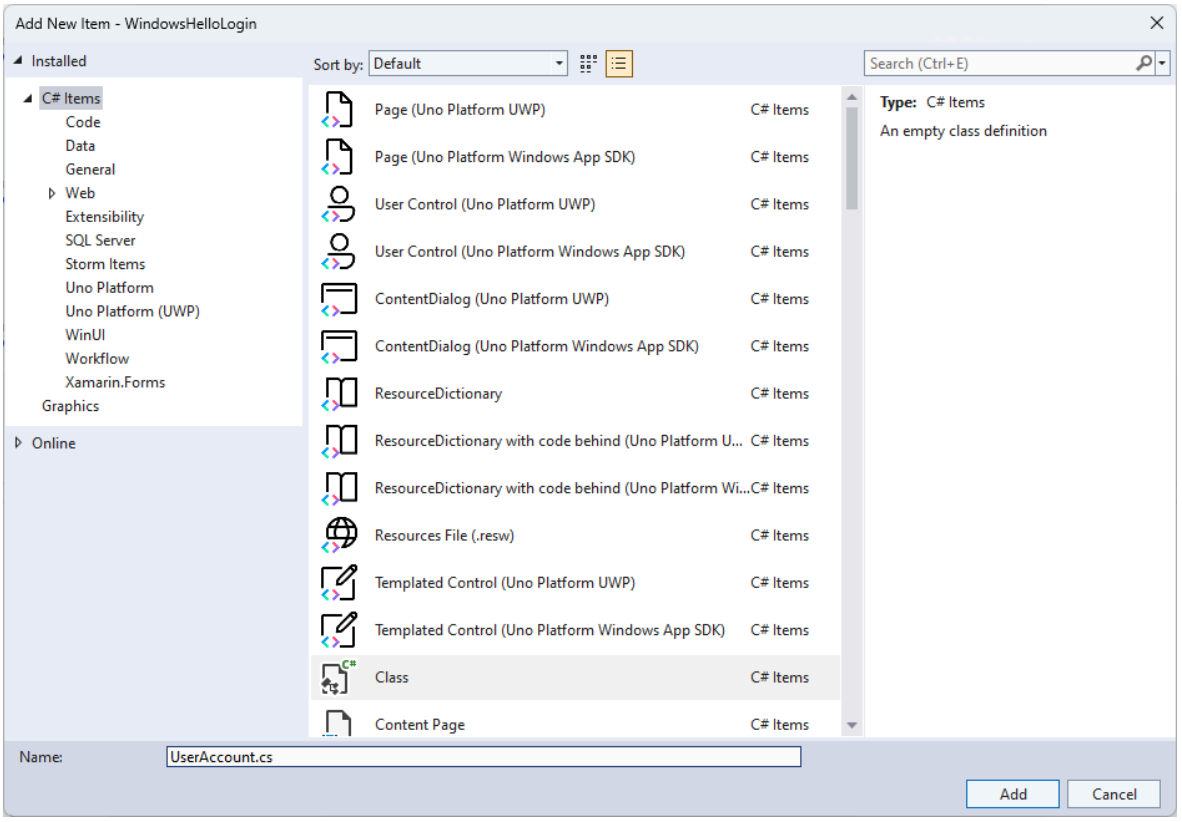
In order to build this project, you'll need some experience with C#, and XAML. You'll also need to be using Visual Studio 2022 on a Windows 10 or Windows 11 machine. See Start developing Windows apps for complete instructions on setting up your development environment.

Exercise 1: Server Side Logic

In this exercise, you start with the Windows Hello application built in the first lab and create a local mock server and database. This hands on lab is designed to teach how Windows Hello can be integrated into an existing system. By using a mock server and mock database, a lot of unrelated setup is eliminated. In your own applications, you will need to replace the mock objects with the real services and databases.

- To begin, open up the **WindowsHelloLogin** solution from the first Windows Hello Hands On Lab.
- You will start by implementing the mock server and mock database. Create a new folder named "AuthService". In Solution Explorer, right-click the **WindowsHelloLogin** project and select **Add** > **New Folder**.
- Create **UserAccount** and **WindowsHelloDevices** classes that will act as models for data to be saved in the mock database. The **UserAccount** will be similar to the user model implemented on a traditional authentication server. Right-click the **AuthService** folder and add a new class named "UserAccount".





- Change the class scope to be public and add the following public properties for the **UserAccount** class. You will need to add a using statement for the `System.ComponentModel.DataAnnotations` namespace.

cs

```cs
using System;
using System.ComponentModel.DataAnnotations;

namespace WindowsHelloLogin.AuthService
{
    public class UserAccount
    {
        [Key, Required]
        public Guid UserId { get; set; }
        [Required]
        public string Username { get; set; }
```

```
        public string Password { get; set; }
        // public List<WindowsHelloDevice> WindowsHelloDevices = new();
    }
}
```

You may have noticed the commented out list of **WindowsHelloDevices**. This is a modification you will need to make to an existing user model in your current implementation. The list of **WindowsHelloDevices** will contain a *deviceID*, the public key made from Windows Hello, and a KeyCredentialAttestationResult. For this exercise, you will need to implement the *keyAttestationResult* as they are only provided by Windows Hello on devices that have a TPM (Trusted Platform Modules) chip. The **KeyCredentialAttestationResult** is a combination of multiple properties and would need to be split in order to save and load them with a database.

- Create a new class in the **AuthService** folder called "WindowsHelloDevice.cs". This is the model for the Windows Hello devices as discussed above. Change the class scope to be public and add the following properties.

```cs
using System;

namespace WindowsHelloLogin.AuthService
{
    public class WindowsHelloDevice
    {
        // These are the new variables that will need to be added to
the existing UserAccount in the Database
        // The DeviceName is used to support multiple devices for the
one user.
        // This way the correct public key is easier to find as a new
public key is made for each device.
        // The KeyAttestationResult is only used if the User device has
a TPM (Trusted Platform Module) chip,
        // in most cases it will not. So will be left out for this
hands on lab.
        public Guid DeviceId { get; set; }
        public byte[] PublicKey { get; set; }
        // public KeyCredentialAttestationResult KeyAttestationResult {
get; set; }
    }
}
```

- Return to UserAccount.cs and uncomment the list of Windows Hello devices.

```cs
```

```
using System.Collections.Generic;

namespace WindowsHelloLogin.AuthService
{
    public class UserAccount
    {
        [Key, Required]
        public Guid UserId { get; set; }
        [Required]
        public string Username { get; set; }
        public string Password { get; set; }
        public List<WindowsHelloDevice> WindowsHelloDevices = new();
    }
}
```

- With the model for the **UserAccount** and the **WindowsHelloDevice** created, you need to create another new class in the **AuthService** folder that will act as the mock database, as this is a mock database from where you will be saving and loading a list of user accounts locally. In the real world, this would be your database implementation. Create a new class in the **AuthService** folder named "MockStore.cs". Change the class scope to public.

- As the mock store will save and load a list of user accounts locally, you can implement the logic to save and load that list using an **XmlSerializer**. You will also need to remember the filename and save location. In MockStore.cs implement the following:

cs

```
using System.Collections.Generic;
using System;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Serialization;
using Windows.Storage;

namespace WindowsHelloLogin.AuthService
{
    public class MockStore
    {
        private const string USER_ACCOUNT_LIST_FILE_NAME =
"userAccountsList.txt";
        // This cannot be a const because the LocalFolder is accessed
at runtime
        private string _userAccountListPath = Path.Combine(
            ApplicationData.Current.LocalFolder.Path,
USER_ACCOUNT_LIST_FILE_NAME);
```

```
        private List<UserAccount> _mockDatabaseUserAccountsList;

#region Save and Load Helpers
        /// <summary>
        /// Create and save a useraccount list file. (Replacing the old
one)
        /// </summary>
        private async Task SaveAccountListAsync()
        {
            string accountsXml = SerializeAccountListToXml();

            if (File.Exists(_userAccountListPath))
            {
                StorageFile accountsFile = await
StorageFile.GetFileFromPathAsync(_userAccountListPath);
                await FileIO.WriteTextAsync(accountsFile, accountsXml);
            }
            else
            {
                StorageFile accountsFile = await
ApplicationData.Current.LocalFolder.CreateFileAsync(USER_ACCOUNT_LIST_F
ILE_NAME);
                await FileIO.WriteTextAsync(accountsFile, accountsXml);
            }
        }

        /// <summary>
        /// Gets the useraccount list file and deserializes it from XML
to a list of useraccount objects.
        /// </summary>
        /// <returns>List of useraccount objects</returns>
        private async Task LoadAccountListAsync()
        {
            if (File.Exists(_userAccountListPath))
            {
                StorageFile accountsFile = await
StorageFile.GetFileFromPathAsync(_userAccountListPath);

                string accountsXml = await
FileIO.ReadTextAsync(accountsFile);
                DeserializeXmlToAccountList(accountsXml);
            }

            // If the UserAccountList does not contain the sampleUser
Initialize the sample users
            // This is only needed as it in a Hand on Lab to
demonstrate a user being migrated.
            // In the real world, user accounts would just be in a
database.
            if (!_mockDatabaseUserAccountsList.Any(f =>
f.Username.Equals("sampleUsername")))
            {
                //If the list is empty, call InitializeSampleAccounts
and return the list
                //await InitializeSampleUserAccountsAsync();
```

```
            }
        }

        /// <summary>
        /// Uses the local list of accounts and returns an XML
formatted string representing the list
        /// </summary>
        /// <returns>XML formatted list of accounts</returns>
        private string SerializeAccountListToXml()
        {
            var xmlizer = new XmlSerializer(typeof(List<UserAccount>));
            var writer = new StringWriter();
            xmlizer.Serialize(writer, _mockDatabaseUserAccountsList);
            return writer.ToString();
        }

        /// <summary>
        /// Takes an XML formatted string representing a list of
accounts and returns a list object of accounts
        /// </summary>
        /// <param name="listAsXml">XML formatted list of
accounts</param>
        /// <returns>List object of accounts</returns>
        private List<UserAccount> DeserializeXmlToAccountList(string
listAsXml)
        {
            var xmlizer = new XmlSerializer(typeof(List<UserAccount>));
            TextReader textreader = new StreamReader(new
MemoryStream(Encoding.UTF8.GetBytes(listAsXml)));
            return _mockDatabaseUserAccountsList =
(xmlizer.Deserialize(textreader)) as List<UserAccount>;
        }
#endregion
    }
}
```

- In the **LoadAccountListAsync** method, you may have noticed that an **InitializeSampleUserAccountsAsync** method was commented out. You will need to create this method in the MockStore.cs. This method will populate the user accounts list so that a login can take place. In the real world, the user database would already be populated. In this step, you will also be creating a constructor that will initialize the user list and call **LoadAccountListAsync**.

cs

```
namespace WindowsHelloLogin.AuthService
{
    public class MockStore
    {
        private const string USER_ACCOUNT_LIST_FILE_NAME =
"userAccountsList.txt";
        // This cannot be a const because the LocalFolder is accessed
```

```
at runtime
        private string _userAccountListPath = Path.Combine(
            ApplicationData.Current.LocalFolder.Path,
USER_ACCOUNT_LIST_FILE_NAME);
        private List<UserAccount> _mockDatabaseUserAccountsList;

        public MockStore()
        {
            _mockDatabaseUserAccountsList = new List<UserAccount>();
            _ = LoadAccountListAsync();
        }

        private async Task InitializeSampleUserAccountsAsync()
        {
            // Create a sample Traditional User Account that only has a
Username and Password
            // This will be used initially to demonstrate how to
migrate to use Windows Hello

            var sampleUserAccount = new UserAccount()
            {
                UserId = Guid.NewGuid(),
                Username = "sampleUsername",
                Password = "samplePassword",
            };

            // Add the sampleUserAccount to the _mockDatabase
            _mockDatabaseUserAccountsList.Add(sampleUserAccount);
            await SaveAccountListAsync();
        }
    }
}
```

- Now that the **InitializeSampleUserAccountsAsync** method exists uncomment the method call in the **LoadAccountListAsync** method.

cs

```
private async Task LoadAccountListAsync()
{
    if (File.Exists(_userAccountListPath))
    {
        StorageFile accountsFile = await
StorageFile.GetFileFromPathAsync(_userAccountListPath);

        string accountsXml = await FileIO.ReadTextAsync(accountsFile);
        DeserializeXmlToAccountList(accountsXml);
    }

    // If the UserAccountList does not contain the sampleUser
Initialize the sample users
    // This is only needed as it in a Hand on Lab to demonstrate a user
migrating
```

```
    // In the real world user accounts would just be in a database
    if (!_mockDatabaseUserAccountsList.Any(f = >
f.Username.Equals("sampleUsername")))
            {
        //If the list is empty InitializeSampleUserAccountsAsync and
return the list
        await InitializeSampleUserAccountsAsync();
    }
}
```

- The user accounts list in mock store can now be saved and loaded. Other parts of the application will need to have access to this list so there will need to be some methods to retrieve this data. Underneath the **InitializeSampleUserAccountsAsync** method, add the following methods to get data. They will allow you to get a user ID, a single user, a list of users for a specific Windows Hello device, and also get the public key for the user on a specific device.

cs

```
public Guid GetUserId(string username)
{
    if (_mockDatabaseUserAccountsList.Any())
    {
        UserAccount account =
_mockDatabaseUserAccountsList.FirstOrDefault(f =>
f.Username.Equals(username));
        if (account != null)
        {
            return account.UserId;
        }
    }
    return Guid.Empty;
}

public UserAccount GetUserAccount(Guid userId)
{
    return _mockDatabaseUserAccountsList.FirstOrDefault(f =>
f.UserId.Equals(userId));
}

public List<UserAccount> GetUserAccountsForDevice(Guid deviceId)
{
    var usersForDevice = new List<UserAccount>();

    foreach (UserAccount account in _mockDatabaseUserAccountsList)
    {
        if (account.WindowsHelloDevices.Any(f =>
f.DeviceId.Equals(deviceId)))
        {
            usersForDevice.Add(account);
        }
    }
```

```
    return usersForDevice;
}

public byte[] GetPublicKey(Guid userId, Guid deviceId)
{
    UserAccount account =
_mockDatabaseUserAccountsList.FirstOrDefault(f =>
f.UserId.Equals(userId));
    if (account != null)
    {
        if (account.WindowsHelloDevices.Any())
        {
            return account.WindowsHelloDevices.FirstOrDefault(p =>
p.DeviceId.Equals(deviceId)).PublicKey;
        }
    }
    return null;
}
```

- The next methods to implement will handle simple operations to add an account, remove an account, and also remove a device. Removing a device is needed as Windows Hello is device-specific. For each device to which you log in, a new public and private key pair will be created by Windows Hello. It's like having a different password for each device you sign in on, the only thing is you don't need to remember all those passwords; the server does. Add the following methods into the MockStore.cs.

cs

```
public async Task<UserAccount> AddAccountAsync(string username)
{
    UserAccount newAccount = null;
    try
    {
        newAccount = new UserAccount()
        {
            UserId = Guid.NewGuid(),
            Username = username,
        };

        _mockDatabaseUserAccountsList.Add(newAccount);
        await SaveAccountListAsync();
    }
    catch (Exception)
    {
        throw;
    }
    return newAccount;
}
```

```
public async Task<bool> RemoveAccountAsync(Guid userId)
{
    UserAccount userAccount = GetUserAccount(userId);
    if (userAccount != null)
    {
        _mockDatabaseUserAccountsList.Remove(userAccount);
        await SaveAccountListAsync();
        return true;
    }
    return false;
}

public async Task<bool> RemoveDeviceAsync(Guid userId, Guid deviceId)
{
    UserAccount userAccount = GetUserAccount(userId);
    WindowsHelloDevice deviceToRemove = null;
    if (userAccount != null)
    {
        foreach (WindowsHelloDevice device in
userAccount.WindowsHelloDevices)
        {
            if (device.DeviceId.Equals(deviceId))
            {
                deviceToRemove = device;
                break;
            }
        }
    }

    if (deviceToRemove != null)
    {
        //Remove the WindowsHelloDevice
        userAccount.WindowsHelloDevices.Remove(deviceToRemove);
        await SaveAccountListAsync();
    }

    return true;
}
```

- In the **MockStore** class add a method that will add Windows Hello related information to an existing **UserAccount**. This method will be called "WindowsHelloUpdateDetailsAsync" and will take parameters to identify the user, and the Windows Hello details. The **KeyAttestationResult** has been commented out when creating a WindowsHelloDevice, in a real world application you would require this.

```cs
using System.Threading.Tasks;
using Windows.Security.Credentials;

public async Task WindowsHelloUpdateDetailsAsync(Guid userId, Guid
```

```
deviceId, byte[] publicKey,
    KeyCredentialAttestationResult keyAttestationResult)
{
    UserAccount existingUserAccount = GetUserAccount(userId);
    if (existingUserAccount != null)
    {
        if (!existingUserAccount.WindowsHelloDevices.Any(f =>
f.DeviceId.Equals(deviceId)))
        {
            existingUserAccount.WindowsHelloDevices.Add(new
WindowsHelloDevice()
            {
                DeviceId = deviceId,
                PublicKey = publicKey,
                // KeyAttestationResult = keyAttestationResult
            });
        }
    }
    await SaveAccountListAsync();
}
```

- The **MockStore** class is now complete, as this represents the database it should be considered private. In order to access the **MockStore**, an **AuthService** class is needed to manipulate the database data. In the **AuthService** folder create a new class called "AuthService.cs". Change the class scope to public and add a singleton instance pattern to make sure only one instance is ever created.

cs

```
namespace WindowsHelloLogin.AuthService
{
    public class AuthService
    {
        // Singleton instance of the AuthService
        // The AuthService is a mock of what a real world server and
service implementation would be
        private static AuthService _instance;
        public static AuthService Instance
        {
            get
            {
                if (null == _instance)
                {
                    _instance = new AuthService();
                }
                return _instance;
            }
        }

        private AuthService()
        { }
```

```
    }
}
```

- The **AuthService** class needs to create an instance of the **MockStore** class and provide access to the properties of the **MockStore** object.

cs

```
namespace WindowsHelloLogin.AuthService
{
    public class AuthService
    {
        //Singleton instance of the AuthService
        //The AuthService is a mock of what a real world server and
database implementation would be
        private static AuthService _instance;
        public static AuthService Instance
        {
            get
            {
                if (null == _instance)
                {
                    _instance = new AuthService();
                }
                return _instance;
            }
        }

        private AuthService()
        { }

        private MockStore _mockStore = new();

        public Guid GetUserId(string username)
        {
            return _mockStore.GetUserId(username);
        }

        public UserAccount GetUserAccount(Guid userId)
        {
            return _mockStore.GetUserAccount(userId);
        }

        public List<UserAccount> GetUserAccountsForDevice(Guid
deviceId)
        {
            return _mockStore.GetUserAccountsForDevice(deviceId);
        }
    }
}
```

- You need methods in the **AuthService** class to access add, remove, and update Windows Hello details methods in the **MockStore** object. At the end of the **AuthService** class definition, add the following methods.

```cs
using System.Threading.Tasks;
using Windows.Security.Credentials;

public async Task RegisterAsync(string username)
{
    await _mockStore.AddAccountAsync(username);
}

public async Task<bool> WindowsHelloRemoveUserAsync(Guid userId)
{
    return await _mockStore.RemoveAccountAsync(userId);
}

public async Task<bool> WindowsHelloRemoveDeviceAsync(Guid userId, Guid
deviceId)
{
    return await _mockStore.RemoveDeviceAsync(userId, deviceId);
}

public async Task WindowsHelloUpdateDetailsAsync(Guid userId, Guid
deviceId, byte[] publicKey,
    KeyCredentialAttestationResult keyAttestationResult)
{
    await _mockStore.WindowsHelloUpdateDetailsAsync(userId, deviceId,
publicKey, keyAttestationResult);
}
```

- The **AuthService** class needs to provide a method to validate credentials. This method will take a username and password and make sure that account exists and the password is valid. An existing system would have an equivalent method to this that checks the user is authorized. Add the following **ValidateCredentials** method to the AuthService.cs file.

```cs
public bool ValidateCredentials(string username, string password)
{
    if (!string.IsNullOrEmpty(username) &&
!string.IsNullOrEmpty(password))
    {
        // This would be used for existing accounts migrating to use
Windows Hello
        Guid userId = GetUserId(username);
        if (userId != Guid.Empty)
```

```
        {
            UserAccount account = GetUserAccount(userId);
            if (account != null)
            {
                if (string.Equals(password, account.Password))
                {
                    return true;
                }
            }
        }
    }
    return false;
}
```

- The **AuthService** class needs a request challenge method that returns a challenge to the client to validate whether the user is who they claim to be. Then another method is needed in the **AuthService** class to receive the signed challenge back from the client. For this hands on lab, the method of how you determine if the signed challenge has been completed has been left incomplete. Every implementation of Windows Hello into an existing authentication system will be slightly different. The public key stored on the server needs to match with the result the client returned to the server. Add these two methods to AuthService.cs.

cs

```
using Windows.Security.Cryptography;
using Windows.Storage.Streams;

public IBuffer WindowsHelloRequestChallenge()
{
    return CryptographicBuffer.ConvertStringToBinary("ServerChallenge",
BinaryStringEncoding.Utf8);
}

public bool SendServerSignedChallenge(Guid userId, Guid deviceId,
byte[] signedChallenge)
{
    // Depending on your company polices and procedures this step will
be different
    // It is at this point you will need to validate the
signedChallenge that is sent back from the client.
    // Validation is used to ensure the correct user is trying to
access this account.
    // The validation process will use the signedChallenge and the
stored PublicKey
    // for the username and the specific device signin is called from.
    // Based on the validation result you will return a bool value to
allow access to continue or to block the account.

    // For this sample validation will not happen as a best practice
solution does not apply and will need to
```

```
        // be configured for each company.
    // Simply just return true.

    // You could get the User's Public Key with something similar to
the following:
    byte[] userPublicKey = _mockStore.GetPublicKey(userId, deviceId);
    return true;
}
```

Exercise 2: Client Side Logic

In this exercise, you will be changing the client side views and helper classes from the first lab to use the **AuthService** class. In the real world, the **AuthService** would be the authentication server and you would need to use Web API's to send and receive data from the server. For this hands on lab, the client and server are both local to keep things simple. The objective is to learn how to use the Windows Hello APIs.

- In the MainPage.xaml.cs, you can remove the **AccountHelper.LoadAccountListAsync** method call in the loaded method as the **AuthService** class creates an instance of the **MockStore** to load the accounts list. The `Loaded` method should now look like the snippet below. Note that the async method definition is removed as nothing is being awaited.

  cs

  ```
  private void MainPage_Loaded(object sender, RoutedEventArgs e)
  {
      Frame.Navigate(typeof(UserSelection));
  }
  ```

- Update the **Login** page interface to require a password be entered. This hands on lab demonstrates how an existing system could be migrated to use Windows Hello and existing accounts will have a username and a password. Also update the explanation at the bottom of the XAML to include the default password. Update the following XAML in Login.xaml.

  XML

  ```
  <Grid>
    <StackPanel>
      <TextBlock Text="Login" FontSize="36" Margin="4"
  TextAlignment="Center"/>

      <TextBlock x:Name="ErrorMessage" Text="" FontSize="20" Margin="4"
  Foreground="Red" TextAlignment="Center"/>
  ```

```
    <TextBlock Text="Enter your credentials below" Margin="0,0,0,20"
               TextWrapping="Wrap" Width="300"
               TextAlignment="Center" VerticalAlignment="Center"
FontSize="16"/>

    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
      <!-- Username Input -->
      <TextBlock x:Name="UserNameTextBlock" Text="Username: "
                 FontSize="20" Margin="4" Width="100"/>
      <TextBox x:Name="UsernameTextBox"
PlaceholderText="sampleUsername" Width="200" Margin="4"/>
    </StackPanel>

    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
      <!-- Password Input -->
      <TextBlock x:Name="PasswordTextBlock" Text="Password: "
                 FontSize="20" Margin="4" Width="100"/>
      <PasswordBox x:Name="PasswordBox"
PlaceholderText="samplePassword" Width="200" Margin="4"/>
    </StackPanel>

    <Button x:Name="LoginButton" Content="Login"
Background="DodgerBlue" Foreground="White"
            Click="LoginButton_Click" Width="80"
HorizontalAlignment="Center" Margin="0,20"/>

    <TextBlock Text="Don't have an account?"
               TextAlignment="Center" VerticalAlignment="Center"
FontSize="16"/>
    <TextBlock x:Name="RegisterButtonTextBlock" Text="Register now"

PointerPressed="RegisterButtonTextBlock_OnPointerPressed"
               Foreground="DodgerBlue"
               TextAlignment="Center" VerticalAlignment="Center"
FontSize="16"/>

    <Border x:Name="WindowsHelloStatus" Background="#22B14C"
            Margin="0,20" Height="100">
      <TextBlock x:Name="WindowsHelloStatusText" Text="Windows Hello is
ready to use!"
                 Margin="4" TextAlignment="Center"
VerticalAlignment="Center" FontSize="20"/>
    </Border>

    <TextBlock x:Name="LoginExplanation" FontSize="24"
TextAlignment="Center" TextWrapping="Wrap"
               Text="Please Note: To demonstrate a login, validation
will only occur using the default username 'sampleUsername' and default
password 'samplePassword'"/>
  </StackPanel>
</Grid>
```

- In the code-behind file for the **Login** class, you will need to change the `Account` private variable at the top of the class to be a `UserAccount`. Change the `OnNavigateTo` event to cast the type to be a `UserAccount`. You will need the following using statement as well.

cs

```cs
using WindowsHelloLogin.AuthService;

namespace WindowsHelloLogin.Views
{
    public sealed partial class Login : Page
    {
        private UserAccount _account;
        private bool _isExistingAccount;

        public Login()
        {
            this.InitializeComponent();
        }

        protected override async void OnNavigatedTo(NavigationEventArgs
e)
        {
            //Check Windows Hello is setup and available on this
machine
            if (await
WindowsHelloHelper.WindowsHelloAvailableCheckAsync())
            {
                if (e.Parameter != null)
                {
                    _isExistingAccount = true;
                    //Set the account to the existing account being
passed in
                    _account = (UserAccount)e.Parameter;
                    UsernameTextBox.Text = _account.Username;
                    await SignInWindowsHelloAsync();
                }
            }
        }

        private async void LoginButton_Click(object sender,
RoutedEventArgs e)
        {
            ErrorMessage.Text = "";
            await SignInWindowsHelloAsync();
        }
    }
}
```

- As the **Login** page is using a `UserAccount` object instead of the previous `Account` object, the WindowsHelloHelper.cs will need to be updated to use a `UserAccount` as a parameter type for some methods. You will need to change the following parameters for the **CreateWindowsHelloKeyAsync**, **RemoveWindowsHelloAccountAsync**, and **GetWindowsHelloAuthenticationMessageAsync** methods. As the `UserAccount` class has a `Guid` for a *UserId*, you will start using the Id in more places to be more precise.

cs

```
public static async Task<bool> CreateWindowsHelloKeyAsync(Guid userId,
string username)
{
    KeyCredentialRetrievalResult keyCreationResult = await
KeyCredentialManager.RequestCreateAsync(username,
KeyCredentialCreationOption.ReplaceExisting);

    return true;
}

public static async Task RemoveWindowsHelloAccountAsync(UserAccount
account)
{

}
public static async Task<bool>
GetWindowsHelloAuthenticationMessageAsync(UserAccount account)
{
    KeyCredentialRetrievalResult openKeyResult = await
KeyCredentialManager.OpenAsync(account.Username);
    //Calling OpenAsync will allow the user access to what is available
in the app and will not require user credentials again.
    //If you wanted to force the user to sign in again you can use the
following:
    //var consentResult = await
Windows.Security.Credentials.UI.UserConsentVerifier.RequestVerification
Async(account.Username);
    //This will ask for the either the password of the currently signed
in Microsoft Account or the PIN used for Windows Hello.

    if (openKeyResult.Status == KeyCredentialStatus.Success)
    {
        //If OpenAsync has succeeded, the next thing to think about is
whether the client application requires access to backend services.
        //If it does here you would Request a challenge from the
Server. The client would sign this challenge and the server
        //would check the signed challenge. If it is correct it would
allow the user access to the backend.
        //You would likely make a new method called RequestSignAsync to
handle all this
```

```
        //for example, RequestSignAsync(openKeyResult);
        //Refer to the second Windows Hello sample for information on
how to do this.

        //For this sample there is not concept of a server implemented
so just return true.
        return true;
    }
    else if (openKeyResult.Status == KeyCredentialStatus.NotFound)
    {
        //If the _account is not found at this stage. It could be one
of two errors.
        //1. Windows Hello has been disabled
        //2. Windows Hello has been disabled and re-enabled cause the
Windows Hello Key to change.
        //Calling CreateWindowsHelloKeyAsync and passing through the
account will attempt to replace the existing Windows Hello Key for that
account.
        //If the error really is that Windows Hello is disabled then
the CreateWindowsHelloKeyAsync method will output that error.
        if (await CreateWindowsHelloKeyAsync(account.UserId,
account.Username))
        {
            //If the Windows Hello Key was again successfully created,
Windows Hello has just been reset.
            //Now that the Windows Hello Key has been reset for the
account retry sign in.
            return await
GetWindowsHelloAuthenticationMessageAsync(account);
        }
    }

    // Can't use Windows Hello right now, try again later
    return false;
}
```

- The **SignInWindowsHelloAsync** method in Login.xaml.cs file will need to be updated to use the **AuthService** instead of the **AccountHelper**. Validation of credentials will happen through the **AuthService**. For this hands on lab, the only configured account is "sampleUsername". This account is created in the **InitializeSampleUserAccountsAsync** method in MockStore.cs. Update the SignInWindowsHelloAsync method in Login.xaml.cs now to reflect the code snippet below.

```cs
private async Task SignInWindowsHelloAsync()
{
    if (_isExistingAccount)
    {
        if (await
```

```
WindowsHelloHelper.GetWindowsHelloAuthenticationMessageAsync(_account))
        {
            Frame.Navigate(typeof(Welcome), _account);
        }
    }
    else if
(AuthService.AuthService.Instance.ValidateCredentials(UsernameTextBox.T
ext, PasswordBox.Password))
    {
        Guid userId =
AuthService.AuthService.Instance.GetUserId(UsernameTextBox.Text);

        if (userId != Guid.Empty)
        {
            //Now that the account exists on server try and create the
necessary details and add them to the account
            if (await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(userId,
UsernameTextBox.Text))
            {
                Debug.WriteLine("Successfully signed in with Windows
Hello!");
                //Navigate to the Welcome Screen.
                _account =
AuthService.AuthService.Instance.GetUserAccount(userId);
                Frame.Navigate(typeof(Welcome), _account);
            }
            else
            {
                //The Windows Hello account creation failed.
                //Remove the account from the server as the details
were not configured
                await
AuthService.AuthService.Instance.WindowsHelloRemoveUserAsync(userId);

                ErrorMessage.Text = "Account Creation Failed";
            }
        }
    }
    else
    {
        ErrorMessage.Text = "Invalid Credentials";
    }
}
```

- As Windows Hello will create a different public and private key pair for each account on each device the **Welcome** page will need to display a list of registered devices for the logged in account and allow each one to be forgotten. In Welcome.xaml add in the following XAML underneath the `ForgetButton`. This will implement a forget device button, an error text area and a list to display all devices.

XML

```
<Grid>
  <StackPanel>
    <TextBlock x:Name="Title" Text="Welcome" FontSize="40"
TextAlignment="Center"/>
    <TextBlock x:Name="UserNameText" FontSize="28"
TextAlignment="Center"/>

    <Button x:Name="BackToUserListButton" Content="Back to User List"
Click="Button_Restart_Click"
            HorizontalAlignment="Center" Margin="0,20"
Foreground="White" Background="DodgerBlue"/>

    <Button x:Name="ForgetButton" Content="Forget Me"
Click="Button_Forget_User_Click"
            Foreground="White"
            Background="Gray"
            HorizontalAlignment="Center"/>

    <Button x:Name="ForgetDeviceButton" Content="Forget Device"
Click="Button_Forget_Device_Click"
            Foreground="White"
            Background="Gray"
            Margin="0,40,0,20"
            HorizontalAlignment="Center"/>

    <TextBlock x:Name="ForgetDeviceErrorTextBlock" Text="Select a
device first"
               TextWrapping="Wrap" Width="300" Foreground="Red"
               TextAlignment="Center" VerticalAlignment="Center"
FontSize="16" Visibility="Collapsed"/>

    <ListView x:Name="UserListView" MaxHeight="500" MinWidth="350"
Width="350" HorizontalAlignment="Center">
      <ListView.ItemTemplate>
        <DataTemplate>
          <Grid Background="Gray" Height="50" Width="350"
HorizontalAlignment="Center" VerticalAlignment="Stretch" >
            <TextBlock Text="{Binding DeviceId}"
HorizontalAlignment="Center" TextAlignment="Center"
VerticalAlignment="Center"
                       Foreground="White"/>
          </Grid>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </StackPanel>
</Grid>
```

- In the Welcome.xaml.cs file, you need to change the private `Account` variable at the top of the class to be a private `UserAccount` variable. Then update the

`OnNavigatedTo` method to use the **AuthService** and retrieve information for the current account. When you have the account information you can set the `ItemsSource` of the list to display the devices. You will need to add a reference to the **AuthService** namespace.

cs

```
using WindowsHelloLogin.AuthService;

namespace WindowsHelloLogin.Views
{
    public sealed partial class Welcome : Page
    {
        private UserAccount _activeAccount;

        public Welcome()
        {
            InitializeComponent();
        }

        protected override void OnNavigatedTo(NavigationEventArgs e)
        {
            _activeAccount = (UserAccount)e.Parameter;
            if (_activeAccount != null)
            {
                UserAccount account =
AuthService.AuthService.Instance.GetUserAccount(_activeAccount.UserId);
                if (account != null)
                {
                    UserListView.ItemsSource =
account.WindowsHelloDevices;
                    UserNameText.Text = account.Username;
                }
            }
        }
    }
}
```

- As you will be using the **AuthService** when removing an account, the reference to the **AccountHelper** in the `Button_Forget_User_Click` method can be removed. The method should now look as below.

cs

```
private async void Button_Forget_User_Click(object sender,
RoutedEventArgs e)
{
    //Remove it from Windows Hello
    await
WindowsHelloHelper.RemoveWindowsHelloAccountAsync(_activeAccount);
```

```
    Debug.WriteLine($"User {_activeAccount.Username} deleted.");

    //Navigate back to UserSelection page.
    Frame.Navigate(typeof(UserSelection));
}
```

- The **WindowsHelloHelper** method is not using the **AuthService** to remove the account. You need to make a call to the **AuthService** and pass the *userId*.

```cs
public static async Task RemoveWindowsHelloAccountAsync(UserAccount
account)
{
    //Open the account with Windows Hello
    KeyCredentialRetrievalResult keyOpenResult = await
KeyCredentialManager.OpenAsync(account.Username);

    if (keyOpenResult.Status == KeyCredentialStatus.Success)
    {
        // In the real world you would send key information to server
to unregister
        await
AuthService.AuthService.Instance.WindowsHelloRemoveUserAsync(account.Us
erId);
    }

    //Then delete the account from the machines list of Windows Hello
Accounts
    await KeyCredentialManager.DeleteAsync(account.Username);
}
```

- Before you can finish implementing the **Welcome** page, you need to create a method in WindowsHelloHelper.cs that will allow a device to be removed. Create a new method that will call **WindowsHelloRemoveDeviceAsync** in **AuthService**.

```cs
public static async Task RemoveWindowsHelloDeviceAsync(UserAccount
account, Guid deviceId)
{
    await
AuthService.AuthService.Instance.WindowsHelloRemoveDeviceAsync(account.
UserId, deviceId);
}
```

- In Welcome.xaml.cs, implement the **Button_Forget_Device_Click** event handler. This will use the selected device from the list of devices and use the Windows Hello

helper to call remove device. Remember to make the event handler async.

```cs
private async void Button_Forget_Device_Click(object sender,
RoutedEventArgs e)
{
    WindowsHelloDevice selectedDevice = UserListView.SelectedItem as
WindowsHelloDevice;
    if (selectedDevice != null)
    {
        //Remove it from Windows Hello
        await
WindowsHelloHelper.RemoveWindowsHelloDeviceAsync(_activeAccount,
selectedDevice.DeviceId);

        Debug.WriteLine($"User {_activeAccount.Username} deleted.");

        if (!UserListView.Items.Any())
        {
            //Navigate back to UserSelection page.
            Frame.Navigate(typeof(UserSelection));
        }
    }
    else
    {
        ForgetDeviceErrorTextBlock.Visibility = Visibility.Visible;
    }
}
```

- The next page you will update is the **UserSelection** page. The **UserSelection** page will need to use the **AuthService** to retrieve all user accounts for the current device. Currently, there is no way for you get a device ID to pass to the **AuthService** so it can return user accounts for that device. In the **Utils** folder, create a new class called "Helpers.cs". Change the class scope to be public static and then add the following method that will allow you to retrieve the current device id.

```cs
using System;
using Windows.Security.ExchangeActiveSyncProvisioning;

namespace WindowsHelloLogin.Utils
{
    public static class Helpers
    {
        public static Guid GetDeviceId()
        {
            //Get the Device ID to pass to the server
```

```
            var deviceInformation = new EasClientDeviceInformation();
            return deviceInformation.Id;
        }
    }
}
```

- In the **UserSelection** page class, only the code-behind needs to change, not the user interface. In UserSelection.xaml.cs, update the **UserSelection_Loaded** method and the **UserSelectionChanged** method to use the `UserAccount` class instead of the `Account` class. You will also need to get all users for this device through the **AuthService**.

cs

```
using System.Linq;
using WindowsHelloLogin.AuthService;

namespace WindowsHelloLogin.Views
{
    public sealed partial class UserSelection : Page
    {
        public UserSelection()
        {
            InitializeComponent();
            Loaded += UserSelection_Loaded;
        }

        private void UserSelection_Loaded(object sender,
RoutedEventArgs e)
        {
            List<UserAccount> accounts =
AuthService.AuthService.Instance.GetUserAccountsForDevice(Helpers.GetDe
viceId());

            if (accounts.Any())
            {
                UserListView.ItemsSource = accounts;
                UserListView.SelectionChanged += UserSelectionChanged;
            }
            else
            {
                //If there are no accounts navigate to the Login page
                Frame.Navigate(typeof(Login));
            }
        }

        /// <summary>
        /// Function called when an account is selected in the list of
accounts
        /// Navigates to the Login page and passes the chosen account
        /// </summary>
        private void UserSelectionChanged(object sender,
```

```
RoutedEventArgs e)
        {
            if (((ListView)sender).SelectedValue != null)
            {
                UserAccount account = (UserAccount)
((ListView)sender).SelectedValue;
                if (account != null)
                {
                    Debug.WriteLine($"Account {account.Username}
selected!");
                }
                Frame.Navigate(typeof(Login), account);
            }
        }
    }
}
```

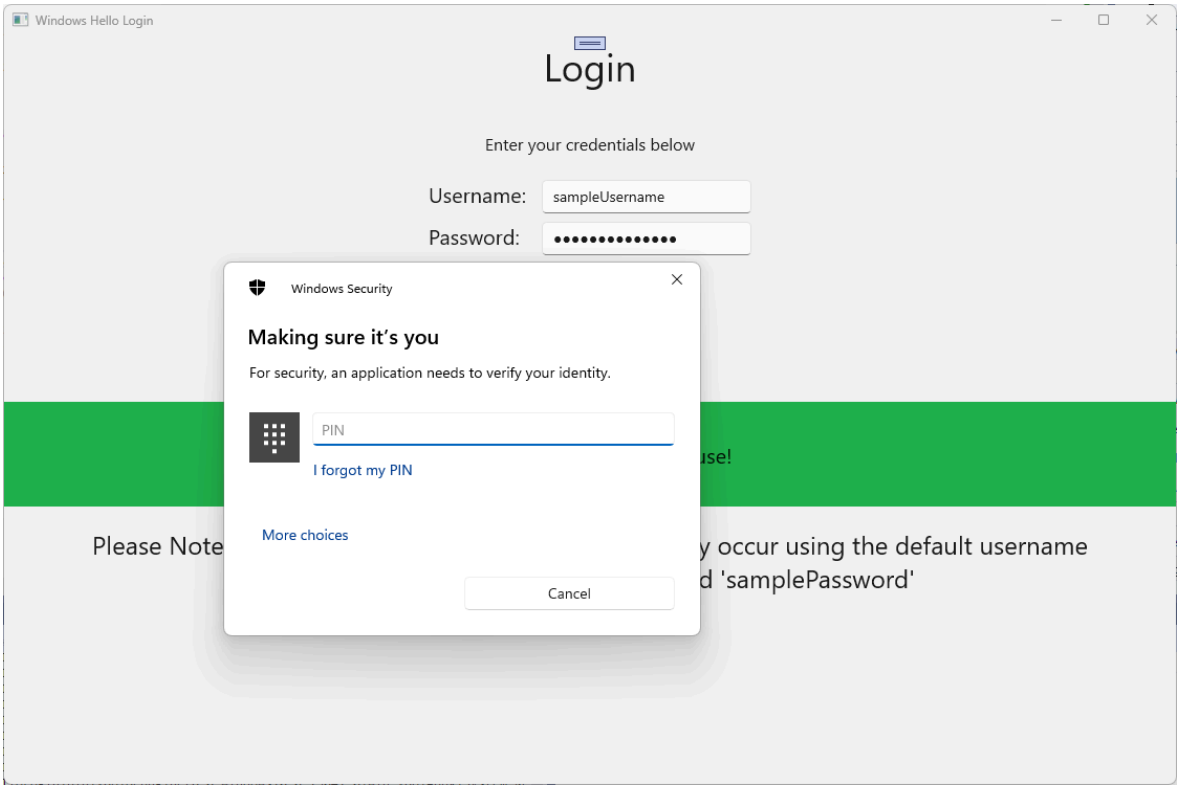
- The **WindowsHelloRegister** page needs to have the code-behind file updated. The user interface does not need any changes. In WindowsHelloRegister.xaml.cs, remove the private `Account` variable at the top of the class, as it's no longer needed. Update the **RegisterButton_Click_Async** event handler to use the **AuthService**. This method will create a new **UserAccount** and then try and update its account details. If Windows Hello fails to create a key, the account will be removed as the registration process failed.

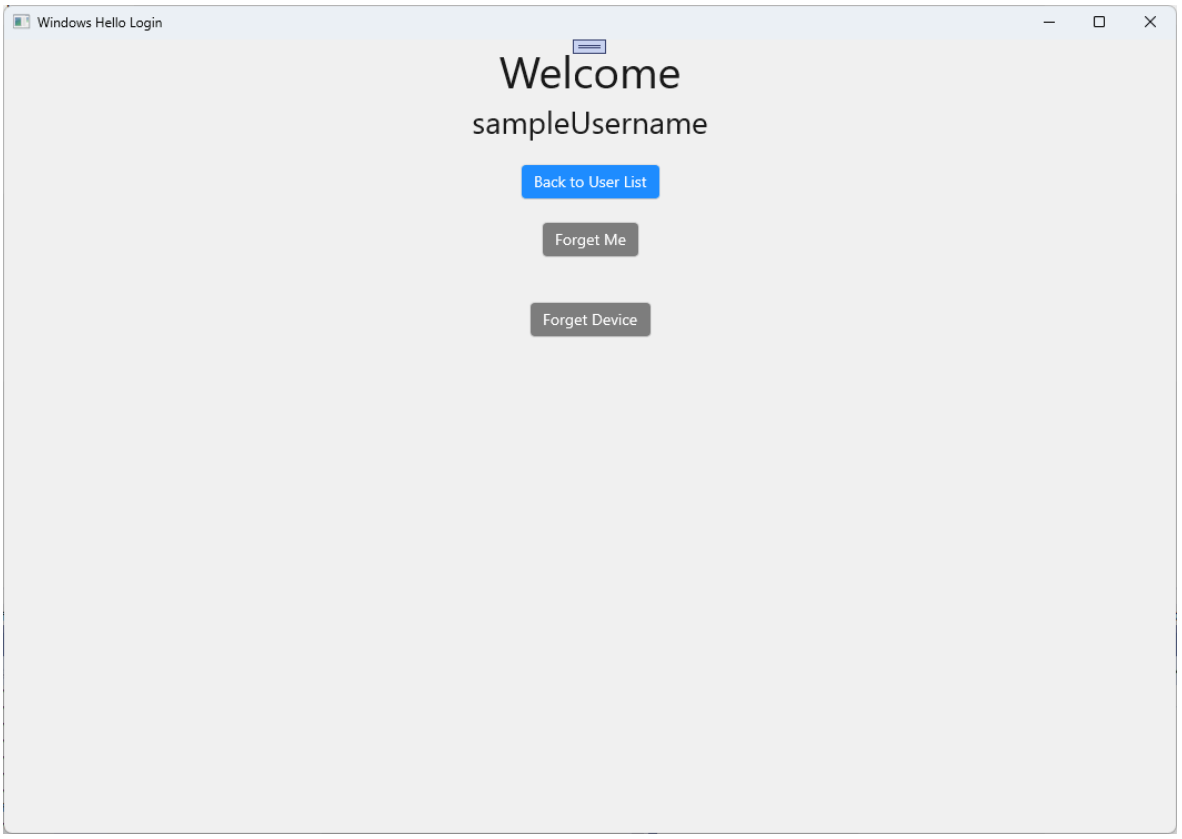
cs

```
private async void RegisterButton_Click_Async(object sender,
RoutedEventArgs e)
{
    ErrorMessage.Text = "";

    //Validate entered credentials are acceptable
    if (!string.IsNullOrEmpty(UsernameTextBox.Text))
    {
        //Register an Account on the AuthService so that we can get
back a userId
        await
AuthService.AuthService.Instance.RegisterAsync(UsernameTextBox.Text);
        Guid userId =
AuthService.AuthService.Instance.GetUserId(UsernameTextBox.Text);

        if (userId != Guid.Empty)
        {
            //Now that the account exists on server try and create the
necessary details and add them to the account
            if (await
WindowsHelloHelper.CreateWindowsHelloKeyAsync(userId,
UsernameTextBox.Text))
            {
                //Navigate to the Welcome Screen.
```

```
                Frame.Navigate(typeof(Welcome),
AuthService.AuthService.Instance.GetUserAccount(userId));
            }
            else
            {
                //The Windows Hello account creation failed.
                //Remove the account from the server as the details
were not configured
                await
AuthService.AuthService.Instance.WindowsHelloRemoveUserAsync(userId);

                ErrorMessage.Text = "Account Creation Failed";
            }
        }
    }
    else
    {
        ErrorMessage.Text = "Please enter a username";
    }
}
```

- Build and run the application. Sign in to the sample user account with the credentials "sampleUsername" and "samplePassword". On the welcome screen, you may notice the Forget devices button is displayed, but there are no devices. When you are creating or migrating a user to work with Windows Hello the account information is not being pushed to the **AuthService**.





- To get the Windows Hello account information to the **AuthService**, the WindowsHelloHelper.cs will need to be updated. In the **CreateWindowsHelloKeyAsync** method, instead of only returning `true` in the case that's successful, you will need to call a new method which will try to get the **KeyAttestation**. While this hands on lab is not recording this information in the **AuthService**, you'll learn how you would get this information on the client side. Update the **CreateWindowsHelloKeyAsync** method as follows:

cs

```
public static async Task<bool> CreateWindowsHelloKeyAsync(Guid userId,
string username)
{
    KeyCredentialRetrievalResult keyCreationResult = await
KeyCredentialManager.RequestCreateAsync(username,
KeyCredentialCreationOption.ReplaceExisting);

    switch (keyCreationResult.Status)
    {
        case KeyCredentialStatus.Success:
            Debug.WriteLine("Successfully made key");
            await GetKeyAttestationAsync(userId, keyCreationResult);
            return true;
        case KeyCredentialStatus.UserCanceled:
            Debug.WriteLine("User cancelled sign-in process.");
            break;
        case KeyCredentialStatus.NotFound:
            // User needs to setup Windows Hello
            Debug.WriteLine($"Windows Hello is not set up!
```

```
{Environment.NewLine}Please go to Windows Settings and set up a PIN to
use it.");
                break;
            default:
                break;
        }

        return false;
    }
```

- Create a **GetKeyAttestationAsync** method in WindowsHelloHelper.cs. This method will demonstrate how to obtain all the necessary information that can be provided by Windows Hello for each account on a specific device.

cs

```
using Windows.Storage.Streams;

private static async Task GetKeyAttestationAsync(Guid userId,
KeyCredentialRetrievalResult keyCreationResult)
{
    KeyCredential userKey = keyCreationResult.Credential;
    IBuffer publicKey = userKey.RetrievePublicKey();
    KeyCredentialAttestationResult keyAttestationResult = await
userKey.GetAttestationAsync();
    IBuffer keyAttestation = null;
    IBuffer certificateChain = null;
    bool keyAttestationIncluded = false;
    bool keyAttestationCanBeRetrievedLater = false;
    KeyCredentialAttestationStatus keyAttestationRetryType = 0;

    if (keyAttestationResult.Status ==
KeyCredentialAttestationStatus.Success)
    {
        keyAttestationIncluded = true;
        keyAttestation = keyAttestationResult.AttestationBuffer;
        certificateChain = keyAttestationResult.CertificateChainBuffer;
        Debug.WriteLine("Successfully made key and attestation");
    }
    else if (keyAttestationResult.Status ==
KeyCredentialAttestationStatus.TemporaryFailure)
    {
        keyAttestationRetryType =
KeyCredentialAttestationStatus.TemporaryFailure;
        keyAttestationCanBeRetrievedLater = true;
        Debug.WriteLine("Successfully made key but not attestation");
    }
    else if (keyAttestationResult.Status ==
KeyCredentialAttestationStatus.NotSupported)
    {
        keyAttestationRetryType =
KeyCredentialAttestationStatus.NotSupported;
        keyAttestationCanBeRetrievedLater = false;
```

```
        Debug.WriteLine("Key created, but key attestation not
supported");
    }

    Guid deviceId = Helpers.GetDeviceId();

    //Update the Windows Hello details with the information we have
just fetched above.
    //await UpdateWindowsHelloDetailsAsync(userId, deviceId,
publicKey.ToArray(), keyAttestationResult);
}
```

- You may have noticed in the **GetKeyAttestationAsync** method that you just added the last line was commented out. This last line will be a new method you create that will send all the Windows Hello information to the **AuthService**. In the real world, you would need to send this to an actual server through a Web API.

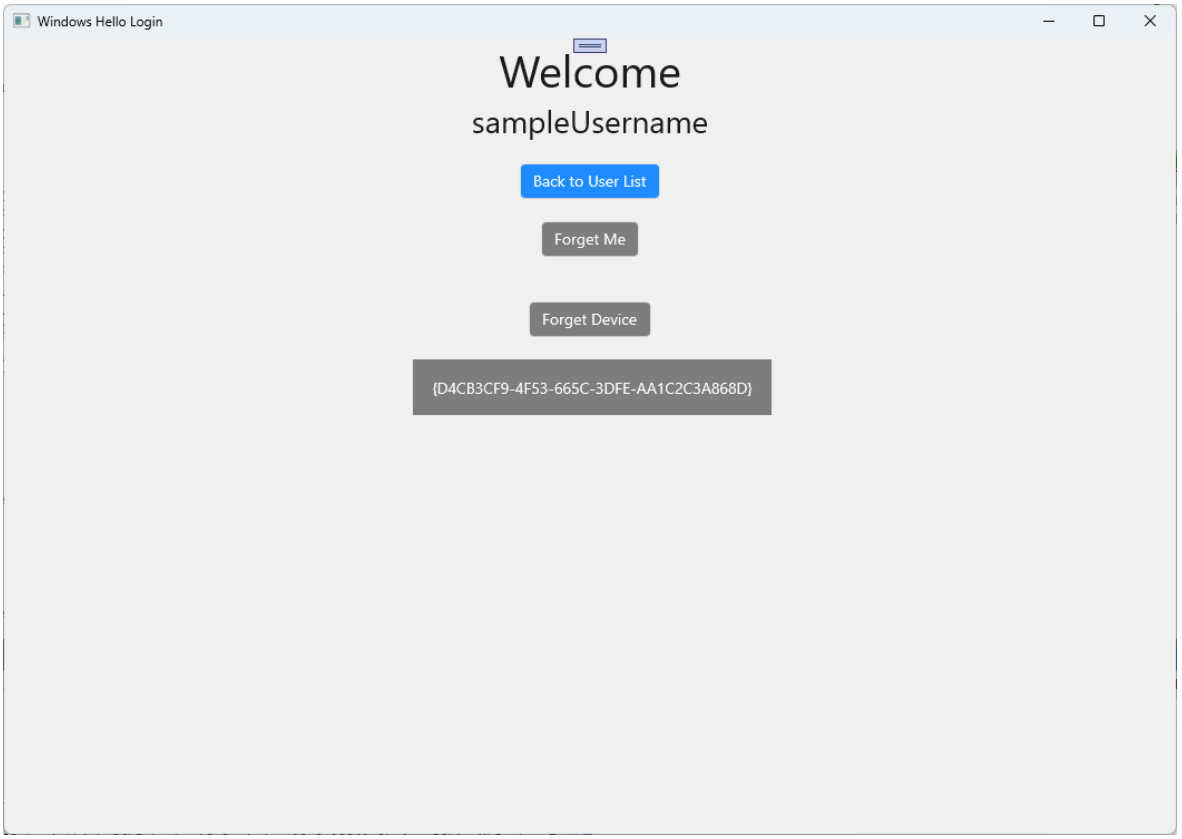
cs

```
using System.Runtime.InteropServices.WindowsRuntime;
using System.Threading.Tasks;

public static async Task<bool> UpdateWindowsHelloDetailsAsync(Guid
userId, Guid deviceId, byte[] publicKey, KeyCredentialAttestationResult
keyAttestationResult)
{
    //In the real world, you would use an API to add Windows Hello
signing info to server for the signed in account.
    //For this tutorial, we do not implement a Web API for our server
and simply mock the server locally.
    //The CreateWindowsHelloKey method handles adding the Windows Hello
account locally to the device using the KeyCredential Manager

    //Using the userId the existing account should be found and
updated.
    await
AuthService.AuthService.Instance.WindowsHelloUpdateDetailsAsync(userId,
deviceId, publicKey, keyAttestationResult);
    return true;
}
```

- Uncomment the last line in the **GetKeyAttestationAsync** method so that the Windows Hello information is being sent to the **AuthService**.

- Build and run the application and sign in with the default credentials as before. On the **Welcome** page, you will now see that the device Id is displayed. If you signed in on another device that would also be displayed here (if you had a cloud hosted auth service). For this hands on lab, the actual device Id is being displayed. In a real

implementation, you would want to display a friendly name that a person could understand and use to identify each device.



- To complete this hands on lab, you need a request and challenge for the user when they select from the user selection page and sign back in. The **AuthService** has two methods that you created to request a challenge, one that uses a signed challenge. In WindowsHelloHelper.cs, create a new method named **RequestSignAsync**. This will request a challenge from the **AuthService**, locally sign that challenge using a Windows Hello API and send the signed challenge to the **AuthService**. In this hands on lab, the **AuthService** will receive the signed challenge and return `true`. In an actual implementation, you would need to implement a verification mechanism to determine if the challenge was signed by the correct user on the correct device. Add the method below to the WindowsHelloHelper.cs

cs

```
private static async Task<bool> RequestSignAsync(Guid userId,
KeyCredentialRetrievalResult openKeyResult)
{
    // Calling userKey.RequestSignAsync() prompts the uses to enter the
PIN or use Biometrics (Windows Hello).
    // The app would use the private key from the user account to sign
the sign-in request (challenge)
    // The client would then send it back to the server and await the
servers response.
    IBuffer challengeMessage =
```

```
AuthService.AuthService.Instance.WindowsHelloRequestChallenge();
    KeyCredential userKey = openKeyResult.Credential;
    KeyCredentialOperationResult signResult = await
userKey.RequestSignAsync(challengeMessage);

    if (signResult.Status == KeyCredentialStatus.Success)
    {
        // If the challenge from the server is signed successfully
        // send the signed challenge back to the server and await the
servers response
        return
AuthService.AuthService.Instance.SendServerSignedChallenge(
            userId, Helpers.GetDeviceId(),
signResult.Result.ToArray());
    }
    else if (signResult.Status == KeyCredentialStatus.UserCanceled)
    {
        // User cancelled the Windows Hello PIN entry.
    }
    else if (signResult.Status == KeyCredentialStatus.NotFound)
    {
        // Must recreate Windows Hello key
    }
    else if (signResult.Status ==
KeyCredentialStatus.SecurityDeviceLocked)
    {
        // Can't use Windows Hello right now, remember that hardware
failed and suggest restart
    }
    else if (signResult.Status == KeyCredentialStatus.UnknownError)
    {
        // Can't use Windows Hello right now, try again later
    }

    return false;
}
```

- In the **WindowsHelloHelper** class, call the **RequestSignAsync** method from the **GetWindowsHelloAuthenticationMessageAsync** method.

cs

```
public static async Task<bool>
GetWindowsHelloAuthenticationMessageAsync(UserAccount account)
{
    KeyCredentialRetrievalResult openKeyResult = await
KeyCredentialManager.OpenAsync(account.Username);
    // Calling OpenAsync will allow the user access to what is
available in the app and will not require user credentials again.
    // If you wanted to force the user to sign in again you can use the
following:
    // var consentResult = await
Windows.Security.Credentials.UI.UserConsentVerifier.RequestVerification
```

```
Async(account.Username);
    // This will ask for the either the password of the currently
signed in Microsoft Account or the PIN used for Windows Hello.

    if (openKeyResult.Status == KeyCredentialStatus.Success)
    {
        //If OpenAsync has succeeded, the next thing to think about is
whether the client application requires access to backend services.
        //If it does here you would Request a challenge from the
Server. The client would sign this challenge and the server
        //would check the signed challenge. If it is correct it would
allow the user access to the backend.
        //You would likely make a new method called RequestSignAsync to
handle all this
        //for example, RequestSignAsync(openKeyResult);
        //Refer to the second Windows Hello sample for information on
how to do this.

        return await RequestSignAsync(account.UserId, openKeyResult);
    }
    else if (openKeyResult.Status == KeyCredentialStatus.NotFound)
    {
        //If the _account is not found at this stage. It could be one
of two errors.
        //1. Windows Hello has been disabled
        //2. Windows Hello has been disabled and re-enabled cause the
Windows Hello Key to change.
        //Calling CreateWindowsHelloKeyAsync and passing through the
account will attempt to replace the existing Windows Hello Key for that
account.
        //If the error really is that Windows Hello is disabled then
the CreateWindowsHelloKeyAsync method will output that error.
        if (await CreateWindowsHelloKeyAsync(account.UserId,
account.Username))
        {
            //If the Windows Hello Key was again successfully created,
Windows Hello has just been reset.
            //Now that the Windows Hello Key has been reset for the
_account retry sign in.
            return await
GetWindowsHelloAuthenticationMessageAsync(account);
        }
    }

    // Can't use Windows Hello right now, try again later
    return false;
}
```

- Throughout this exercise, you have updated the client side application to use the **AuthService**. By doing this, you have been able to eliminate the need for the **Account** class and the **AccountHelper** class. Delete the **Account** class, the **Models** folder, and the **AccountHelper** class in the **Utils** folder. You'll need to remove all

reference to the `WindowsHelloLogin.Models` namespace throughout the application before the solution will successfully build.

- Build and run the application and enjoy using Windows Hello with the mock service and database.

In this hands on lab, you have learned how to use the Windows Hello APIs to replace the need for passwords when using authentication from a Windows machine. When you consider how much energy is expended by people maintaining passwords and supporting lost passwords in existing systems, you should see the benefit of moving to this new Windows Hello system of authentication.

We have left as an exercise for you the details of how you'll implement the authentication on the service and server side. It's expected that most developers will have existing systems that will need to be migrated to start working with Windows Hello. The details of each of these system will differ.

Related topics

- Windows Hello
- Windows Hello login app

Speech, voice, and conversation in Windows 11 and Windows 10

Article • 07/10/2024

Speech can be an effective, natural, and enjoyable way for people to interact with your Windows applications, complementing, or even replacing, traditional interaction experiences based on mouse, keyboard, touch, controller, or gestures.

Speech-based features such as speech recognition, dictation, speech synthesis (also known as text-to-speech or TTS), and conversational voice assistants (such as Cortana or Alexa) can provide accessible and inclusive user experiences that enable people to use your applications when other input devices might not suffice.

This page provides information on how the various Windows development frameworks provide speech recognition, speech synthesis, and conversation support for developers building Windows applications.

Platform-specific documentation

Universal Windows Platform (UWP)

Build speech-enabled apps on the modern platform for Windows 10 (and later) applications and games, on any Windows device (including PCs, phones, Xbox, HoloLens, and more), and publish them to the Microsoft Store.

Speech interactions

Speech recognition

Continuous dictation

Speech synthesis

Conversational agents

Cortana interactions in Windows apps
(not supported in Windows 10 May 2020 Update and newer)

Win32 platform

Develop speech-enabled applications for Windows desktop and Windows Server using the tools, information, and sample engines and applications provided here.

Microsoft Speech Platform - Software Development Kit (SDK) (Version 11)

Microsoft Speech SDK, version 5.1

.NET Framework

Develop accessible apps and tools on the established platform for managed Windows applications with a XAML UI model and the .NET Framework.

System.Speech Programming Guide for .NET Framework

Azure speech services

Integrate speech processing into apps and services.

Speech to text

Text to speech

Speech translation

Speaker Recognition

Voice-first virtual assistants

Legacy features

Legacy, deprecated, and/or unsupported versions of Microsoft speech and conversation technology.

Microsoft Agent

Microsoft Speech API (SAPI) 5.3

Microsoft Speech API (SAPI) 5.4

The Bing Speech Recognition Control

Samples

Download and run full Windows samples that demonstrate various accessibility features and functionality.

Code sample browser

The new samples browser (replaces the MSDN Code Gallery).

Windows classic samples on GitHub

These samples demonstrate the functionality and programming model for Windows and Windows Server.

Universal Windows Platform (UWP) samples on GitHub

These samples demonstrate the API usage patterns for the Universal Windows Platform (UWP) in the Windows Software Development Kit (SDK) for Windows 10 and later.

WinUI 2 Gallery

This app demonstrates the various Xaml controls supported in the Fluent Design System.

Other resources

Blogs and news

The latest from the world of Microsoft speech.

Community and support

Where Windows developers and users meet and learn together.

In the news

Speech blogs

Windows community - Speech

Windows Speech Developer's Forum

Stack Overflow

User interface and input

Article • 10/27/2022

This article provides an overview of the Windows UI frameworks that are currently maintained by Microsoft and compares their capabilities.

Microsoft produces both UI frameworks and app platforms. App platforms typically include a UI framework, while UI frameworks are either standalone (not shipped with an app platform) or can be used with multiple app platforms (see Choose your app platform).

The frameworks discussed here include WinUI for both Windows App SDK (WinUI 3) and UWP (WinUI 2), Windows Presentation Foundation (WPF), and Windows Forms (WinForms).

User interface fundamentals

When building a modern Windows app, you have a selection of UI frameworks to choose from. UI frameworks provide your app with built in controls, styles, animations, input handling, and more.

There are five main components that go into creating a user interface for your Windows app. These components are usually built into each UI framework.

- Controls and styles
    - A control is a UI element that displays content or enables interaction. Controls are the building blocks of the user interface.

      Here's an example of a Button control that's available in UWP, WinUI 2, and WinUI 3. When you place this control into your app, it automatically receives the default design that the UI framework provides.

    - Styles consist of colors, typography, icons, Fluent materials, and more that can be used throughout your app's design to create a truly unique experience.

Here's an example of a style component called Acrylic, available in WinUI 2 and WinUI 3. Acrylic is a brush that you can use on surfaces within your app or as the background of your app. It provides a translucent texture.

- Input and interaction

  End users may interact with your app and provide input to your app (such as selection or typing) through different methods. Some examples of input are:
  - Mouse
  - Touch
  - Gamepad
  - Pen
  - Keyboard
  - Surface Dial
  - Touchpad
  - Speech

- Device support

  End users access Windows apps on a variety of devices, and UI frameworks may only support certain ones. Some common devices for Windows apps to run on are:
  - PCs (often referred to as "desktop", but includes laptops as well)
  - Tablets
  - HoloLens
  - Xbox
  - Surface Hub

- Motion and animation

Built-in animations can really give your app a polished look and feel, and provide consistency with first-party apps throughout Windows.

An example of a built-in animation in UWP, WinUI 2, and WinUI 3 is the animation that occurs when the end user switches between light and dark mode. When the end user switches modes for their entire PC, the app's UI will automatically update as well with a transition animation.

- Usability and accessibility

  In order to ensure your app is delightful to use for every single user, you must take accessibility into account.

  UI frameworks provide built-in accessibility to controls and styles with purposeful keyboard behavior, screenreader support and more. Many also provide APIs for accessible actions in custom controls, like interacting with screenreaders.

UI frameworks

Each UI framework released by Microsoft has unique capabilities, follows different design languages, and provides different experiences to the end user. This section will compare all the main UI frameworks that you'll be choosing from when you begin to build your app.

The table below shows a brief summary of a few main capabilities between these UI frameworks. For more details on each framework, navigate through the tabs further below.

Expand table

Capability	Windows App SDK (WinUI 3)	WinUI 2 for UWP	WPF	WinForms
Languages supported	C#/.NET 6 and later, C++/WinRT	C#/.NET Native, C++/WinRT, C++/CX, VB	C#/.NET 6 (and later) and .NET Framework, C++/CLI (Managed Extensions for C++), F#, VB	C#/.NET 6 (and later) and .NET Framework, C++/CLI (Managed Extensions for C++), F#, VB
Devices supported	PCs (incl. laptops and tablets), support for all Windows 10	All Windows 10 devices (PCs, tablets, HoloLens,	Desktop PCs and laptops	Desktop PCs and laptops

Capability	Windows App SDK (WinUI 3)	WinUI 2 for UWP	WPF	WinForms
	devices coming soon	Xbox, Surface Dial, and more)		
Inputs supported	All Windows 10 inputs supported	All Windows 10 inputs supported	Mouse and keyboard	Mouse and keyboard
Windows OS version supported	Windows 10 version 1809 or later	Windows 10 version 1703 or later	Windows XP or later	Windows XP or later
WebView support	Chromium-based WebView2	Non-chromium WebView	WebView2 support coming soon	WebView2 support coming soon
Open Source	Coming soon	Yes	Yes (.NET 6 and later only)	Yes (.NET 6 and later only)

For more information about each of these UI frameworks, see the information on the following tabs.

Windows App SDK (WinUI 3)

Windows App SDK (WinUI 3)

For most new Windows apps, we recommend WinUI with the Windows App SDK (WinUI 3) to build your user interface. WinUI 3 provides consistent, intuitive, and accessible experiences using the latest user interface (UI) patterns.

WinUI 3 is completely decoupled from the Windows OS and ships as a part of the Windows App SDK, which is a set of tools and components that represent the next evolution in the Windows app development platform.

WinUI 3 is the latest generation of WinUI. WinUI 2 and 3 share many of the same controls, styles, and other UI fundamentals (see Comparison of WinUI 2 and WinUI 3).

Key app scenarios enabled by WinUI 3

- Modern Windows apps that need to run on a variety of modern devices, with a range of modern inputs
- Desktop/Win32 apps that are written in C++

- Graphics-heavy apps or games that want to take advantage of DirectX and Win2D
- Apps with a lot of integrated web content that need high-performance
- Apps that seek to provide experiences that "fit right in" on the Windows OS and with other first party Windows apps

Helpful documentation for WinUI 3

- Overview: WinUI 3
- Get started: Create your first WinUI 3 project
- Writing XAML: XAML Platform
- API Reference: WinUI API Reference
- Controls: Controls overview
- Input: Input and interactions
- Motion: Motion for Windows apps
- Accessibility: Usability for Windows apps
- Samples: WinUI 3 Gallery app

Apply Mica or Acrylic materials in desktop apps for Windows 11

Article • 07/16/2024

Materials in Windows 11 are visual effects applied to UX surfaces that resemble real life artifacts. Occluding materials, like Mica and Acrylic, are used as base layers beneath interactive UI controls.

Mica is an opaque material that incorporates the user's theme and desktop wallpaper to create a highly personalized appearance. Mica is designed for performance as it only captures the background wallpaper once to create its visualization, so we recommend it for the foundation layer of your app, especially in the title bar area.

Acrylic is a semi-transparent material that replicates the effect of frosted glass. It's used only for transient, light-dismiss surfaces such as flyouts and context menus.

This article describes how to apply Mica or Acrylic as the base layer of your Windows App SDK/WinUI 3 XAML app.

> **Note**
>
> - To use an in-app AcrylicBrush, see Acrylic material.
> - To use backdrop materials in a Win32 app, see Apply Mica in Win32 desktop apps for Windows 11.
> - To use backdrop materials in a UWP/WinUI 2 app, see Apply Mica with WinUI 2 for UWP or Acrylic material.

How to use a backdrop material

✓ **Important APIs**: Window.SystemBackdrop property, MicaBackdrop class, DesktopAcrylicBackdrop class, SystemBackdropConfiguration class

Open the WinUI 3 Gallery app and see the System Backdrops in action

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

To apply Mica or Acrylic material to your app, you set the `SystemBackdrop` property to a XAML SystemBackdrop (typically, one of the built-in backdrops, MicaBackdrop or DesktopAcrylicBackdrop).

These elements have a `SystemBackdrop` property:

- CommandBarFlyoutCommandBar.SystemBackdrop
- ContentIsland.SystemBackdrop
- DesktopWindowXamlSource.SystemBackdrop
- FlyoutBase.SystemBackdrop
- MenuFlyoutPresenter.SystemBackdrop
- Popup.SystemBackdrop
- Window.SystemBackdrop

These examples show how to set the system backdrop in XAML and in code.

Mica

Mica is typically used as the backdrop for an app Window.

XAML

```
<Window
    ... >

    <Window.SystemBackdrop>
        <MicaBackdrop Kind="BaseAlt"/>
    </Window.SystemBackdrop>

</Window>
```

C#

```
public MainWindow()
{
    this.InitializeComponent();

    SystemBackdrop = new MicaBackdrop()
                         { Kind = MicaKind.BaseAlt };
}
```

Acrylic

Acrylic is typically used as the backdrop for transient UI, like a flyout.

XAML

```
<Flyout
    ... >

    <Flyout.SystemBackdrop>
        <DesktopAcrylicBackdrop/>
    </Flyout.SystemBackdrop>
</Flyout>
```

C#

```
Flyout flyout = new Flyout()
{
    SystemBackdrop = new DesktopAcrylicBackdrop()
};
```

How to use a system backdrop controller

> **Note**
>
> Starting with Windows App SDK 1.3, you can apply material by setting the `Window.SystemBackdrop` property to a XAML `SystemBackdrop` as described in the previous section. This is the recommended way to apply a material.
>
> The remainder of this article shows how to use the Composition MicaController and DesktopAcrylicController APIs.

To use a backdrop material in your app, you can use one of the controllers that implements the ISystemBackdropController interface (MicaController or DesktopAcrylicController). These classes manage both the rendering of the system backdrop material as well as the handling of system policy for the material.

To use Mica as your backdrop material, create a MicaController object. To use Acrylic, create a DesktopAcrylicController object. The set up and supporting code is the same for each type of system backdrop material.

This code shows how to create a `MicaController`.

C#

```
MicaController m_backdropController;

bool TrySetSystemBackdrop()
```

```
{
    if (MicaController.IsSupported())
    {
        ...
        m_backdropController = new MicaController();
        ...
    }
}
```

To use the Mica Alt variant of Mica, create a `MicaController` object and set the Kind property to MicaKind.BaseAlt.

C#

```
MicaController m_backdropController;

bool TrySetSystemBackdrop()
{
    if (MicaController.IsSupported())
    {
        ...
        m_backdropController = new MicaController()
        {
            Kind = MicaKind.BaseAlt
        };
        ...
    }
}
```

This code shows how to create a `DesktopAcrylicController`.

C#

```
DesktopAcrylicController m_backdropController;

bool TrySetSystemBackdrop()
{
    if (DesktopAcrylicController.IsSupported())
    {
        ...
        m_backdropController = new DesktopAcrylicController();
        ...
    }
}
```

The controller reacts to the system Light and Dark themes by default. To override this behavior, you can set the following properties on the controller:

- FallbackColor

- LuminosityOpacity
- TintColor
- TintOpacity

> **Note**
>
> After customizing any of the controller's four properties, it no longer applies default Light or Dark values when the associated SystemBackdropConfiguration.Theme changes. You need to manually update those properties to match the new theme.

In order to use the backdrop material in your app, the following items are required:

- **System support**

  The system where the app runs must support the backdrop material. Call the MicaController.IsSupported or DesktopAcrylicController.IsSupported method to ensure the backdrop material is supported at runtime.

- **A valid target**

  You must provide a target that implements the ICompositionSupportsSystemBackdrop interface. In a XAML app, the XAML Window implements this interface and is used as the backdrop target.

- **A SystemBackdropConfiguration object**

  The SystemBackdropConfiguration provides the system backdrop controller with app-specific policy information to properly configure the system backdrop material.

- **A DispatcherQueue object**.

  You need an available Windows.System.DispatcherQueue on the main XAML thread. See the `WindowsSystemDispatcherQueueHelper` class in the example code, or in the WinUI 3 Gallery sample.

Example: Use Mica in a Windows AppSDK/WinUI 3 app

This example shows how to set up the Mica backdrop material in a XAML app.

> **Tip**
>
> Also, see these example projects on GitHub:
>
> **C#**: SampleSystemBackdropsWindow in the WinUI3 Gallery.
>
> **C++/WinRT**: Windows App SDK Mica sample.

C#

```
using Microsoft.UI.Composition.SystemBackdrops;
using Microsoft.UI.Xaml;
using System.Runtime.InteropServices; // For DllImport
using WinRT; // required to support
Window.As<ICompositionSupportsSystemBackdrop>()

public sealed partial class MainWindow : Window
{
    WindowsSystemDispatcherQueueHelper m_wsdqHelper; // See below for
implementation.
    MicaController m_backdropController;
    SystemBackdropConfiguration m_configurationSource;

    public MainWindow()
    {
        this.InitializeComponent();

        TrySetSystemBackdrop();
    }

    bool TrySetSystemBackdrop()
    {
        if
(Microsoft.UI.Composition.SystemBackdrops.MicaController.IsSupported())
        {
            m_wsdqHelper = new WindowsSystemDispatcherQueueHelper();
            m_wsdqHelper.EnsureWindowsSystemDispatcherQueueController();

            // Create the policy object.
            m_configurationSource = new SystemBackdropConfiguration();
            this.Activated += Window_Activated;
            this.Closed += Window_Closed;
            ((FrameworkElement)this.Content).ActualThemeChanged +=
Window_ThemeChanged;

            // Initial configuration state.
            m_configurationSource.IsInputActive = true;
            SetConfigurationSourceTheme();

            m_backdropController = new
Microsoft.UI.Composition.SystemBackdrops.MicaController();
```

```
            // Enable the system backdrop.
            // Note: Be sure to have "using WinRT;" to support the
Window.As<...>() call.

m_backdropController.AddSystemBackdropTarget(this.As<Microsoft.UI.Compositio
n.ICompositionSupportsSystemBackdrop>());

m_backdropController.SetSystemBackdropConfiguration(m_configurationSource);
            return true; // succeeded
        }

        return false; // Mica is not supported on this system
    }

    private void Window_Activated(object sender, WindowActivatedEventArgs
args)
        {
            m_configurationSource.IsInputActive = args.WindowActivationState
!= WindowActivationState.Deactivated;
        }

    private void Window_Closed(object sender, WindowEventArgs args)
        {
            // Make sure any Mica/Acrylic controller is disposed
            // so it doesn't try to use this closed window.
            if (m_backdropController != null)
            {
                m_backdropController.Dispose();
                m_backdropController = null;
            }
            this.Activated -= Window_Activated;
            m_configurationSource = null;
        }

    private void Window_ThemeChanged(FrameworkElement sender, object args)
        {
            if (m_configurationSource != null)
            {
                SetConfigurationSourceTheme();
            }
        }

    private void SetConfigurationSourceTheme()
        {
            switch (((FrameworkElement)this.Content).ActualTheme)
            {
                case ElementTheme.Dark: m_configurationSource.Theme =
Microsoft.UI.Composition.SystemBackdrops.SystemBackdropTheme.Dark; break;
                case ElementTheme.Light: m_configurationSource.Theme =
Microsoft.UI.Composition.SystemBackdrops.SystemBackdropTheme.Light; break;
                case ElementTheme.Default: m_configurationSource.Theme =
Microsoft.UI.Composition.SystemBackdrops.SystemBackdropTheme.Default; break;
            }
        }
}
```

```
class WindowsSystemDispatcherQueueHelper
{
    [StructLayout(LayoutKind.Sequential)]
    struct DispatcherQueueOptions
    {
        internal int dwSize;
        internal int threadType;
        internal int apartmentType;
    }

    [DllImport("CoreMessaging.dll")]
    private static extern int CreateDispatcherQueueController([In]
DispatcherQueueOptions options, [In, Out, MarshalAs(UnmanagedType.IUnknown)]
ref object dispatcherQueueController);

    object m_dispatcherQueueController = null;
    public void EnsureWindowsSystemDispatcherQueueController()
    {
        if (Windows.System.DispatcherQueue.GetForCurrentThread() != null)
        {
            // one already exists, so we'll just use it.
            return;
        }

        if (m_dispatcherQueueController == null)
        {
            DispatcherQueueOptions options;
            options.dwSize = Marshal.SizeOf(typeof(DispatcherQueueOptions));
            options.threadType = 2;    // DQTYPE_THREAD_CURRENT
            options.apartmentType = 2; // DQTAT_COM_STA

            CreateDispatcherQueueController(options, ref
m_dispatcherQueueController);
        }
    }
}
```

Related articles

- Materials in Windows 11
- Mica
- Acrylic
- Apply Mica in Win32 desktop apps for Windows 11

XAML custom panels overview

Article • 04/06/2022

A *panel* is an object that provides a layout behavior for child elements it contains, when the Extensible Application Markup Language (XAML) layout system runs and your app UI is rendered.

> **Important APIs**: Panel, ArrangeOverride, MeasureOverride

You can define custom panels for XAML layout by deriving a custom class from the Panel class. You provide behavior for your panel by overriding the MeasureOverride and ArrangeOverride, supplying logic that measures and arranges the child elements.

The Panel base class

To define a custom panel class, you can either derive from the Panel class directly, or derive from one of the practical panel classes that aren't sealed, such as Grid or StackPanel. It's easier to derive from **Panel**, because it can be difficult to work around the existing layout logic of a panel that already has layout behavior. Also, a panel with behavior might have existing properties that aren't relevant for your panel's layout features.

From Panel, your custom panel inherits these APIs:

- The Children property.
- The Background, ChildrenTransitions and IsItemsHost properties, and the dependency property identifiers. None of these properties are virtual, so you don't typically override or replace them. You don't typically need these properties for custom panel scenarios, not even for reading values.
- The layout override methods MeasureOverride and ArrangeOverride. These were originally defined by FrameworkElement. The base Panel class doesn't override these, but practical panels like Grid do have override implementations that are implemented as native code and are run by the system. Providing new (or additive) implementations for **ArrangeOverride** and **MeasureOverride** is the bulk of the effort you need to define a custom panel.
- All the other APIs of FrameworkElement, UIElement and DependencyObject, such as Height, Visibility and so on. You sometimes reference values of these properties in your layout overrides, but they aren't virtual so you don't typically override or replace them.

This focus here is to describe XAML layout concepts, so you can consider all the possibilities for how a custom panel can and should behave in layout. If you'd rather jump right in and see an example custom panel implementation, see BoxPanel, an example custom panel.

The Children property

The **Children** property is relevant to a custom panel because all classes derived from **Panel** use the **Children** property as the place to store their contained child elements in a collection. **Children** is designated as the XAML content property for the **Panel** class, and all classes derived from **Panel** can inherit the XAML content property behavior. If a property is designated the XAML content property, that means that XAML markup can omit a property element when specifying that property in markup, and the values are set as immediate markup children (the "content"). For example, if you derive a class named **CustomPanel** from **Panel** that defines no new behavior, you can still use this markup:

XAML

```
<local:CustomPanel>
  <Button Name="button1"/>
  <Button Name="button2"/>
</local:CustomPanel>
```

When a XAML parser reads this markup, **Children** is known to be the XAML content property for all **Panel** derived types, so the parser will add the two **Button** elements to the **UIElementCollection** value of the **Children** property. The XAML content property facilitates a streamlined parent-child relationship in the XAML markup for a UI definition. For more info about XAML content properties, and how collection properties are populated when XAML is parsed, see the XAML syntax guide.

The collection type that's maintaining the value of the **Children** property is the **UIElementCollection** class. **UIElementCollection** is a strongly typed collection that uses **UIElement** as its enforced item type. **UIElement** is a base type that's inherited by hundreds of practical UI element types, so the type enforcement here is deliberately loose. But it does enforce that you couldn't have a **Brush** as a direct child of a **Panel**, and it generally means that only elements that are expected to be visible in UI and participate in layout will be found as child elements in a **Panel**.

Typically, a custom panel accepts any **UIElement** child element by a XAML definition, by simply using the characteristics of the **Children** property as-is. As an advanced scenario,

you could support further type checking of child elements, when you iterate over the collection in your layout overrides.

Besides looping through the **Children** collection in the overrides, your panel logic might also be influenced by `Children.Count`. You might have logic that is allocating space at least partly based on the number of items, rather than desired sizes and the other characteristics of individual items.

Overriding the layout methods

The basic model for the layout override methods (**MeasureOverride** and **ArrangeOverride**) is that they should iterate through all the children and call each child element's specific layout method. The first layout cycle starts when the XAML layout system sets the visual for the root window. Because each parent invokes layout on its children, this propagates a call to layout methods to every possible UI element that is supposed to be part of a layout. In XAML layout, there are two stages: measure, then arrange.

You don't get any built-in layout method behavior for **MeasureOverride** and **ArrangeOverride** from the base **Panel** class. Items in **Children** won't automatically render as part of the XAML visual tree. It is up to you to make the items known to the layout process, by invoking layout methods on each of the items you find in **Children** through a layout pass within your **MeasureOverride** and **ArrangeOverride** implementations.

There's no reason to call base implementations in layout overrides unless you have your own inheritance. The native methods for layout behavior (if they exist) run regardless, and not calling base implementation from overrides won't prevent the native behavior from happening.

During the measure pass, your layout logic queries each child element for its desired size, by calling the **Measure** method on that child element. Calling the **Measure** method establishes the value for the **DesiredSize** property. The **MeasureOverride** return value is the desired size for the panel itself.

During the arrange pass, the positions and sizes of child elements are determined in x-y space and the layout composition is prepared for rendering. Your code must call **Arrange** on each child element in **Children** so that the layout system detects that the element belongs in the layout. The **Arrange** call is a precursor to composition and rendering; it informs the layout system where that element goes, when the composition is submitted for rendering.

Many properties and values contribute to how the layout logic will work at runtime. A way to think about the layout process is that the elements with no children (generally the most deeply nested element in the UI) are the ones that can finalize measurements first. They don't have any dependencies on child elements that influence their desired size. They might have their own desired sizes, and these are size suggestions until the layout actually takes place. Then, the measure pass continues walking up the visual tree until the root element has its measurements and all the measurements can be finalized.

The candidate layout must fit within the current app window or else parts of the UI will be clipped. Panels often are the place where the clipping logic is determined. Panel logic can determine what size is available from within the MeasureOverride implementation, and may have to push the size restrictions onto the children and divide space amongst children so that everything fits as best it can. The result of layout is ideally something that uses various properties of all parts of the layout but still fits within the app window. That requires both a good implementation for layout logic of the panels, and also a judicious UI design on the part of any app code that builds a UI using that panel. No panel design is going to look good if the overall UI design includes more child elements than can possibly fit in the app.

A large part of what makes the layout system work is that any element that's based on FrameworkElement already has some of its own inherent behavior when acting as a child in a container. For example, there are several APIs of **FrameworkElement** that either inform layout behavior or are needed to make layout work at all. These include:

- DesiredSize (actually a UIElement property)
- ActualHeight and ActualWidth
- Height and Width
- Margin
- LayoutUpdated event
- HorizontalAlignment and VerticalAlignment
- ArrangeOverride and MeasureOverride methods
- Arrange and Measure methods: these have native implementations defined at the FrameworkElement level, which handle the element-level layout action

MeasureOverride

The MeasureOverride method has a return value that's used by the layout system as the starting DesiredSize for the panel itself, when the Measure method is called on the panel by its parent in layout. The logic choices within the method are just as important as what it returns, and the logic often influences what value is returned.

All MeasureOverride implementations should loop through Children, and call the Measure method on each child element. Calling the **Measure** method establishes the value for the DesiredSize property. This might inform how much space the panel itself needs, as well as how that space is divided among elements or sized for a particular child element.

Here's a very basic skeleton of a MeasureOverride method:

CSharp

```
protected override Size MeasureOverride(Size availableSize)
{
    Size returnSize; //TODO might return availableSize, might do something
else

    //loop through each Child, call Measure on each
    foreach (UIElement child in Children)
    {
        child.Measure(new Size()); // TODO determine how much space the
panel allots for this child, that's what you pass to Measure
        Size childDesiredSize = child.DesiredSize; //TODO determine how the
returned Size is influenced by each child's DesiredSize
        //TODO, logic if passed-in Size and net DesiredSize are different,
does that matter?
    }
    return returnSize;
}
```

Elements often have a natural size by the time they're ready for layout. After the measure pass, the DesiredSize might indicate that natural size, if the *availableSize* you passed for Measure was smaller. If the natural size is larger than *availableSize* you passed for **Measure**, the **DesiredSize** is constrained to *availableSize*. That's how **Measure**'s internal implementation behaves, and your layout overrides should take that behavior into account.

Some elements don't have a natural size because they have **Auto** values for Height and Width. These elements use the full *availableSize*, because that's what an **Auto** value represents: size the element to the maximum available size, which the immediate layout parent communicates by calling Measure with *availableSize*. In practice, there's always some measurement that a UI is sized to (even if that's the top level window.) Eventually, the measure pass resolves all the **Auto** values to parent constraints and all **Auto** value elements get real measurements (which you can get by checking ActualWidth and ActualHeight, after layout completes).

It's legal to pass a size to Measure that has at least one infinite dimension, to indicate that the panel can attempt to size itself to fit measurements of its content. Each child

element being measured sets its DesiredSize value using its natural size. Then, during the arrange pass, the panel typically arranges using that size.

Text elements such as TextBlock have a calculated ActualWidth and ActualHeight based on their text string and text properties even if no Height or Width value is set, and these dimensions should be respected by your panel logic. Clipping text is a particularly bad UI experience.

Even if your implementation doesn't use the desired size measurements, it's best to call the Measure method on each child element, because there are internal and native behaviors that are triggered by **Measure** being called. For an element to participate in layout, each child element must have **Measure** called on it during the measure pass and the Arrange method called on it during the arrange pass. Calling these methods sets internal flags on the object and populates values (such as the DesiredSize property) that the system's layout logic needs when it builds the visual tree and renders the UI.

The MeasureOverride return value is based on the panel's logic interpreting the DesiredSize or other size considerations for each of the child elements in Children when Measure is called on them. What to do with **DesiredSize** values from children and how the **MeasureOverride** return value should use them is up to your own logic's interpretation. You don't typically add up the values without modification, because the input of **MeasureOverride** is often a fixed available size that's being suggested by the panel's parent. If you exceed that size, the panel itself might get clipped. You'd typically compare the total size of children to the panel's available size and make adjustments if necessary.

Tips and guidance

- Ideally, a custom panel should be suitable for being the first true visual in a UI composition, perhaps at a level immediately under Page, UserControl or another element that is the XAML page root. In MeasureOverride implementations, don't routinely return the input Size without examining the values. If the return **Size** has an **Infinity** value in it, this can throw exceptions in runtime layout logic. An **Infinity** value can come from the main app window, which is scrollable and therefore doesn't have a maximum height. Other scrollable content might have the same behavior.
- Another common mistake in MeasureOverride implementations is to return a new default Size (values for height and width are 0). You might start with that value, and it might even be the correct value if your panel determines that none of the children should be rendered. But, a default **Size** results in your panel not being sized correctly by its host. It requests no space in the UI, and therefore gets no space and doesn't render. All your panel code otherwise might be functioning fine,

but you still won't see your panel or contents thereof if it's being composed with zero height, zero width.

- Within the overrides, avoid the temptation to cast child elements to FrameworkElement and use properties that are calculated as a result of layout, particularly ActualWidth and ActualHeight. For most common scenarios, you can base the logic on the child's DesiredSize value and you won't need any of the Height or Width related properties of a child element. For specialized cases, where you know the type of element and have additional information, for example the natural size of an image file, you can use your element's specialized information because it's not a value that is actively being altered by layout systems. Including layout-calculated properties as part of layout logic substantially increases the risk of defining an unintentional layout loop. These loops cause a condition where a valid layout can't be created and the system can throw a LayoutCycleException if the loop is not recoverable.
- Panels typically divide their available space between multiple child elements, although exactly how space is divided varies. For example, Grid implements layout logic that uses its RowDefinition and ColumnDefinition values to divide the space into the **Grid** cells, supporting both star-sizing and pixel values. If they're pixel values, the size available for each child is already known, so that's what is passed as input size for a grid-style Measure.
- Panels themselves can introduce reserved space for padding between items. If you do this, make sure to expose the measurements as a property that's distinct from Margin or any **Padding** property.
- Elements might have values for their ActualWidth and ActualHeight properties based on a previous layout pass. If values change, app UI code can put handlers for LayoutUpdated on elements if there's special logic to run, but panel logic typically doesn't need to check for changes with event handling. The layout system is already making the determinations of when to re-run layout because a layout-relevant property changed value, and a panel's MeasureOverride or ArrangeOverride are called automatically in the appropriate circumstances.

ArrangeOverride

The ArrangeOverride method has a Size return value that's used by the layout system when rendering the panel itself, when the Arrange method is called on the panel by its parent in layout. It's typical that the input *finalSize* and the **ArrangeOverride** returned **Size** are the same. If they aren't, that means the panel is attempting to make itself a different size than what the other participants in layout claim is available. The final size was based on having previously run the measure pass of layout through your panel

code, so that's why returning a different size isn't typical: it means you are deliberately ignoring measure logic.

Don't return a Size with an **Infinity** component. Trying to use such a **Size** throws an exception from internal layout.

All ArrangeOverride implementations should loop through Children, and call the Arrange method on each child element. Like Measure, **Arrange** doesn't have a return value. Unlike **Measure**, no calculated property gets set as a result (however, the element in question typically fires a LayoutUpdated event).

Here's a very basic skeleton of an ArrangeOverride method:

CSharp

```
protected override Size ArrangeOverride(Size finalSize)
{
    //loop through each Child, call Arrange on each
    foreach (UIElement child in Children)
    {
        Point anchorPoint = new Point(); //TODO more logic for topleft
corner placement in your panel
       // for this child, and based on finalSize or other internal state of
your panel
        child.Arrange(new Rect(anchorPoint, child.DesiredSize)); //OR, set a
different Size
    }
    return finalSize; //OR, return a different Size, but that's rare
}
```

The arrange pass of layout might happen without being preceded by a measure pass. However, this only happens when the layout system has determined no properties have changed that would have affected the previous measurements. For example, if an alignment changes, there's no need to re-measure that particular element because its DesiredSize would not change when its alignment choice changes. On the other hand, if ActualHeight changes on any element in a layout, a new measure pass is needed. The layout system automatically detects true measure changes and invokes the measure pass again, and then runs another arrange pass.

The input for Arrange takes a Rect value. The most common way to construct this **Rect** is to use the constructor that has a Point input and a Size input. The **Point** is the point where the top left corner of the bounding box for the element should be placed. The **Size** is the dimensions used to render that particular element. You often use the DesiredSize for that element as this **Size** value, because establishing the **DesiredSize** for all elements involved in layout was the purpose of the measure pass of layout. (The

measure pass determines all-up sizing of the elements in an iterative way so that the layout system can optimize how elements are placed once it gets to the arrange pass.)

What typically varies between **ArrangeOverride** implementations is the logic by which the panel determines the **Point** component of how it arranges each child. An absolute positioning panel such as **Canvas** uses the explicit placement info that it gets from each element through **Canvas.Left** and **Canvas.Top** values. A space-dividing panel such as **Grid** would have mathematical operations that divided the available space into cells and each cell would have an x-y value for where its content should be placed and arranged. An adaptive panel such as **StackPanel** might be expanding itself to fit content in its orientation dimension.

There are still additional positioning influences on elements in layout, beyond what you directly control and pass to **Arrange**. These come from the internal native implementation of **Arrange** that's common to all **FrameworkElement** derived types and augmented by some other types such as text elements. For example, elements can have margin and alignment, and some can have padding. These properties often interact. For more info, see Alignment, margin, and padding.

Panels and controls

Avoid putting functionality into a custom panel that should instead be built as a custom control. The role of a panel is to present any child element content that exists within it, as a function of layout that happens automatically. The panel might add decorations to content (similar to how a **Border** adds the border around the element it presents), or perform other layout-related adjustments like padding. But that's about as far as you should go when extending the visual tree output beyond reporting and using information from the children.

If there's any interaction that's accessible to the user, you should write a custom control, not a panel. For example, a panel shouldn't add scrolling viewports to content it presents, even if the goal is to prevent clipping, because the scrollbars, thumbs and so on are interactive control parts. (Content might have scrollbars after all, but you should leave that up to the child's logic. Don't force it by adding scrolling as a layout operation.) You might create a control and also write a custom panel that plays an important role in that control's visual tree, when it comes to presenting content in that control. But the control and the panel should be distinct code objects.

One reason the distinction between control and panel is important is because of Microsoft UI Automation and accessibility. Panels provide a visual layout behavior, not a logical behavior. How a UI element appears visually is not an aspect of UI that is typically important to accessibility scenarios. Accessibility is about exposing the parts of an app

that are logically important to understanding a UI. When interaction is required, controls should expose the interaction possibilities to the UI Automation infrastructure. For more info, see Custom automation peers.

Other layout API

There are some other APIs that are part of the layout system, but aren't declared by **Panel**. You might use these in a panel implementation or in a custom control that uses panels.

- **UpdateLayout**, **InvalidateMeasure**, and **InvalidateArrange** are methods that initiate a layout pass. **InvalidateArrange** might not trigger a measure pass, but the other two do. Never call these methods from within a layout method override, because they're almost sure to cause a layout loop. Control code doesn't typically need to call them either. Most aspects of layout are triggered automatically by detecting changes to the framework-defined layout properties such as **Width** and so on.
- **LayoutUpdated** is an event that fires when some aspect of layout of the element has changed. This isn't specific to panels; the event is defined by **FrameworkElement**.
- **SizeChanged** is an event that fires only after layout passes are finalized, and indicates that **ActualHeight** or **ActualWidth** have changed as a result. This is another **FrameworkElement** event. There are cases where **LayoutUpdated** fires, but **SizeChanged** does not. For example the internal contents might be rearranged, but the element's size didn't change.

Related topics

Reference

- **FrameworkElement.ArrangeOverride**
- **FrameworkElement.MeasureOverride**
- **Panel**

Concepts

- Alignment, margin, and padding

BoxPanel, an example custom panel

Article • 07/13/2022

Learn to write code for a custom **Panel** class, implementing **ArrangeOverride** and **MeasureOverride** methods, and using the **Children** property.

> **Important APIs**: **Panel**, **ArrangeOverride**,**MeasureOverride**

The example code shows a custom panel implementation, but we don't devote a lot of time explaining the layout concepts that influence how you can customize a panel for different layout scenarios. If you want more info about these layout concepts and how they might apply to your particular layout scenario, see XAML custom panels overview.

A *panel* is an object that provides a layout behavior for child elements it contains, when the XAML layout system runs and your app UI is rendered. You can define custom panels for XAML layout by deriving a custom class from the **Panel** class. You provide behavior for your panel by overriding the **ArrangeOverride** and **MeasureOverride** methods, supplying logic that measures and arranges the child elements. This example derives from **Panel**. When you start from **Panel**, **ArrangeOverride** and **MeasureOverride** methods don't have a starting behavior. Your code is providing the gateway by which child elements become known to the XAML layout system and get rendered in the UI. So, it's really important that your code accounts for all child elements and follows the patterns the layout system expects.

Your layout scenario

When you define a custom panel, you're defining a layout scenario.

A layout scenario is expressed through:

- What the panel will do when it has child elements
- When the panel has constraints on its own space
- How the logic of the panel determines all the measurements, placement, positions, and sizings that eventually result in a rendered UI layout of children

With that in mind, the `BoxPanel` shown here is for a particular scenario. In the interest of keeping the code foremost in this example, we won't explain the scenario in detail yet, and instead concentrate on the steps needed and the coding patterns. If you want to know more about the scenario first, skip ahead to "The scenario for BoxPanel", and then come back to the code.

Start by deriving from Panel

Start by deriving a custom class from **Panel**. Probably the easiest way to do this is to define a separate code file for this class, using the **Add** | **New Item** | **Class** context menu options for a project from the **Solution Explorer** in Microsoft Visual Studio. Name the class (and file) `BoxPanel`.

The template file for a class doesn't start with many **using** statements because it's not specifically for Windows apps. So first, add **using** statements. The template file also starts with a few **using** statements that you probably don't need, and can be deleted. Here's a suggested list of **using** statements that can resolve types you'll need for typical custom panel code:

CSharp

```
using System;
using System.Collections.Generic; // if you need to cast IEnumerable for
iteration, or define your own collection properties
using Windows.Foundation; // Point, Size, and Rect
using Windows.UI.Xaml; // DependencyObject, UIElement, and FrameworkElement
using Windows.UI.Xaml.Controls; // Panel
using Windows.UI.Xaml.Media; // if you need Brushes or other utilities
```

Now that you can resolve **Panel**, make it the base class of `BoxPanel`. Also, make `BoxPanel` public:

CSharp

```
public class BoxPanel : Panel
{
}
```

At the class level, define some **int** and **double** values that will be shared by several of your logic functions, but which won't need to be exposed as public API. In the example, these are named: `maxrc`, `rowcount`, `colcount`, `cellwidth`, `cellheight`, `maxcellheight`, `aspectratio`.

After you've done this, the complete code file looks like this (removing comments on **using**, now that you know why we have them):

CSharp

```
using System;
using System.Collections.Generic;
using Windows.Foundation;
```

```
using Windows.UI.Xaml;
using Windows.UI.Xaml.Controls;
using Windows.UI.Xaml.Media;

public class BoxPanel : Panel
{
    int maxrc, rowcount, colcount;
    double cellwidth, cellheight, maxcellheight, aspectratio;
}
```

From here on out, we'll be showing you one member definition at a time, be that a method override or something supporting such as a dependency property. You can add these to the skeleton above in any order.

MeasureOverride

CSharp

```
protected override Size MeasureOverride(Size availableSize)
{
    // Determine the square that can contain this number of items.
    maxrc = (int)Math.Ceiling(Math.Sqrt(Children.Count));
    // Get an aspect ratio from availableSize, decides whether to trim row
or column.
    aspectratio = availableSize.Width / availableSize.Height;

    // Now trim this square down to a rect, many times an entire row or
column can be omitted.
    if (aspectratio > 1)
    {
        rowcount = maxrc;
        colcount = (maxrc > 2 && Children.Count <= maxrc * (maxrc - 1)) ?
maxrc - 1 : maxrc;
    }
    else
    {
        rowcount = (maxrc > 2 && Children.Count <= maxrc * (maxrc - 1)) ?
maxrc - 1 : maxrc;
        colcount = maxrc;
    }

    // Now that we have a column count, divide available horizontal, that's
our cell width.
    cellwidth = (int)Math.Floor(availableSize.Width / colcount);
    // Next get a cell height, same logic of dividing available vertical by
rowcount.
    cellheight = Double.IsInfinity(availableSize.Height) ?
Double.PositiveInfinity : availableSize.Height / rowcount;

    foreach (UIElement child in Children)
    {
```

```
            child.Measure(new Size(cellwidth, cellheight));
            maxcellheight = (child.DesiredSize.Height > maxcellheight) ?
child.DesiredSize.Height : maxcellheight;
        }
        return LimitUnboundedSize(availableSize);
}
```

The necessary pattern of a MeasureOverride implementation is the loop through each element in Panel.Children. Always call the Measure method on each of these elements. **Measure** has a parameter of type Size. What you're passing here is the size that your panel is committing to have available for that particular child element. So, before you can do the loop and start calling **Measure**, you need to know how much space each cell can devote. From the **MeasureOverride** method itself, you have the *availableSize* value. That is the size that the panel's parent used when it called **Measure**, which was the trigger for this **MeasureOverride** being called in the first place. So a typical logic is to devise a scheme whereby each child element divides the space of the panel's overall *availableSize*. You then pass each division of size to **Measure** of each child element.

How `BoxPanel` divides size is fairly simple: it divides its space into a number of boxes that's largely controlled by the number of items. Boxes are sized based on row and column count and the available size. Sometimes one row or column from a square isn't needed, so it's dropped and the panel becomes a rectangle rather than square in terms of its row : column ratio. For more info about how this logic was arrived at, skip ahead to "The scenario for BoxPanel".

So what does the measure pass do? It sets a value for the read-only DesiredSize property on each element where Measure was called. Having a **DesiredSize** value is possibly important once you get to the arrange pass, because the **DesiredSize** communicates what the size can or should be when arranging and in the final rendering. Even if you don't use **DesiredSize** in your own logic, the system still needs it.

It's possible for this panel to be used when the height component of *availableSize* is unbounded. If that's true, the panel doesn't have a known height to divide. In this case, the logic for the measure pass informs each child that it doesn't have a bounded height, yet. It does so by passing a Size to the Measure call for children where Size.Height is infinite. That's legal. When **Measure** is called, the logic is that the DesiredSize is set as the minimum of these: what was passed to **Measure**, or that element's natural size from factors such as explicitly-set Height and Width.

> ⓘ **Note**
>
> The internal logic of StackPanel also has this behavior: **StackPanel** passes an infinite dimension value to Measure on children, indicating that there is no constraint on

children in the orientation dimension. **StackPanel** typically sizes itself dynamically, to accommodate all children in a stack that grows in that dimension.

However, the panel itself can't return a **Size** with an infinite value from **MeasureOverride**; that throws an exception during layout. So, part of the logic is to find out the maximum height that any child requests, and use that height as the cell height in case that isn't coming from the panel's own size constraints already. Here's the helper function `LimitUnboundedSize` that was referenced in previous code, which then takes that maximum cell height and uses it to give the panel a finite height to return, as well as assuring that `cellheight` is a finite number before the arrange pass is initiated:

CSharp

```
// This method limits the panel height when no limit is imposed by the
panel's parent.
// That can happen to height if the panel is close to the root of main app
window.
// In this case, base the height of a cell on the max height from desired
size
// and base the height of the panel on that number times the #rows.
Size LimitUnboundedSize(Size input)
{
    if (Double.IsInfinity(input.Height))
    {
        input.Height = maxcellheight * colcount;
        cellheight = maxcellheight;
    }
    return input;
}
```

ArrangeOverride

CSharp

```
protected override Size ArrangeOverride(Size finalSize)
{
     int count = 1;
     double x, y;
     foreach (UIElement child in Children)
     {
          x = (count - 1) % colcount * cellwidth;
          y = ((int)(count - 1) / colcount) * cellheight;
          Point anchorPoint = new Point(x, y);
          child.Arrange(new Rect(anchorPoint, child.DesiredSize));
          count++;
     }
```

```
        return finalSize;
    }
```

The necessary pattern of an ArrangeOverride implementation is the loop through each element in Panel.Children. Always call the Arrange method on each of these elements.

Note how there aren't as many calculations as in MeasureOverride; that's typical. The size of children is already known from the panel's own **MeasureOverride** logic, or from the DesiredSize value of each child set during the measure pass. However, we still need to decide the location within the panel where each child will appear. In a typical panel, each child should render at a different position. A panel that creates overlapping elements isn't desirable for typical scenarios (although it's not out of the question to create panels that have purposeful overlaps, if that's really your intended scenario).

This panel arranges by the concept of rows and columns. The number of rows and columns was already calculated (it was necessary for measurement). So now the shape of the rows and columns plus the known sizes of each cell contribute to the logic of defining a rendering position (the `anchorPoint`) for each element that this panel contains. That Point, along with the Size already known from measure, are used as the two components that construct a Rect. **Rect** is the input type for Arrange.

Panels sometimes need to clip their content. If they do, the clipped size is the size that's present in DesiredSize, because the Measure logic sets it as the minimum of what was passed to **Measure**, or other natural size factors. So you don't typically need to specifically check for clipping during Arrange; the clipping just happens based on passing the **DesiredSize** through to each **Arrange** call.

You don't always need a count while going through the loop if all the info you need for defining the rendering position is known by other means. For example, in Canvas layout logic, the position in the Children collection doesn't matter. All the info needed to position each element in a **Canvas** is known by reading Canvas.Left and Canvas.Top values of children as part of the arrange logic. The `BoxPanel` logic happens to need a count to compare to the *colcount* so it's known when to begin a new row and offset the *y* value.

It's typical that the input *finalSize* and the Size you return from a ArrangeOverride implementation are the same. For more info about why, see "**ArrangeOverride**" section of XAML custom panels overview.

A refinement: controlling the row vs. column count

You could compile and use this panel just as it is now. However, we'll add one more refinement. In the code just shown, the logic puts the extra row or column on the side that's longest in aspect ratio. But for greater control over the shapes of cells, it might be desirable to choose a 4x3 set of cells instead of 3x4 even if the panel's own aspect ratio is "portrait." So we'll add an optional dependency property that the panel consumer can set to control that behavior. Here's the dependency property definition, which is very basic:

CSharp

```
// Property
public Orientation Orientation
{
    get { return (Orientation)GetValue(OrientationProperty); }
    set { SetValue(OrientationProperty, value); }
}

// Dependency Property Registration
public static readonly DependencyProperty OrientationProperty =
        DependencyProperty.Register(nameof(Orientation),
typeof(Orientation), typeof(BoxPanel), new PropertyMetadata(null,
OnOrientationChanged));

// Changed callback so we invalidate our layout when the property changes.
private static void OnOrientationChanged(DependencyObject dependencyObject,
DependencyPropertyChangedEventArgs args)
{
    if (dependencyObject is BoxPanel panel)
    {
        panel.InvalidateMeasure();
    }
}
```

And below is how using `Orientation` impacts the measure logic in `MeasureOverride`. Really all it's doing is changing how `rowcount` and `colcount` are derived from `maxrc` and the true aspect ratio, and there are corresponding size differences for each cell because of that. When `Orientation` is **Vertical** (default), it inverts the value of the true aspect ratio before using it for row and column counts for our "portrait" rectangle layout.

CSharp

```
// Get an aspect ratio from availableSize, decides whether to trim row or
column.
aspectratio = availableSize.Width / availableSize.Height;

// Transpose aspect ratio based on Orientation property.
if (Orientation == Orientation.Vertical) { aspectratio = 1 / aspectratio; }
```

The scenario for BoxPanel

The particular scenario for `BoxPanel` is that it's a panel where one of the main determinants of how to divide space is by knowing the number of child items, and dividing the known available space for the panel. Panels are innately rectangle shapes. Many panels operate by dividing that rectangle space into further rectangles; that's what **Grid** does for its cells. In **Grid**'s case, the size of the cells is set by **ColumnDefinition** and **RowDefinition** values, and elements declare the exact cell they go into with **Grid.Row** and **Grid.Column** attached properties. Getting good layout from a **Grid** usually requires knowing the number of child elements beforehand, so that there are enough cells and each child element sets its attached properties to fit into its own cell.

But what if the number of children is dynamic? That's certainly possible; your app code can add items to collections, in response to any dynamic run-time condition you consider to be important enough to be worth updating your UI. If you're using data binding to backing collections/business objects, getting such updates and updating the UI is handled automatically, so that's often the preferred technique (see Data binding in depth).

But not all app scenarios lend themselves to data binding. Sometimes, you need to create new UI elements at runtime and make them visible. `BoxPanel` is for this scenario. A changing number of child items is no problem for `BoxPanel` because it's using the child count in calculations, and adjusts both the existing and new child elements into a new layout so they all fit.

An advanced scenario for extending `BoxPanel` further (not shown here) could both accommodate dynamic children and use a child's **DesiredSize** as a stronger factor for the sizing of individual cells. This scenario might use varying row or column sizes or non-grid shapes so that there's less "wasted" space. This requires a strategy for how multiple rectangles of various sizes and aspect ratios can all fit into a containing rectangle both for aesthetics and smallest size. `BoxPanel` doesn't do that; it's using a simpler technique for dividing space. `BoxPanel`'s technique is to determine the least square number that's greater than the child count. For example, 9 items would fit in a 3x3 square. 10 items require a 4x4 square. However, you can often fit items while still removing one row or column of the starting square, to save space. In the count=10 example, that fits in a 4x3 or 3x4 rectangle.

You might wonder why the panel wouldn't instead choose 5x2 for 10 items, because that fits the item number neatly. However, in practice, panels are sized as rectangles that seldom have a strongly oriented aspect ratio. The least-squares technique is a way to

bias the sizing logic to work well with typical layout shapes and not encourage sizing where the cell shapes get odd aspect ratios.

Related topics

Reference

- **FrameworkElement.ArrangeOverride**
- **FrameworkElement.MeasureOverride**
- **Panel**

Concepts

- Alignment, margin, and padding

Attached Layouts

Article • 05/26/2022

A container (for example, Panel) that delegates its layout logic to another object relies on the attached layout object to provide the layout behavior for its child elements. An attached layout model provides flexibility for an application to alter the layout of items at runtime, or more easily share aspects of layout between different parts of the UI (for example, items in the rows of a table that appear to be aligned within a column).

In this topic, we cover what's involved in creating an attached layout (virtualizing and non-virtualizing), the concepts and classes you'll need to understand, and the trade-offs you'll need to consider when deciding between them.

Expand table

Get WinUI
This control is included as part of WinUI, a NuGet package that contains new controls and UI features for Windows apps. For more info, including installation instructions, see the WinUI overview.

> **Important APIs**:
>
> - ScrollViewer
> - ItemsRepeater
> - Layout
>   - NonVirtualizingLayout
>   - VirtualizingLayout
> - LayoutContext
>   - NonVirtualizingLayoutContext
>   - VirtualizingLayoutContext
> - LayoutPanel (Preview)

Key Concepts

Performing layout requires that two questions be answered for every element:

1. What ***size*** will this element be?
2. What will the ***position*** of this element be?

XAML's layout system, which answers these questions, is briefly covered as part of the discussion of Custom panels.

Containers and Context

Conceptually, XAML's Panel fills two important roles in the framework:

1. It can contain child elements and introduces branching in the tree of elements.
2. It applies a specific layout strategy to those children.

For this reason, a Panel in XAML has often been synonymous with layout, but technically-speaking, does more than just layout.

The ItemsRepeater also behaves like Panel, but, unlike Panel, it does not expose a Children property that would allow programmatically adding or removing UIElement children. Instead, the lifetime of its children are automatically managed by the framework to correspond to a collection of data items. Although it is not derived from Panel, it behaves and is treated by the framework like a Panel.

> **Note**
>
> The **LayoutPanel** is a container, derived from Panel, that delegates its logic to the attached **Layout** object. LayoutPanel is in *Preview* and is currently available only in the *Prerelease* drops of the WinUI package.

Containers

Conceptually, Panel is a container of elements that also has the ability to render pixels for a Background. Panels provide a way to encapsulate common layout logic in an easy to use package.

The concept of **attached layout** makes the distinction between the two roles of container and layout more clear. If the container delegates its layout logic to another object we would call that object the attached layout as seen in the snippet below. Containers that inherit from FrameworkElement, such as the LayoutPanel, automatically expose the common properties that provide input to XAML's layout process (for example, Height and Width).

XAML

```
<LayoutPanel>
    <LayoutPanel.Layout>
        <UniformGridLayout/>
```

```
    </LayoutPanel.Layout>
    <Button Content="1"/>
    <Button Content="2"/>
    <Button Content="3"/>
</LayoutPanel>
```

During the layout process the container relies on the attached *UniformGridLayout* to measure and arrange its children.

Per-Container State

With an attached layout, a single instance of the layout object may be associated with *many* containers like in the snippet below; therefore, it must not depend on or directly reference the host container. For example:

XAML

```
<!-- ... --->
<Page.Resources>
    <ExampleLayout x:Name="exampleLayout"/>
<Page.Resources>

<LayoutPanel x:Name="example1" Layout="{StaticResource exampleLayout}"/>
<LayoutPanel x:Name="example2" Layout="{StaticResource exampleLayout}"/>
<!-- ... --->
```

For this situation *ExampleLayout* must carefully consider the state that it uses in its layout calculation and where that state is stored to avoid impacting the layout for elements in one panel with the other. It would be analogous to a custom Panel whose MeasureOverride and ArrangeOverride logic depends on the values of its *static* properties.

LayoutContext

The purpose of the LayoutContext is to deal with those challenges. It provides the attached layout the ability to interact with the host container, such as retrieving child elements, without introducing a direct dependency between the two. The context also enables the layout to store any state it requires that might be related to the container's child elements.

Simple, non-virtualizing layouts often do not need to maintain any state, making it a non-issue. A more complex layout, such as Grid, however, may choose to maintain state between the measure and arrange call to avoid re-computing a value.

Virtualizing layouts *often* need to maintain some state between both the measure and arrange as well as between iterative layout passes.

Initializing and Uninitializing Per-Container State

When a layout is attached to a container, its InitializeForContextCore method is called and provides an opportunity to initialize an object to store state.

Similarly, when the layout is being removed from a container, the UninitializeForContextCore method will be called. This gives the layout an opportunity to clean up any state it had associated with that container.

The layout's state object can be stored with and retrieved from the container with the LayoutState property on the context.

UI Virtualization

UI virtualization means delaying the creation of a UI object until *when it's needed*. It's a performance optimization. For non-scrolling scenarios determining *when needed* may be based on any number of things that are app-specific. In those cases, apps should consider using the x:Load. It does not require any special handling in your layout.

In scrolling-based scenarios such as a list, determining *when needed* is often based on "will it be visible to a user" which depends heavily on where it was placed during the layout process and requires special considerations. This scenario is a focus for this document.

> ⓘ **Note**
>
> Although not covered in this document, the same capabilities that enable UI virtualization in scrolling scenarios could be applied in non-scrolling scenarios. For example, a data-driven ToolBar control that manages the lifetime of the commands it presents and responds to changes in available space by recycling / moving elements between a visible area and an overflow menu.

Getting Started

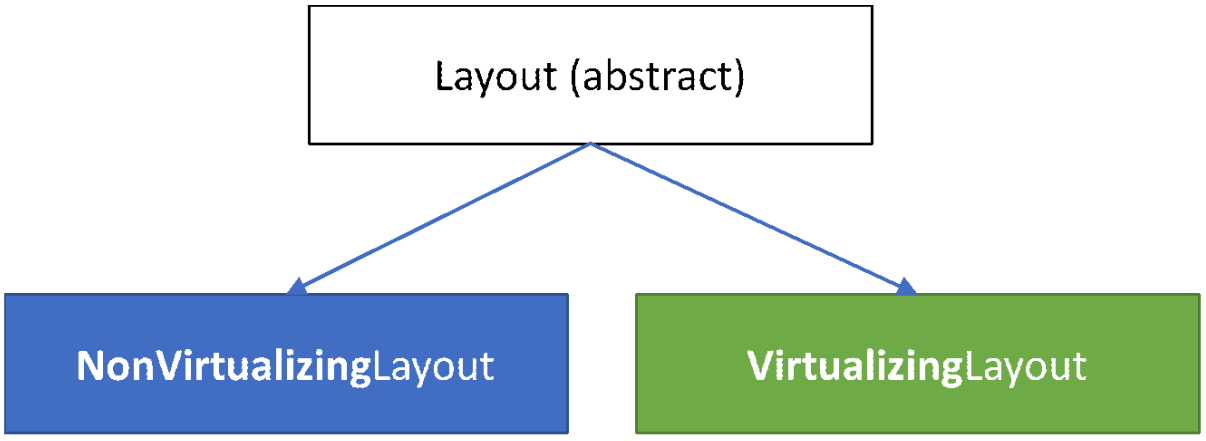
First, decide whether the layout you need to create should support UI virtualization.

A few things to keep in mind...

1. Non-virtualizing layouts are easier to author. If the number of items will always be small then authoring a non-virtualizing layout is recommended.
2. The platform provides a set of attached layouts that work with the ItemsRepeater and LayoutPanel to cover common needs. Familiarize yourself with those before deciding you need to define a custom layout.
3. Virtualizing layouts always have some additional CPU and memory cost/complexity/overhead compared to a non-virtualizing layout. As a general rule of thumb if the children the layout will need to manage will likely fit in an area that is 3x the size of the viewport, then there may not be much gain from a virtualizing layout. The 3x size is discussed in greater detail later in this doc, but is due to the asynchronous nature of scrolling on Windows and its impact on virtualization.

> **Tip**
>
> As a point of reference, the default settings for the **ListView** (and **ItemsRepeater**) are that recycling doesn't begin until the number of items are enough to fill up 3x the size of the current viewport.

Choose your base type



The base Layout type has two derived types that serve as the start point for authoring an attached layout:

1. NonVirtualizingLayout
2. VirtualizingLayout

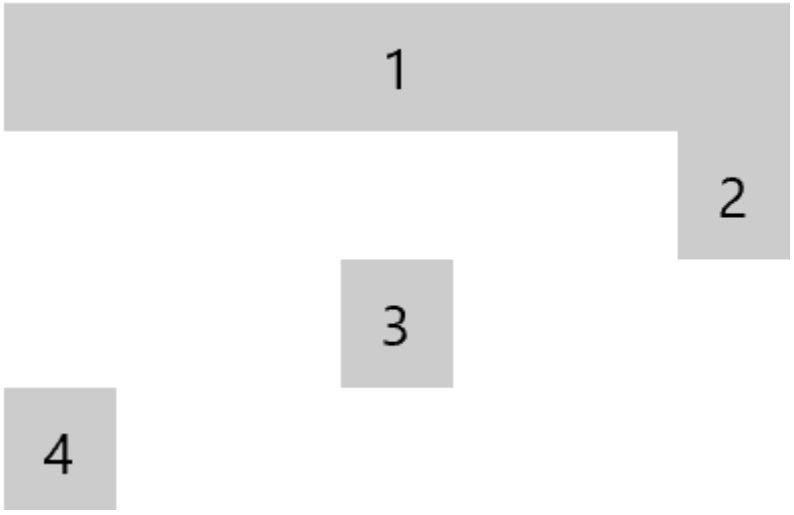
Non-Virtualizing Layout

The approach for creating a non-virtualizing layout should feel familiar to anyone that has created a Custom Panel. The same concepts apply. The primary difference is that a

NonVirtualizingLayoutContext is used to access the Children collection, and layout may choose to store state.

1. Derive from the base type NonVirtualizingLayout (instead of Panel).
2. *(Optional)* Define dependency properties that when changed will invalidate the layout.
3. ***(New**/Optional)* Initialize any state object required by the layout as part of the InitializeForContextCore. Stash it with the host container by using the LayoutState provided with the context.
4. Override the MeasureOverride and call the Measure method on all the children.
5. Override the ArrangeOverride and call the Arrange method on all the children.
6. ***(New**/Optional)* Clean up any saved state as part of the UninitializeForContextCore.

Example: A Simple Stack Layout (Varying-Sized Items)



Here is a very basic non-virtualizing stack layout of varying sized items. It lacks any properties to adjust the layout's behavior. The implementation below illustrates how the layout relies on the context object provided by the container to:

1. Get the count of children, and
2. Access each child element by index.

C#

```
public class MyStackLayout : NonVirtualizingLayout
{
    protected override Size MeasureOverride(NonVirtualizingLayoutContext
context, Size availableSize)
    {
        double extentHeight = 0.0;
        foreach (var element in context.Children)
        {
            element.Measure(availableSize);
```

```
            extentHeight += element.DesiredSize.Height;
        }

        return new Size(availableSize.Width, extentHeight);
    }

    protected override Size ArrangeOverride(NonVirtualizingLayoutContext
context, Size finalSize)
    {
        double offset = 0.0;
        foreach (var element in context.Children)
        {
            element.Arrange(
                new Rect(0, offset, finalSize.Width,
element.DesiredSize.Height));
            offset += element.DesiredSize.Height;
        }

        return finalSize;
    }
}
```

XAML

```
 <LayoutPanel MaxWidth="196">
    <LayoutPanel.Layout>
        <local:MyStackLayout/>
    </LayoutPanel.Layout>

    <Button HorizontalAlignment="Stretch">1</Button>
    <Button HorizontalAlignment="Right">2</Button>
    <Button HorizontalAlignment="Center">3</Button>
    <Button>4</Button>

</LayoutPanel>
```

Virtualizing Layouts

Similar to a non-virtualizing layout, the high-level steps for a virtualizing layout are the same. The complexity is largely in determining what elements will fall within the viewport and should be realized.

1. Derive from the base type VirtualizingLayout.
2. (Optional) Define your dependency properties that when changed will invalidate the layout.
3. Initialize any state object that will be required by the layout as part of the InitializeForContextCore. Stash it with the host container by using the LayoutState provided with the context.

4. Override the MeasureOverride and call the Measure method for each child that should be realized.
    a. The GetOrCreateElementAt method is used to retrieve a UIElement that has been prepared by the framework (for example, data bindings applied).
5. Override the ArrangeOverride and call the Arrange method for each realized child.
6. (Optional) Clean up any saved state as part of the UninitializeForContextCore.

> **Tip**
>
> The value returned by the **MeasureOverride** is used as the size of the virtualized content.

There are two general approaches to consider when authoring a virtualizing layout. Whether to choose one or the other largely depends on "how will you determine the size of an element". If its enough to know the index of an item in the data set or the data itself dictates its eventual size, then we'd consider it **data-dependent**. These are more straightforward to create. If, however, the only way to determine the size for an item is to create and measure the UI then we'd say it is **content-dependent**. These are more complex.

The Layout Process

Whether you're creating a data or content-dependent layout, it's important to understand the layout process and the impact of Windows' asynchronous scrolling.

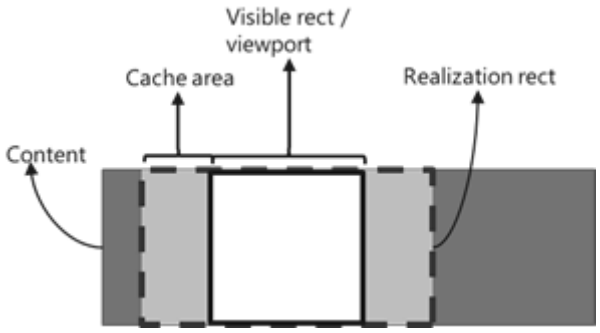
An (over)simplified view of the steps performed by the framework from start-up to displaying UI on screen is that:

1. It parses the markup.
2. Generates a tree of elements.
3. Performs a layout pass.
4. Performs a render pass.

With UI virtualization, creating the elements that would normally be done in step 2 is delayed or ended early once its been determined that sufficient content has been created to fill the viewport. A virtualizing container (for example, ItemsRepeater) defers to its attached layout to drive this process. It provides the attached layout with a VirtualizingLayoutContext that surfaces the additional information that a virtualizing layout needs.

The RealizationRect (i.e. Viewport)

Scrolling on Windows happens asynchronous to the UI thread. It is not controlled by the framework's layout. Rather, the interaction and movement occurs in the system's compositor. The advantage of this approach is that panning content can always be done at 60fps. The challenge, however, is that the "viewport", as seen by the layout, might be slightly out-of-date relative to what is actually visible on screen. If a user scrolls quickly, they may outpace the speed of the UI thread to generate new content and "pan to black". For this reason, it's often necessary for a virtualizing layout to generate an additional buffer of prepared elements sufficient to fill an area larger than the viewport. When under heavier load during scrolling the user is still presented with content.



Since element creation is costly, virtualizing containers (for example, ItemsRepeater) will initially provide the attached layout with a RealizationRect that matches the viewport. On idle time the container may grow the buffer of prepared content by making repeated calls to the layout using an increasingly larger realization rect. This behavior is a performance optimization that attempts to strike a balance between fast startup time and a good panning experience. The maximum buffer size that the ItemsRepeater will generate is controlled by its VerticalCacheLength and HorizontalCacheLength properties.

Re-using Elements (Recycling)

The layout is expected to size and position the elements to fill the RealizationRect each time it is run. By default the VirtualizingLayout will recycle any unused elements at the end of each layout pass.

The VirtualizingLayoutContext that is passed to the layout as part of the MeasureOverride and ArrangeOverride provides the additional information a virtualizing layout needs. Some of the most commonly used things it provides are the ability to:

1. Query the number of items in the data (ItemCount).
2. Retrieve a specific item using the GetItemAt method.
3. Retrieve a RealizationRect that represents the viewport and buffer that the layout should fill with realized elements.
4. Request the UIElement for a specific item with the GetOrCreateElement method.

Requesting an element for a given index will cause that element to be marked as "in use" for that pass of the layout. If the element does not already exist, then it will be realized and automatically prepared for use (for example, inflating the UI tree defined in a DataTemplate, processing any data binding, etc.). Otherwise, it will be retrieved from a pool of existing instances.

At the end of each measure pass, any existing, realized element that was not marked "in use" is automatically considered available for re-use unless the option to SuppressAutoRecycle was used when the element was retrieved via the GetOrCreateElementAt method. The framework automatically moves it to a recycle pool and makes it available. It may subsequently be pulled for use by a different container. The framework tries to avoid this when possible as there is some cost associated with re-parenting an element.

If a virtualizing layout knows at the beginning of each measure which elements will no longer fall within the realization rect then it can optimize its re-use. Rather than relying on the framework's default behavior. The layout can preemptively move elements to the recycle pool by using the RecycleElement method. Calling this method before requesting new elements causes those existing elements to be available when the layout later issues a GetOrCreateElementAt request for an index that isn't already associated with an element.

The VirtualizingLayoutContext provides two additional properties designed for layout authors creating a content-dependent layout. They are discussed in more detail later.

1. A RecommendedAnchorIndex that provides an optional *input* to layout.
2. A LayoutOrigin that is an optional *output* of the layout.

Data-dependent Virtualizing Layouts

A virtualizing layout is easier if you know what the size of every item should be without needing to measure the content to show. In this doc we'll simply refer to this category of virtualizing layouts as **data layouts** since they usually involve inspecting the data. Based on the data an app may pick a visual representation with a known size - perhaps because its part of the data or was previously determined by design.

The general approach is for the layout to:

1. Calculate a size and position of every item.
2. As part of the MeasureOverride:
    a. Use the RealizationRect to determine which items should appear within the viewport.

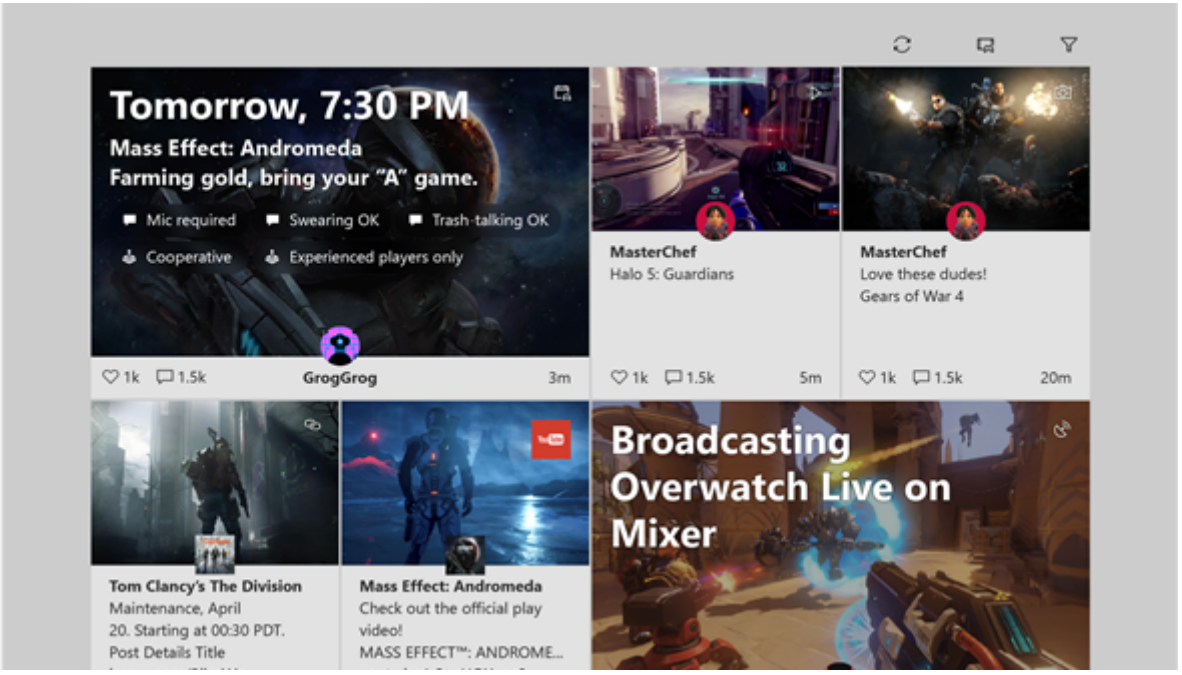
    b. Retrieve the UIElement that should represent the item with the GetOrCreateElementAt method.
    c. Measure the UIElement with the pre-calculated size.
3. As part of the ArrangeOverride, Arrange each realized UIElement with the precalculated position.

> **Note**
>
> A data layout approach is often incompatible with *data virtualization*. Specifically, where the only data loaded into memory is that data required to fill what's visible to the user. Data virtualization isn't referring to lazy or incremental loading of data as a user scrolls down where that data remains resident. Rather, it's referring to when items are released from memory as they're scrolled out of view. Having a data layout that inspects every data item as part of a data layout would prevent data virtualization from working as expected. An exception is a layout like the UniformGridLayout which assumes that everything has the same size.

> **Tip**
>
> If you're creating a custom control for a control library that will be used by others in a wide variety of situations then a data layout may not be an option for you.

Example: Xbox Activity Feed layout

The UI for the Xbox Activity Feed uses a repeating pattern where each line has a wide tile, followed by two narrow tiles that is inverted on the subsequent line. In this layout, the size for every item is a function of the item's position in the data set and the known size for the tiles (wide vs narrow).



The code below walks through what a custom virtualizing UI for the activity feed might be to illustrate the general approach you might take for a **data layout**.

> **Tip**
>
> If you have the **WinUI 3 Gallery** app installed, click here to **open the app and see the ItemsRepeater in action**. Get the app from the **Microsoft Store** or get the source code on **GitHub**.

Implementation

C#

```
/// <summary>
///  This is a custom layout that displays elements in two different sizes
///  wide (w) and narrow (n). There are two types of rows
///  odd rows - narrow narrow wide
///  even rows - wide narrow narrow
///  This pattern repeats.
/// </summary>

public class ActivityFeedLayout : VirtualizingLayout // STEP #1 Inherit from
base attached layout
{
    // STEP #2 - Parameterize the layout
    #region Layout parameters

    // We'll cache copies of the dependency properties to avoid calling
GetValue during layout since that
    // can be quite expensive due to the number of times we'd end up calling
these.
```

```
    private double _rowSpacing;
    private double _colSpacing;
    private Size _minItemSize = Size.Empty;

    /// <summary>
    /// Gets or sets the size of the whitespace gutter to include between
rows
    /// </summary>
    public double RowSpacing
    {
        get { return _rowSpacing; }
        set { SetValue(RowSpacingProperty, value); }
    }

    /// <summary>
    /// Gets or sets the size of the whitespace gutter to include between
items on the same row
    /// </summary>
    public double ColumnSpacing
    {
        get { return _colSpacing; }
        set { SetValue(ColumnSpacingProperty, value); }
    }

    public Size MinItemSize
    {
        get { return _minItemSize; }
        set { SetValue(MinItemSizeProperty, value); }
    }

    public static readonly DependencyProperty RowSpacingProperty =
        DependencyProperty.Register(
            nameof(RowSpacing),
            typeof(double),
            typeof(ActivityFeedLayout),
            new PropertyMetadata(0, OnPropertyChanged));

    public static readonly DependencyProperty ColumnSpacingProperty =
        DependencyProperty.Register(
            nameof(ColumnSpacing),
            typeof(double),
            typeof(ActivityFeedLayout),
            new PropertyMetadata(0, OnPropertyChanged));

    public static readonly DependencyProperty MinItemSizeProperty =
        DependencyProperty.Register(
            nameof(MinItemSize),
            typeof(Size),
            typeof(ActivityFeedLayout),
            new PropertyMetadata(Size.Empty, OnPropertyChanged));

    private static void OnPropertyChanged(DependencyObject obj,
DependencyPropertyChangedEventArgs args)
    {
        var layout = obj as ActivityFeedLayout;
```

```
        if (args.Property == RowSpacingProperty)
        {
            layout._rowSpacing = (double)args.NewValue;
        }
        else if (args.Property == ColumnSpacingProperty)
        {
            layout._colSpacing = (double)args.NewValue;
        }
        else if (args.Property == MinItemSizeProperty)
        {
            layout._minItemSize = (Size)args.NewValue;
        }
        else
        {
            throw new InvalidOperationException("Don't know what you are
talking about!");
        }

        layout.InvalidateMeasure();
    }

    #endregion

    #region Setup / teardown // STEP #3: Initialize state

    protected override void
InitializeForContextCore(VirtualizingLayoutContext context)
    {
        base.InitializeForContextCore(context);

        var state = context.LayoutState as ActivityFeedLayoutState;
        if (state == null)
        {
            // Store any state we might need since (in theory) the layout
could be in use by multiple
            // elements simultaneously
            // In reality for the Xbox Activity Feed there's probably only a
single instance.
            context.LayoutState = new ActivityFeedLayoutState();
        }
    }

    protected override void
UninitializeForContextCore(VirtualizingLayoutContext context)
    {
        base.UninitializeForContextCore(context);

        // clear any state
        context.LayoutState = null;
    }

    #endregion

    #region Layout // STEP #4,5 - Measure and Arrange
```

```
    protected override Size MeasureOverride(VirtualizingLayoutContext
context, Size availableSize)
    {
        if (this.MinItemSize == Size.Empty)
        {
            var firstElement = context.GetOrCreateElementAt(0);
            firstElement.Measure(new Size(double.PositiveInfinity,
double.PositiveInfinity));

            // setting the member value directly to skip invalidating layout
            this._minItemSize = firstElement.DesiredSize;
        }

        // Determine which rows need to be realized.  We know every row will
have the same height and
        // only contain 3 items.  Use that to determine the index for the
first and last item that
        // will be within that realization rect.
        var firstRowIndex = Math.Max(
            (int)(context.RealizationRect.Y / (this.MinItemSize.Height +
this.RowSpacing)) - 1,
            0);
        var lastRowIndex = Math.Min(
            (int)(context.RealizationRect.Bottom / (this.MinItemSize.Height
+ this.RowSpacing)) + 1,
            (int)(context.ItemCount / 3));

        // Determine which items will appear on those rows and what the rect
will be for each item
        var state = context.LayoutState as ActivityFeedLayoutState;
        state.LayoutRects.Clear();

        // Save the index of the first realized item.  We'll use it as a
starting point during arrange.
        state.FirstRealizedIndex = firstRowIndex * 3;

        // ideal item width that will expand/shrink to fill available space
        double desiredItemWidth = Math.Max(this.MinItemSize.Width,
(availableSize.Width - this.ColumnSpacing * 3) / 4);

        // Foreach item between the first and last index,
        //     Call GetElementOrCreateElementAt which causes an element to
either be realized or retrieved
        //       from a recycle pool
        //     Measure the element using an appropriate size
        //
        // Any element that was previously realized which we don't retrieve
in this pass (via a call to
        // GetElementOrCreateAt) will be automatically cleared and set aside
for later re-use.
        // Note: While this work fine, it does mean that more elements than
are required may be
        // created because it isn't until after our MeasureOverride
completes that the unused elements
        // will be recycled and available to use.  We could avoid this by
```

```
choosing to track the first/last
        // index from the previous layout pass.  The diff between the
previous range and current range
        // would represent the elements that we can pre-emptively make
available for re-use by calling
        // context.RecycleElement(element).
        for (int rowIndex = firstRowIndex; rowIndex < lastRowIndex;
rowIndex++)
        {
            int firstItemIndex = rowIndex * 3;
            var boundsForCurrentRow = CalculateLayoutBoundsForRow(rowIndex,
desiredItemWidth);

            for (int columnIndex = 0; columnIndex < 3; columnIndex++)
            {
                var index = firstItemIndex + columnIndex;
                var rect = boundsForCurrentRow[index % 3];
                var container = context.GetOrCreateElementAt(index);

                container.Measure(
                    new Size(boundsForCurrentRow[columnIndex].Width,
boundsForCurrentRow[columnIndex].Height));

                state.LayoutRects.Add(boundsForCurrentRow[columnIndex]);
            }
        }

        // Calculate and return the size of all the content (realized or
not) by figuring out
        // what the bottom/right position of the last item would be.
        var extentHeight = ((int)(context.ItemCount / 3) - 1) *
(this.MinItemSize.Height + this.RowSpacing) + this.MinItemSize.Height;

        // Report this as the desired size for the layout
        return new Size(desiredItemWidth * 4 + this.ColumnSpacing * 2,
extentHeight);
    }

    protected override Size ArrangeOverride(VirtualizingLayoutContext
context, Size finalSize)
    {
        // walk through the cache of containers and arrange
        var state = context.LayoutState as ActivityFeedLayoutState;
        var virtualContext = context as VirtualizingLayoutContext;
        int currentIndex = state.FirstRealizedIndex;

        foreach (var arrangeRect in state.LayoutRects)
        {
            var container =
virtualContext.GetOrCreateElementAt(currentIndex);
            container.Arrange(arrangeRect);
            currentIndex++;
        }

        return finalSize;
```

```
    }

    #endregion
    #region Helper methods

    private Rect[] CalculateLayoutBoundsForRow(int rowIndex, double
desiredItemWidth)
    {
        var boundsForRow = new Rect[3];

        var yoffset = rowIndex * (this.MinItemSize.Height +
this.RowSpacing);
        boundsForRow[0].Y = boundsForRow[1].Y = boundsForRow[2].Y = yoffset;
        boundsForRow[0].Height = boundsForRow[1].Height =
boundsForRow[2].Height = this.MinItemSize.Height;

        if (rowIndex % 2 == 0)
        {
            // Left tile (narrow)
            boundsForRow[0].X = 0;
            boundsForRow[0].Width = desiredItemWidth;
            // Middle tile (narrow)
            boundsForRow[1].X = boundsForRow[0].Right + this.ColumnSpacing;
            boundsForRow[1].Width = desiredItemWidth;
            // Right tile (wide)
            boundsForRow[2].X = boundsForRow[1].Right + this.ColumnSpacing;
            boundsForRow[2].Width = desiredItemWidth * 2 +
this.ColumnSpacing;
        }
        else
        {
            // Left tile (wide)
            boundsForRow[0].X = 0;
            boundsForRow[0].Width = (desiredItemWidth * 2 +
this.ColumnSpacing);
            // Middle tile (narrow)
            boundsForRow[1].X = boundsForRow[0].Right + this.ColumnSpacing;
            boundsForRow[1].Width = desiredItemWidth;
            // Right tile (narrow)
            boundsForRow[2].X = boundsForRow[1].Right + this.ColumnSpacing;
            boundsForRow[2].Width = desiredItemWidth;
        }

        return boundsForRow;
    }

    #endregion
}

internal class ActivityFeedLayoutState
{
    public int FirstRealizedIndex { get; set; }

    /// <summary>
    /// List of layout bounds for items starting with the
```

```
    /// FirstRealizedIndex.
    /// </summary>
    public List<Rect> LayoutRects
    {
        get
        {
            if (_layoutRects == null)
            {
                _layoutRects = new List<Rect>();
            }

            return _layoutRects;
        }
    }

    private List<Rect> _layoutRects;
}
```

(Optional) Managing the Item to UIElement Mapping

By default, the VirtualizingLayoutContext maintains a mapping between the realized elements and the index in the data source they represent. A layout can choose to manage this mapping itself by always requesting the option to SuppressAutoRecycle when retrieving an element via the GetOrCreateElementAt method which prevents the default auto-recycling behavior. A layout may choose to do this, for example, if it will only be used when scrolling is restricted to one direction and the items it considers will always be contiguous (i.e. knowing the index of the first and last element is enough to know all the elements that should be realized).

Example: Xbox Activity Feed measure

The snippet below shows the additional logic that could be added to the MeasureOverride in the previous sample to manage the mapping.

C#

```
    protected override Size MeasureOverride(VirtualizingLayoutContext
context, Size availableSize)
    {
        //...

        // Determine which items will appear on those rows and what the rect
will be for each item
        var state = context.LayoutState as ActivityFeedLayoutState;
        state.LayoutRects.Clear();

         // Recycle previously realized elements that we know we won't need
so that they can be used to
```

```
        // fill in gaps without requiring us to realize additional elements.
        var newFirstRealizedIndex = firstRowIndex * 3;
        var newLastRealizedIndex = lastRowIndex * 3 + 3;
        for (int i = state.FirstRealizedIndex; i < newFirstRealizedIndex;
i++)
        {
            context.RecycleElement(state.IndexToElementMap.Get(i));
            state.IndexToElementMap.Clear(i);
        }

        for (int i = state.LastRealizedIndex; i < newLastRealizedIndex; i++)
        {
            context.RecycleElement(context.IndexElementMap.Get(i));
            state.IndexToElementMap.Clear(i);
        }

        // ...

        // Foreach item between the first and last index,
        //     Call GetElementOrCreateElementAt which causes an element to
either be realized or retrieved
        //       from a recycle pool
        //     Measure the element using an appropriate size
        //
        for (int rowIndex = firstRowIndex; rowIndex < lastRowIndex;
rowIndex++)
        {
            int firstItemIndex = rowIndex * 3;
            var boundsForCurrentRow = CalculateLayoutBoundsForRow(rowIndex,
desiredItemWidth);

            for (int columnIndex = 0; columnIndex < 3; columnIndex++)
            {
                var index = firstItemIndex + columnIndex;
                var rect = boundsForCurrentRow[index % 3];
                UIElement container = null;
                if (state.IndexToElementMap.Contains(index))
                {
                    container = state.IndexToElementMap.Get(index);
                }
                else
                {
                    container = context =
context.GetOrCreateElementAt(index, ElementRealizationOptions.ForceCreate |
ElementRealizationOptions.SuppressAutoRecycle);
                    state.IndexToElementMap.Add(index, container);
                }

                container.Measure(
                    new Size(boundsForCurrentRow[columnIndex].Width,
boundsForCurrentRow[columnIndex].Height));

                state.LayoutRects.Add(boundsForCurrentRow[columnIndex]);
            }
        }
```

```
        // ...
    }

internal class ActivityFeedLayoutState
{
    // ...
    Dictionary<int, UIElement> IndexToElementMap { get; set; }
    // ...
}
```

Content-dependent Virtualizing Layouts

If you must first measure the UI content for an item to figure out its exact size then it is a **content-dependent layout**. You can also think of it as a layout where each item must size itself rather than the layout telling the item its size. Virtualizing layouts that fall into this category are more involved.

> ⓘ Note
>
> Content-dependent layouts don't (shouldn't) break data virtualization.

Estimations

Content-dependent layouts rely on estimation to guess both the size of unrealized content and the position of the realized content. As those estimates change it will cause the realized content to regularly shift positions within the scrollable area. This can lead to a very frustrating and jarring user experience if not mitigated. The potential issues and mitigations are discussed here.

> ⓘ Note
>
> Data layouts that consider every item and know the exact size of all items, realized or not, and their positions can avoid these issues entirely.

Scroll Anchoring

XAML provides a mechanism to mitigate sudden viewport shifts by having scrolling controls support scroll anchoring by implementing the IScrollAnchorPovider interface. As the user manipulates the content, the scrolling control continually selects an element from the set of candidates that were opted-in to be tracked. If the position of the

anchor element shifts during the layout then the scroll control automatically shifts its viewport to maintain the viewport.

The value of the RecommendedAnchorIndex provided to the layout may reflect that currently selected anchor element chosen by the scrolling control. Alternatively, if a developer explicitly requests that an element be realized for an index with the GetOrCreateElement method on the ItemsRepeater, then that index is given as the RecommendedAnchorIndex on the next layout pass. This enables the layout to be prepared for the likely scenario that a developer realizes an element and subsequently requests that it be brought into view via the StartBringIntoView method.

The RecommendedAnchorIndex is the index for the item in the data source that a content-dependent layout should position first when estimating the position of its items. It should serve as the starting point for positioning other realized items.

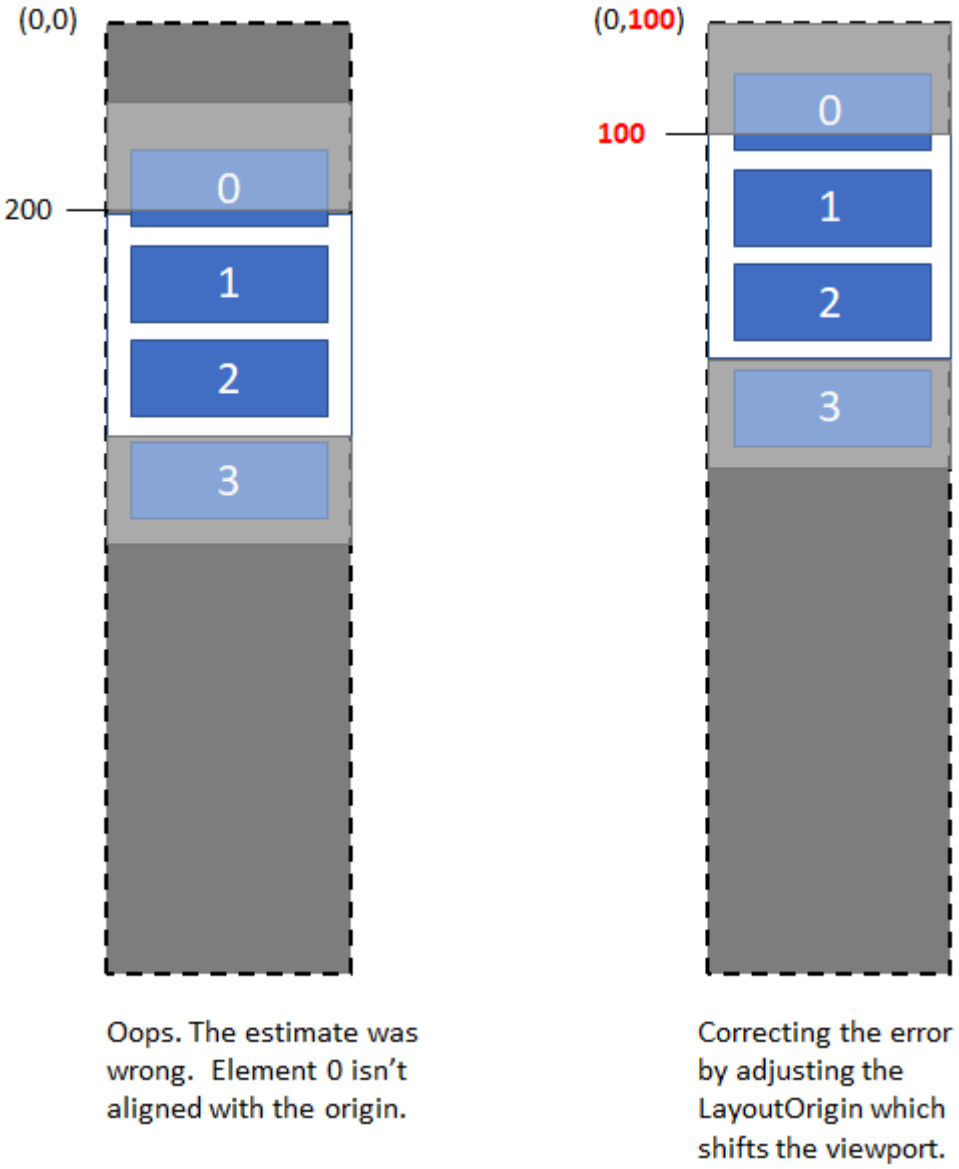
Impact on ScrollBars

Even with scroll anchoring, if the layout's estimates vary a lot, perhaps due to significant variations in the size of content, then the position of the thumb for the ScrollBar may appear to jump around. This can be jarring for a user if the thumb doesn't appear to track the position of their mouse pointer when they're dragging it.

The more accurate the layout can be in its estimations then the less likely a user will see the ScrollBar's thumb jumping.

Layout Corrections

A content-dependent layout should be prepared to rationalize its estimate with reality. For example, as the user scrolls to the top of the content and the layout realizes the very first element, it may find that the element's anticipated position relative to the element from which it started would cause it to appear somewhere other than the origin of (x:0, y:0). When this occurs, the layout can use the LayoutOrigin property to set the position it calculated as the new layout origin. The net result is similar to scroll anchoring in which the scrolling control's viewport is automatically adjusted to account for the content's position as reported by the layout.



Disconnected Viewports

The size returned from the layout's MeasureOverride method represents the best guess at the size of the content which may change with each successive layout. As a user scrolls the layout will be continually re-evaluated with an updated RealizationRect.

If a user drags the thumb very quickly then its possible for the viewport, from the perspective of the layout, to appear to make large jumps where the prior position doesn't overlap the now current position. This is due to the asynchronous nature of scrolling. It's also possible for an app that is consuming the layout to request that an element be brought into view for an item that is not currently realized and is estimated to lay outside the current range tracked by the layout.

When the layout discovers its guess is incorrect and/or sees an unexpected viewport shift, it needs to reorient its starting position. The virtualizing layouts that ship as part of the XAML controls are developed as content-dependent layouts as they place fewer restrictions on the nature of the content that will be shown.

Example: Simple Virtualizing Stack Layout for Variable-Sized Items

The sample below demonstrates a simple stack layout for variable-sized items that:

- supports UI virtualization,
- uses estimations to guess the size of unrealized items,
- is aware of potential discontinuous viewport shifts, and
- applies layout corrections to account for those shifts.

Usage: Markup

XAML

```
<ScrollViewer>

  <ItemsRepeater x:Name="repeater" >
    <ItemsRepeater.Layout>

      <local:VirtualizingStackLayout />

    </ItemsRepeater.Layout>
    <ItemsRepeater.ItemTemplate>
      <DataTemplate x:Key="item">
        <UserControl IsTabStop="True" UseSystemFocusVisuals="True"
Margin="5">
          <StackPanel BorderThickness="1" Background="LightGray" Margin="5">
            <Image x:Name="recipeImage" Source="{Binding ImageUri}"
Width="100" Height="100"/>
              <TextBlock x:Name="recipeDescription"
                         Text="{Binding Description}"
                         TextWrapping="Wrap"
                         Margin="10" />
          </StackPanel>
        </UserControl>
      </DataTemplate>
    </ItemsRepeater.ItemTemplate>
  </ItemsRepeater>

</ScrollViewer>
```

Codebehind: Main.cs

C#

```
string _lorem = @"Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Etiam laoreet erat vel massa rutrum, eget mollis massa vulputate. Vivamus
semper augue leo, eget faucibus nulla mattis nec. Donec scelerisque lacus at
dui ultricies, eget auctor ipsum placerat. Integer aliquet libero sed nisi
eleifend, nec rutrum arcu lacinia. Sed a sem et ante gravida congue sit amet
```

```
ut augue. Donec quis pellentesque urna, non finibus metus. Proin sed ornare
tellus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam
laoreet erat vel massa rutrum, eget mollis massa vulputate. Vivamus semper
augue leo, eget faucibus nulla mattis nec. Donec scelerisque lacus at dui
ultricies, eget auctor ipsum placerat. Integer aliquet libero sed nisi
eleifend, nec rutrum arcu lacinia. Sed a sem et ante gravida congue sit amet
ut augue. Donec quis pellentesque urna, non finibus metus. Proin sed ornare
tellus.";

var rnd = new Random();
var data = new ObservableCollection<Recipe>(Enumerable.Range(0,
300).Select(k =>
               new Recipe
               {
                   ImageUri = new Uri(string.Format("ms-
appx:///Images/recipe{0}.png", k % 8 + 1)),
                   Description = k + " - " + _lorem.Substring(0,
rnd.Next(50, 350))
               }));

repeater.ItemsSource = data;
```

Code: VirtualizingStackLayout.cs

C#

```
// This is a sample layout that stacks elements one after
// the other where each item can be of variable height. This is
// also a virtualizing layout - we measure and arrange only elements
// that are in the viewport. Not measuring/arranging all elements means
// that we do not have the complete picture and need to estimate sometimes.
// For example the size of the layout (extent) is an estimation based on the
// average heights we have seen so far. Also, if you drag the mouse thumb
// and yank it quickly, then we estimate what goes in the new viewport.

// The layout caches the bounds of everything that are in the current
viewport.
// During measure, we might get a suggested anchor (or start index), we use
that
// index to start and layout the rest of the items in the viewport relative
to that
// index. Note that since we are estimating, we can end up with negative
origin when
// the viewport is somewhere in the middle of the extent. This is achieved
by setting the
// LayoutOrigin property on the context. Once this is set, future viewport
will account
// for the origin.
public class VirtualizingStackLayout : VirtualizingLayout
{
    // Estimation state
    List<double> m_estimationBuffer = Enumerable.Repeat(0d, 100).ToList();
    int m_numItemsUsedForEstimation = 0;
```

```
    double m_totalHeightForEstimation = 0;

    // State to keep track of realized bounds
    int m_firstRealizedDataIndex = 0;
    List<Rect> m_realizedElementBounds = new List<Rect>();

    Rect m_lastExtent = new Rect();

    protected override Size MeasureOverride(VirtualizingLayoutContext
context, Size availableSize)
    {
        var viewport = context.RealizationRect;
        DebugTrace("MeasureOverride: Viewport " + viewport);

        // Remove bounds for elements that are now outside the viewport.
        // Proactive recycling elements means we can reuse it during this
measure pass again.
        RemoveCachedBoundsOutsideViewport(viewport);

        // Find the index of the element to start laying out from - the
anchor
        int startIndex = GetStartIndex(context, availableSize);

        // Measure and layout elements starting from the start index,
forward and backward.
        Generate(context, availableSize, startIndex, forward:true);
        Generate(context, availableSize, startIndex, forward:false);

        // Estimate the extent size. Note that this can have a non 0 origin.
        m_lastExtent = EstimateExtent(context, availableSize);
        context.LayoutOrigin = new Point(m_lastExtent.X, m_lastExtent.Y);
        return new Size(m_lastExtent.Width, m_lastExtent.Height);
    }

    protected override Size ArrangeOverride(VirtualizingLayoutContext
context, Size finalSize)
    {
        DebugTrace("ArrangeOverride: Viewport" + context.RealizationRect);
        for (int realizationIndex = 0; realizationIndex <
m_realizedElementBounds.Count; realizationIndex++)
        {
            int currentDataIndex = m_firstRealizedDataIndex +
realizationIndex;
            DebugTrace("Arranging " + currentDataIndex);

            // Arrange the child. If any alignment needs to be done, it
            // can be done here.
            var child = context.GetOrCreateElementAt(currentDataIndex);
            var arrangeBounds = m_realizedElementBounds[realizationIndex];
            arrangeBounds.X -= m_lastExtent.X;
            arrangeBounds.Y -= m_lastExtent.Y;
            child.Arrange(arrangeBounds);
        }

        return finalSize;
```

```
    }

    // The data collection has changed, since we are maintaining the bounds
of elements
    // in the viewport, we will update the list to account for the
collection change.
    protected override void OnItemsChangedCore(VirtualizingLayoutContext
context, object source, NotifyCollectionChangedEventArgs args)
    {
        InvalidateMeasure();
        if (m_realizedElementBounds.Count > 0)
        {
            switch (args.Action)
            {
                case NotifyCollectionChangedAction.Add:
                    OnItemsAdded(args.NewStartingIndex,
args.NewItems.Count);
                    break;
                case NotifyCollectionChangedAction.Replace:
                    OnItemsRemoved(args.OldStartingIndex,
args.OldItems.Count);
                    OnItemsAdded(args.NewStartingIndex,
args.NewItems.Count);
                    break;
                case NotifyCollectionChangedAction.Remove:
                    OnItemsRemoved(args.OldStartingIndex,
args.OldItems.Count);
                    break;
                case NotifyCollectionChangedAction.Reset:
                    m_realizedElementBounds.Clear();
                    m_firstRealizedDataIndex = 0;
                    break;
                default:
                    throw new NotImplementedException();
            }
        }
    }

    // Figure out which index to use as the anchor and start laying out
around it.
    private int GetStartIndex(VirtualizingLayoutContext context, Size
availableSize)
    {
        int startDataIndex = -1;
        var recommendedAnchorIndex = context.RecommendedAnchorIndex;
        bool isSuggestedAnchorValid = recommendedAnchorIndex != -1;

        if (isSuggestedAnchorValid)
        {
            if (IsRealized(recommendedAnchorIndex))
            {
                startDataIndex = recommendedAnchorIndex;
            }
            else
            {
```

```
                ClearRealizedRange();
                startDataIndex = recommendedAnchorIndex;
            }
        }
        else
        {
            // Find the first realized element that is visible in the
viewport.
            startDataIndex =
GetFirstRealizedDataIndexInViewport(context.RealizationRect);
            if (startDataIndex < 0)
            {
                startDataIndex =
EstimateIndexForViewport(context.RealizationRect, context.ItemCount);
                ClearRealizedRange();
            }
        }

        // We have an anchorIndex, realize and measure it and
        // figure out its bounds.
        if (startDataIndex != -1 & context.ItemCount > 0)
        {
            if (m_realizedElementBounds.Count == 0)
            {
                m_firstRealizedDataIndex = startDataIndex;
            }

            var newAnchor = EnsureRealized(startDataIndex);
            DebugTrace("Measuring start index " + startDataIndex);
            var desiredSize = MeasureElement(context, startDataIndex,
availableSize);

            var bounds = new Rect(
                0,
                newAnchor ?
                    (m_totalHeightForEstimation /
m_numItemsUsedForEstimation) * startDataIndex :
GetCachedBoundsForDataIndex(startDataIndex).Y,
                availableSize.Width,
                desiredSize.Height);
            SetCachedBoundsForDataIndex(startDataIndex, bounds);
        }

        return startDataIndex;
    }


    private void Generate(VirtualizingLayoutContext context, Size
availableSize, int anchorDataIndex, bool forward)
    {
        // Generate forward or backward from anchorIndex until we hit the
end of the viewport
        int step = forward ? 1 : -1;
        int previousDataIndex = anchorDataIndex;
        int currentDataIndex = previousDataIndex + step;
```

```
        var viewport = context.RealizationRect;
        while (IsDataIndexValid(currentDataIndex, context.ItemCount) &&
            ShouldContinueFillingUpSpace(previousDataIndex, forward,
viewport))
        {
            EnsureRealized(currentDataIndex);
            DebugTrace("Measuring " + currentDataIndex);
            var desiredSize = MeasureElement(context, currentDataIndex,
availableSize);
            var previousBounds =
GetCachedBoundsForDataIndex(previousDataIndex);
            Rect currentBounds = new Rect(0,
                                          forward ? previousBounds.Y +
previousBounds.Height : previousBounds.Y - desiredSize.Height,
                                          availableSize.Width,
                                          desiredSize.Height);
            SetCachedBoundsForDataIndex(currentDataIndex, currentBounds);
            previousDataIndex = currentDataIndex;
            currentDataIndex += step;
        }
    }

    // Remove bounds that are outside the viewport, leaving one extra since
our
    // generate stops after generating one extra to know that we are outside
the
    // viewport.
    private void RemoveCachedBoundsOutsideViewport(Rect viewport)
    {
        int firstRealizedIndexInViewport = 0;
        while (firstRealizedIndexInViewport < m_realizedElementBounds.Count
&&

!Intersects(m_realizedElementBounds[firstRealizedIndexInViewport],
viewport))
        {
            firstRealizedIndexInViewport++;
        }

        int lastRealizedIndexInViewport = m_realizedElementBounds.Count - 1;
        while (lastRealizedIndexInViewport >= 0 &&

!Intersects(m_realizedElementBounds[lastRealizedIndexInViewport], viewport))
        {
            lastRealizedIndexInViewport--;
        }

        if (firstRealizedIndexInViewport > 0)
        {
            m_firstRealizedDataIndex += firstRealizedIndexInViewport;
            m_realizedElementBounds.RemoveRange(0,
firstRealizedIndexInViewport);
        }

        if (lastRealizedIndexInViewport >= 0 && lastRealizedIndexInViewport
```

```
< m_realizedElementBounds.Count - 2)
        {
            m_realizedElementBounds.RemoveRange(lastRealizedIndexInViewport
+ 2, m_realizedElementBounds.Count - lastRealizedIndexInViewport - 3);
        }
    }

    private bool Intersects(Rect bounds, Rect viewport)
    {
        return !(bounds.Bottom < viewport.Top ||
            bounds.Top > viewport.Bottom);
    }

    private bool ShouldContinueFillingUpSpace(int dataIndex, bool forward,
Rect viewport)
    {
        var bounds = GetCachedBoundsForDataIndex(dataIndex);
        return forward ?
            bounds.Y < viewport.Bottom :
            bounds.Y > viewport.Top;
    }

    private bool IsDataIndexValid(int currentDataIndex, int itemCount)
    {
        return currentDataIndex >= 0 && currentDataIndex < itemCount;
    }

    private int EstimateIndexForViewport(Rect viewport, int dataCount)
    {
        double averageHeight = m_totalHeightForEstimation /
m_numItemsUsedForEstimation;
        int estimatedIndex = (int)(viewport.Top / averageHeight);
        // clamp to an index within the collection
        estimatedIndex = Math.Max(0, Math.Min(estimatedIndex, dataCount));
        return estimatedIndex;
    }

    private int GetFirstRealizedDataIndexInViewport(Rect viewport)
    {
        int index = -1;
        if (m_realizedElementBounds.Count > 0)
        {
            for (int i = 0; i < m_realizedElementBounds.Count; i++)
            {
                if (m_realizedElementBounds[i].Y < viewport.Bottom &&
                   m_realizedElementBounds[i].Bottom > viewport.Top)
                {
                    index = m_firstRealizedDataIndex + i;
                    break;
                }
            }
        }

        return index;
    }
```

```
    private Size MeasureElement(VirtualizingLayoutContext context, int
index, Size availableSize)
    {
        var child = context.GetOrCreateElementAt(index);
        child.Measure(availableSize);

        int estimationBufferIndex = index % m_estimationBuffer.Count;
        bool alreadyMeasured = m_estimationBuffer[estimationBufferIndex] !=
0;
        if (!alreadyMeasured)
        {
            m_numItemsUsedForEstimation++;
        }

        m_totalHeightForEstimation -=
m_estimationBuffer[estimationBufferIndex];
        m_totalHeightForEstimation += child.DesiredSize.Height;
        m_estimationBuffer[estimationBufferIndex] =
child.DesiredSize.Height;

        return child.DesiredSize;
    }

    private bool EnsureRealized(int dataIndex)
    {
        if (!IsRealized(dataIndex))
        {
            int realizationIndex = RealizationIndex(dataIndex);
            Debug.Assert(dataIndex == m_firstRealizedDataIndex - 1 ||
                dataIndex == m_firstRealizedDataIndex +
m_realizedElementBounds.Count ||
                m_realizedElementBounds.Count == 0);

            if (realizationIndex == -1)
            {
                m_realizedElementBounds.Insert(0, new Rect());
            }
            else
            {
                m_realizedElementBounds.Add(new Rect());
            }

            if (m_firstRealizedDataIndex > dataIndex)
            {
                m_firstRealizedDataIndex = dataIndex;
            }

            return true;
        }

        return false;
    }

    // Figure out the extent of the layout by getting the number of items
```

```
remaining
    // above and below the realized elements and getting an estimation based
on
    // average item heights seen so far.
    private Rect EstimateExtent(VirtualizingLayoutContext context, Size
availableSize)
    {
        double averageHeight = m_totalHeightForEstimation /
m_numItemsUsedForEstimation;

        Rect extent = new Rect(0, 0, availableSize.Width, context.ItemCount
* averageHeight);

        if (context.ItemCount > 0 && m_realizedElementBounds.Count > 0)
        {
            extent.Y = m_firstRealizedDataIndex == 0 ?
                            m_realizedElementBounds[0].Y :
                            m_realizedElementBounds[0].Y -
(m_firstRealizedDataIndex - 1) * averageHeight;

            int lastRealizedIndex = m_firstRealizedDataIndex +
m_realizedElementBounds.Count;
            if (lastRealizedIndex == context.ItemCount - 1)
            {
                var lastBounds =
m_realizedElementBounds[m_realizedElementBounds.Count - 1];
                extent.Y = lastBounds.Bottom;
            }
            else
            {
                var lastBounds =
m_realizedElementBounds[m_realizedElementBounds.Count - 1];
                int lastRealizedDataIndex = m_firstRealizedDataIndex +
m_realizedElementBounds.Count;
                int numItemsAfterLastRealizedIndex = context.ItemCount -
lastRealizedDataIndex;
                extent.Height = lastBounds.Bottom +
numItemsAfterLastRealizedIndex * averageHeight - extent.Y;
            }
        }

        DebugTrace("Extent " + extent + " with average height " +
averageHeight);
        return extent;
    }

    private bool IsRealized(int dataIndex)
    {
        int realizationIndex = dataIndex - m_firstRealizedDataIndex;
        return realizationIndex >= 0 && realizationIndex <
m_realizedElementBounds.Count;
    }

    // Index in the m_realizedElementBounds collection
    private int RealizationIndex(int dataIndex)
```

```
    {
        return dataIndex - m_firstRealizedDataIndex;
    }

    private void OnItemsAdded(int index, int count)
    {
        // Using the old indexes here (before it was updated by the
collection change)
        // if the insert data index is between the first and last realized
data index, we need
        // to insert items.
        int lastRealizedDataIndex = m_firstRealizedDataIndex +
m_realizedElementBounds.Count - 1;
        int newStartingIndex = index;
        if (newStartingIndex > m_firstRealizedDataIndex &&
            newStartingIndex <= lastRealizedDataIndex)
        {
            // Inserted within the realized range
            int insertRangeStartIndex = newStartingIndex -
m_firstRealizedDataIndex;
            for (int i = 0; i < count; i++)
            {
                // Insert null (sentinel) here instead of an element, that
way we do not
                // end up creating a lot of elements only to be thrown out
in the next layout.
                int insertRangeIndex = insertRangeStartIndex + i;
                int dataIndex = newStartingIndex + i;
                // This is to keep the contiguousness of the mapping
                m_realizedElementBounds.Insert(insertRangeIndex, new
Rect());
            }
        }
        else if (index <= m_firstRealizedDataIndex)
        {
            // Items were inserted before the realized range.
            // We need to update m_firstRealizedDataIndex;
            m_firstRealizedDataIndex += count;
        }
    }

    private void OnItemsRemoved(int index, int count)
    {
        int lastRealizedDataIndex = m_firstRealizedDataIndex +
m_realizedElementBounds.Count - 1;
        int startIndex = Math.Max(m_firstRealizedDataIndex, index);
        int endIndex = Math.Min(lastRealizedDataIndex, index + count - 1);
        bool removeAffectsFirstRealizedDataIndex = (index <=
m_firstRealizedDataIndex);

        if (endIndex >= startIndex)
        {
            ClearRealizedRange(RealizationIndex(startIndex), endIndex -
startIndex + 1);
        }
```

```
        if (removeAffectsFirstRealizedDataIndex &&
            m_firstRealizedDataIndex != -1)
        {
            m_firstRealizedDataIndex -= count;
        }
    }

    private void ClearRealizedRange(int startRealizedIndex, int count)
    {
        m_realizedElementBounds.RemoveRange(startRealizedIndex, count);
        if (startRealizedIndex == 0)
        {
            m_firstRealizedDataIndex = m_realizedElementBounds.Count == 0 ?
0 : m_firstRealizedDataIndex + count;
        }
    }

    private void ClearRealizedRange()
    {
        m_realizedElementBounds.Clear();
        m_firstRealizedDataIndex = 0;
    }

    private Rect GetCachedBoundsForDataIndex(int dataIndex)
    {
        return m_realizedElementBounds[RealizationIndex(dataIndex)];
    }

    private void SetCachedBoundsForDataIndex(int dataIndex, Rect bounds)
    {
        m_realizedElementBounds[RealizationIndex(dataIndex)] = bounds;
    }

    private Rect GetCachedBoundsForRealizationIndex(int relativeIndex)
    {
        return m_realizedElementBounds[relativeIndex];
    }

    void DebugTrace(string message, params object[] args)
    {
        Debug.WriteLine(message, args);
    }
}
```

Related articles

- ItemsRepeater
- ScrollViewer

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Transforms overview

Article • 11/26/2024

Learn how to use transforms in the Windows Runtime API, by changing the relative coordinate systems of elements in the UI. This can be used to adjust the appearance of individual XAML elements, such as scaling, rotating, or transforming the position in x-y space.

What is a transform?

A *transform* defines how to map, or transform, points from one coordinate space to another coordinate space. When a transform is applied to a UI element, it changes how that UI element is rendered to the screen as part of the UI.

Think of transforms in four broad classifications: translation, rotation, scaling and skew (or shear). For the purposes of using graphics APIs to change the appearance of UI elements, it's usually easiest to create transforms that define only one operation at a time. So the Windows Runtime defines a discrete class for each of these transform classifications:

- **TranslateTransform**: translates an element in x-y space, by setting values for **X** and **Y**.
- **ScaleTransform**: scales the transform based on a center point, by setting values for **CenterX**, **CenterY**, **ScaleX** and **ScaleY**.
- **RotateTransform**: rotates in x-y space, by setting values for **Angle**, **CenterX** and **CenterY**.
- **SkewTransform**: skews or shears in x-y space, by setting values for **AngleX**, **AngleY**, **CenterX** and **CenterY**.

Of these, you're likely to use **TranslateTransform** and **ScaleTransform** most often for UI scenarios.

You can combine transforms, and there are two Windows Runtime classes that support this: **CompositeTransform** and **TransformGroup**. In a **CompositeTransform**, transforms are applied in this order: scale, skew, rotate, translate. Use **TransformGroup** instead of **CompositeTransform** if you want the transforms applied in a different order. For more info, see **CompositeTransform**.

Transforms and layout

In XAML layout, transforms are applied after the layout pass is complete, so available space calculations and other layout decisions have been made before the transforms are applied. Because layout comes first, you'll sometimes get unexpected results if you transform elements that are in a **Grid** cell or similar layout container that allocates space during layout. The transformed element may appear truncated or obscured because it's trying to draw into an area that didn't calculate the post-transform dimensions when dividing space within its parent container. You may need to experiment with the transform results and adjust some settings. For example, instead of relying on adaptive layout and star sizing, you may need to change the **Center** properties or declare fixed pixel measurements for layout space to make sure the parent allots enough space.

Migration note: Windows Presentation Foundation (WPF) had a **LayoutTransform** property that applied transforms prior to the layout pass. But Windows Runtime XAML doesn't support a **LayoutTransform** property. (Microsoft Silverlight didn't have this property either.)

As an alternative, the Windows Community Toolkit provides the LayoutTransformControl that applies Matrix transformations on any FrameworkElement of your application.

Applying a transform to a UI element

When you apply a transform to an object, you typically do so to set the property **UIElement.RenderTransform**. Setting this property does not literally change the object pixel by pixel. What the property really does is apply the transform within the local coordinate space in which that object exists. Then the rendering logic and operation (post-layout) renders the combined coordinate spaces, making it look like the object has changed appearance and also potentially its layout position (if **TranslateTransform** was applied).

By default, each render transform is centered at the origin of the target object's local coordinate system—its (0,0). The only exception is a **TranslateTransform**, which has no center properties to set because the translation effect is the same regardless of where it is centered. But the other transforms each have properties that set **CenterX** and **CenterY** values.

Whenever you use transforms with **UIElement.RenderTransform**, remember that there's another property on **UIElement** that affects how the transform behaves: **RenderTransformOrigin**. What **RenderTransformOrigin** declares is whether the whole transform should apply to the default (0,0) point of an element or to some other origin point within the relative coordinate space of that element. For typical elements, (0,0) places the transform to the top left corner. Depending on what effect you want, you might choose to change **RenderTransformOrigin** rather than adjusting the **CenterX** and

CenterY values on transforms. Note that if you apply both **RenderTransformOrigin** and **CenterX** / **CenterY** values, the results can be pretty confusing, especially if you're animating any of the values.

For hit-testing purposes, an object to which a transform is applied continues to respond to input in an expected way that's consistent to its visual appearance in x-y space. For example, if you've used a **TranslateTransform** to move a **Rectangle** 400 pixels laterally in the UI, that **Rectangle** responds to **PointerPressed** events when the user presses the point where the **Rectangle** appears visually. You won't get false events if the user presses the area where the **Rectangle** was before being translated. For any z-index considerations that affect hit testing, applying a transform makes no difference; the z-index that governs which element handles input events for a point in x-y space is still evaluated using the child order as declared in a container. That order is usually the same as the order in which you declare the elements in XAML, although for child elements of a **Canvas** object you can adjust the order by applying the **Canvas.ZIndex** attached property to child elements.

Other transform properties

- **Brush.Transform**, **Brush.RelativeTransform**: These influence how a **Brush** uses coordinate space within the area that the **Brush** is applied to set visual properties such as foregrounds and backgrounds. These transforms aren't relevant for the most common brushes (which are typically setting solid colors with **SolidColorBrush**) but might be occasionally useful when painting areas with an **ImageBrush** or **LinearGradientBrush**.
- **Geometry.Transform**: You might use this property to apply a transform to a geometry prior to using that geometry for a **Path.Data** property value.

Animating a transform

Transform objects can be animated. To animate a **Transform**, apply an animation of a compatible type to the property you want to animate. This typically means you're using **DoubleAnimation** or **DoubleAnimationUsingKeyFrames** objects to define the animation, because all of the transform properties are of type **Double**. Animations that affect a transform that's used for a **UIElement.RenderTransform** value are not considered to be dependent animations, even if they have a nonzero duration. For more info about dependent animations, see Storyboarded animations.

If you animate properties to produce an effect similar to a transform in terms of the net visual appearance—for example, animating the **Width** and **Height** of a **FrameworkElement** rather than applying a **TranslateTransform**—such animations are

almost always treated as dependent animations. You'd have to enable the animations and there could be significant performance issues with the animation, especially if you're trying to support user interaction while that object is being animated. For that reason it's preferable to use a transform and animate it rather than animating any other property where the animation would be treated as a dependent animation.

To target the transform, there must be an existing **Transform** as the value for **RenderTransform**. You typically put an element for the appropriate transform type in the initial XAML, sometimes with no properties set on that transform.

You typically use an indirect targeting technique to apply animations to the properties of a transform. For more info about indirect targeting syntax, see Storyboarded animations and Property-path syntax.

Default styles for controls sometimes define animations of transforms as part of their visual-state behavior. For example, the visual states for **ProgressRing** use animated **RotateTransform** values to "spin" the dots in the ring.

Here's a simple example of how to animate a transform. In this case, it's animating the **Angle** of a **RotateTransform** to spin a **Rectangle** in place around its visual center. This example names the **RotateTransform** so doesn't need indirect animation targeting, but you could alternatively leave the transform unnamed, name the element that the transform's applied to, and use indirect targeting such as `(UIElement.RenderTransform).(RotateTransform.Angle)`.

XML

```
<StackPanel Margin="15">
  <StackPanel.Resources>
    <Storyboard x:Name="myStoryboard">
      <DoubleAnimation
       Storyboard.TargetName="myTransform"
       Storyboard.TargetProperty="Angle"
       From="0" To="360" Duration="0:0:5"
       RepeatBehavior="Forever" />
    </Storyboard>
  </StackPanel.Resources>
  <Rectangle Width="50" Height="50" Fill="RoyalBlue"
   PointerPressed="StartAnimation">
    <Rectangle.RenderTransform>
      <RotateTransform x:Name="myTransform" Angle="45" CenterX="25"
CenterY="25" />
    </Rectangle.RenderTransform>
  </Rectangle>
</StackPanel>
```

XML

```
void StartAnimation (object sender, RoutedEventArgs e) {
    myStoryboard.Begin();
}
```

Accounting for coordinate frames of reference at run time

UIElement has a method named TransformToVisual, which can generate a Transform that correlates the coordinate frames of reference for two UI elements. You can use this to compare an element to the app's default coordinate frame of reference if you pass the root visual as the first parameter. This can be useful if you've captured an input event from a different element, or if you are trying to predict layout behavior without actually requesting a layout pass.

Event data obtained from pointer events provides access to a GetCurrentPoint method, where you can specify a *relativeTo* parameter to change the coordinate frame of reference to a specific element rather than the app default. This approach simply applies a translate transform internally and transforms the x-y coordinate data for you when it creates the returned PointerPoint object.

Describing a transform mathematically

A transform can be described in terms of a transformation matrix. A 3×3 matrix is used to describe the transformations in a two-dimensional, x-y plane. Affine transformation matrices can be multiplied to form any number of linear transformations, such as rotation and skew (shear), followed by translation. The final column of an affine transformation matrix is equal to (0, 0, 1), so you need to specify only the members of the first two columns in the mathematical description.

The mathematical description of a transform might be useful to you if you have a mathematical background or a familiarity with graphics-programming techniques that also use matrices to describe transformations of coordinate space. There's a Transform-derived class that enables you to express a transformation directly in terms of its 3×3 matrix: MatrixTransform. **MatrixTransform** has a Matrix property, which holds a structure that has six properties: M11, M12, M21, M22, OffsetX and OffsetY. Each Matrix property uses a **Double** value and corresponds to the six relevant values (columns 1 and 2) of an affine transformation matrix.

Expand table

Column 1	Column 2	Column 3
M11	M12	0
M21	M22	0
OffsetX	OffsetY	1

Any transform that you could describe with a **TranslateTransform**, **ScaleTransform**, **RotateTransform**, or **SkewTransform** object could be described equally by a **MatrixTransform** with a **Matrix** value. But you typically just use **TranslateTransform** and the others because the properties on those transform classes are easier to conceptualize than setting the vector components in a **Matrix**. It's also easier to animate the discrete properties of transforms; a **Matrix** is actually a structure and not a **DependencyObject**, so it can't support animated individual values.

Some XAML design tools that enable you to apply transformation operations will serialize the results as a **MatrixTransform**. In this case it may be best to use the same design tool again to change the transformation effect and serialize the XAML again, rather than trying to manipulate the **Matrix** values yourself directly in the XAML.

3-D transforms

In Windows 10, XAML introduced a new property, **UIElement.Transform3D**, that can be used to create 3D effects with UI. To do this, use **PerspectiveTransform3D** to add a shared 3D perspective or "camera" to your scene, and then use **CompositeTransform3D** to transform an element in 3D space, like you would use **CompositeTransform**. See **UIElement.Transform3D** for a discussion of how to implement 3D transforms.

For simpler 3D effects that only apply to a single object, the **UIElement.Projection** property can be used. Using a **PlaneProjection** as the value for this property is equivalent to applying a fixed perspective transform and one or more 3D transforms to the element. This type of transform is described in more detail in 3-D perspective effects for XAML UI.

Related topics

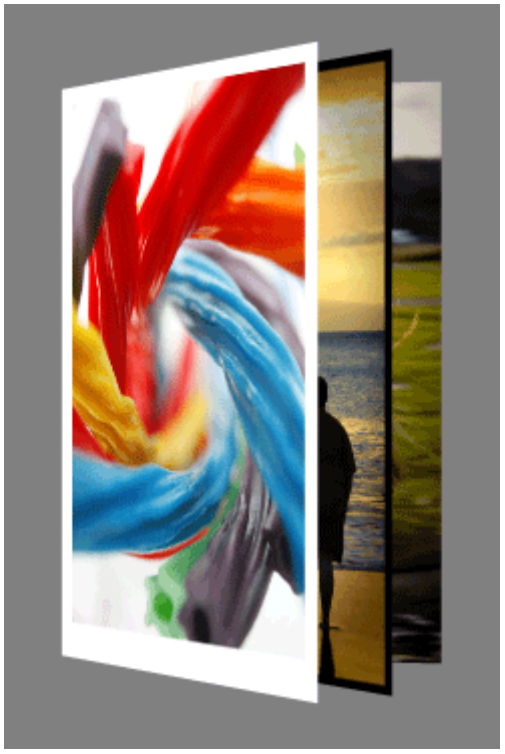
- **UIElement.Transform3D**
- 3-D perspective effects for XAML UI
- **Transform**

3-D perspective effects for XAML UI

Article • 11/26/2024

You can apply 3-D effects to content in your Windows Runtime apps using perspective transforms. For example, you can create the illusion that an object is rotated toward or away from you, as shown here.

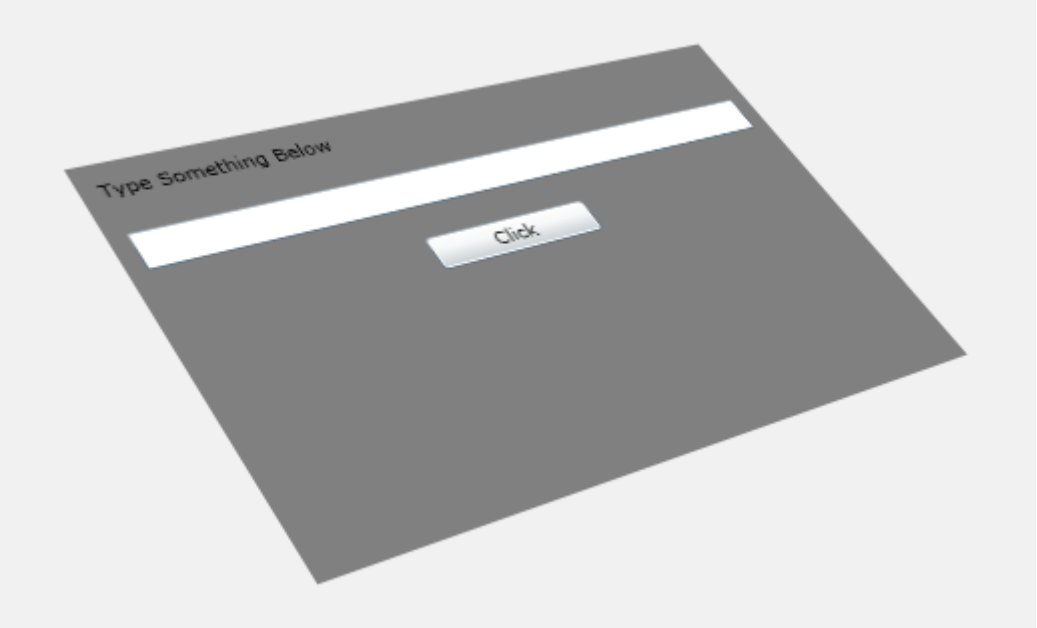
Another common usage for perspective transforms is to arrange objects in relation to one another to create a 3-D effect, as here.



Besides creating static 3-D effects, you can animate the perspective transform properties to create moving 3-D effects.

You just saw perspective transforms applied to images, but you can apply these effects to any **UIElement**, including controls. For example, you can apply a 3-D effect to an

entire container of controls like this:



Here is the XAML code used to create this sample:

XML

```
<StackPanel Margin="35" Background="Gray">
    <StackPanel.Projection>
        <PlaneProjection RotationX="-35" RotationY="-35" RotationZ="15"  />
    </StackPanel.Projection>
    <TextBlock Margin="10">Type Something Below</TextBlock>
    <TextBox Margin="10"></TextBox>
    <Button Margin="10" Content="Click" Width="100" />
</StackPanel>
```

Here we focus on the properties of **PlaneProjection** which is used to rotate and move objects in 3-D space. The next sample allows you to experiment with these properties and see their effect on an object.

PlaneProjection class

You can apply 3D effects can to any **UIElement**, by setting the UIElement's **Projection** property using a **PlaneProjection**. The **PlaneProjection** defines how the transform is rendered in space. The next example shows a simple case.

XML

```
<Image Source="kid.png">
    <Image.Projection>
        <PlaneProjection RotationX="-35"   />
```

```
    </Image.Projection>
</Image>
```

This figure shows what the image renders as. The x-axis, y-axis, and z-axis are shown as red lines. The image is rotated backward 35 degrees around the x-axis using the **RotationX** property.

The **RotationY** property rotates around the y-axis of the center of rotation.

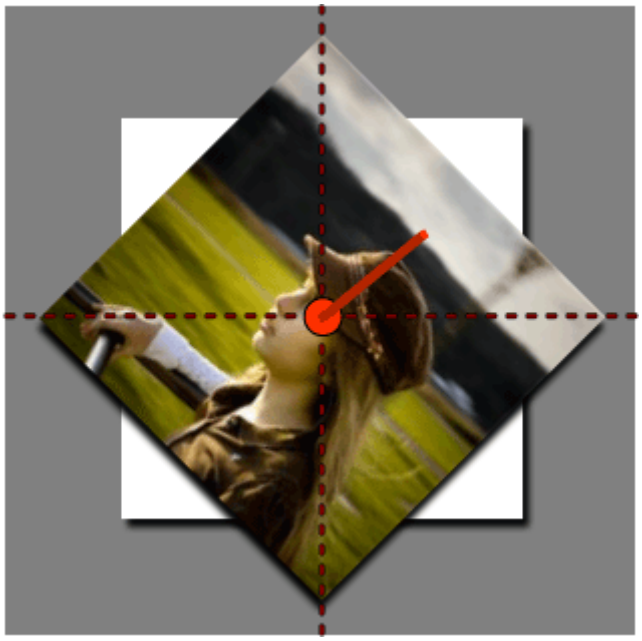
XML

```
<Image Source="kid.png">
    <Image.Projection>
        <PlaneProjection RotationY="-35"    />
    </Image.Projection>
</Image>
```

The **RotationZ** property rotates around the z-axis of the center of rotation (a line that is perpendicular to the plane of the object).

XML

```
<Image Source="kid.png">
    <Image.Projection>
        <PlaneProjection RotationZ="-45"/>
    </Image.Projection>
</Image>
```



The rotation properties can specify a positive or negative value to rotate in either direction. The absolute number can be greater than 360, which rotates the object more than one full rotation.

You can move the center of rotation by using the CenterOfRotationX, CenterOfRotationY, and CenterOfRotationZ properties. By default, the axes of rotation run directly through the center of the object, causing the object to rotate around its center. But if you move the center of rotation to the outer edge of the object, it will rotate around that edge. The default values for **CenterOfRotationX** and **CenterOfRotationY** are 0.5, and the default value for **CenterOfRotationZ** is 0. For **CenterOfRotationX** and **CenterOfRotationY**, values between 0 and 1 set the pivot point at some location within the object. A value of 0 denotes one object edge and 1 denotes the opposite edge. Values outside of this range are allowed and will move the center of rotation accordingly. Because the z-axis of the center of rotation is drawn through the plane of the object, you can move the center of rotation behind the object using a negative number and in front of the object (toward you) using a positive number.

CenterOfRotationX moves the center of rotation along the x-axis parallel to the object while CenterOfRotationY moves the center or rotation along the y-axis of the object. The next illustrations demonstrate using different values for **CenterOfRotationY**.

XML

```
<Image Source="kid.png">
    <Image.Projection>
        <PlaneProjection RotationX="-35" CenterOfRotationY="0.5" />
    </Image.Projection>
</Image>
```

CenterOfRotationY = "0.5" (default)

XML

```
<Image Source="kid.png">
    <Image.Projection>
        <PlaneProjection RotationX="-35" CenterOfRotationY="0.1"/>
    </Image.Projection>
</Image>
```

CenterOfRotationY = "0.1"

Notice how the image rotates around the center when the **CenterOfRotationY** property is set to the default value of 0.5 and rotates near the upper edge when set to 0.1. You

see similar behavior when changing the CenterOfRotationX property to move where the RotationY property rotates the object.

XML

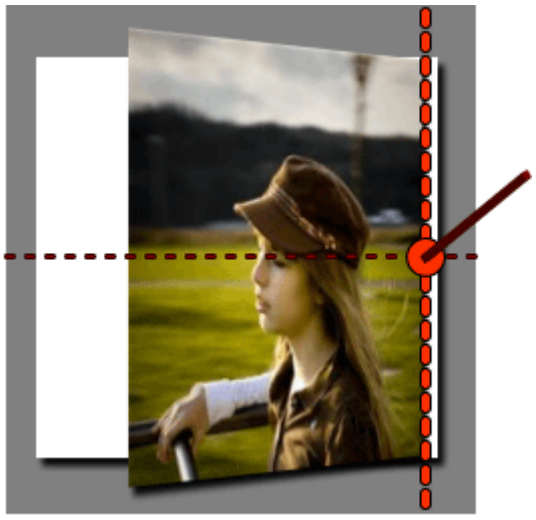
```
<Image Source="kid.png">
    <Image.Projection>
        <PlaneProjection RotationY="-35" CenterOfRotationX="0.5" />
    </Image.Projection>
</Image>
```

CenterOfRotationX = "0.5" (default)

XML

```
<Image Source="kid.png">
    <Image.Projection>
        <PlaneProjection RotationY="-35" CenterOfRotationX="0.5" />
    </Image.Projection>
</Image>
```

CenterOfRotationX = "0.9" (right-hand edge)



Use the CenterOfRotationZ to place the center of rotation above or below the plane of the object. This way, you can rotate the object around the point analogous to a planet orbiting around a star.

Positioning an object

So far, you learned how to rotate an object in space. You can position these rotated objects in space relative to one another by using these properties:

- LocalOffsetX moves an object along the x-axis of the plane of a rotated object.
- LocalOffsetY moves an object along the y-axis of the plane of a rotated object.
- LocalOffsetZ moves an object along the z-axis of the plane of a rotated object.
- GlobalOffsetX moves an object along the screen-aligned x-axis.
- GlobalOffsetY moves an object along the screen-aligned y-axis.
- GlobalOffsetZ moves an object along the screen-aligned z-axis.

Local offset

The LocalOffsetX, LocalOffsetY, and LocalOffsetZ properties translate an object along the respective axis of the plane of the object after it has been rotated. Therefore, the rotation of the object determines the direction that the object is translated. To demonstrate this concept, the next sample animates **LocalOffsetX** from 0 to 400 and RotationY from 0 to 65 degrees.

Notice in the preceding sample that the object is moving along its own x-axis. At the very beginning of the animation, when the RotationY value is near zero (parallel to the screen), the object moves along the screen in the x direction, but as the object rotates toward you, the object moves along the x-axis of the plane of the object toward you. On the other hand, if you animated the **RotationY** property to -65 degrees, the object would curve away from you.

LocalOffsetY works similarly to LocalOffsetX, except that it moves along the vertical axis, so changing RotationX affects the direction **LocalOffsetY** moves the object. In the next sample, **LocalOffsetY** is animated from 0 to 400 and **RotationX** from 0 to 65 degrees.

LocalOffsetZ translates the object perpendicular to the plane of the object as though a vector was drawn directly through the center from behind the object out toward you. To demonstrate how **LocalOffsetZ** works, the next sample animates **LocalOffsetZ** from 0 to 400 and RotationX from 0 to 65 degrees.

At the beginning of the animation, when the RotationX value is near zero (parallel to the screen), the object moves directly out toward you, but as the face of the object rotates down, the object moves down instead.

Global offset

The GlobalOffsetX, GlobalOffsetY, and GlobalOffsetZ properties translate the object along axes relative to the screen. That is, unlike the local offset properties, the axis the object moves along is independent of any rotation applied to the object. These properties are useful when you want to simply move the object along the x-, y-, or z-axis of the screen without worrying about the rotation applied to the object.

The next sample animates GlobalOffsetX from 0 to 400 and RotationY from 0 to 65 degrees.

Notice in this sample that the object does not change course as it rotates. That is because the object is being moved along the x-axis of the screen without regard to its rotation.

More complex semi-3D scenarios

You can use the Matrix3DProjection and Matrix3D types for more complex semi-3D scenarios than are possible with the PlaneProjection. **Matrix3DProjection** provides you with a complete 3D transform matrix to apply to any UIElement, so that you can apply arbitrary model transformation matrices and perspective matrices to elements. Keep in mind that these API are minimal and therefore if you use them, you will need to write the code that correctly creates the 3D transform matrices. Because of this, it is easier to use **PlaneProjection** for simple 3D scenarios.

Manage app windows (Windows App SDK)

Article • 07/09/2024

This topic contains a Code example section.

The Windows App SDK provides the easy-to-use **Microsoft.UI.Windowing.AppWindow** class. **AppWindow** is framework-agnostic, and available to all Windows apps including Win32, WPF, and WinForms. You can contrast the framework-agnostic nature of **AppWindow** to **Microsoft.UI.Xaml.Window**, which is the window class specifically for the WinUI 3 framework. **AppWindow** is also an evolution of the Universal Windows Platform's (UWP's) **Windows.UI.WindowManagement.AppWindow**.

The Windows App SDK version of **Microsoft.UI.Windowing.AppWindow** doesn't rely on asynchronous patterns; and it provides immediate feedback to your app about whether API calls have succeeded.

Also Get started with WinUI and Use the Windows App SDK in an existing project.

The AppWindow class

Microsoft.UI.Windowing.AppWindow is a high-level windowing API that allows for easy-to-use windowing scenarios. **AppWindow** integrates well with the Windows UI/UX, and with other apps.

AppWindow represents a high-level abstraction of a system-managed container for the content of an app. It's the container in which your content is hosted; and it represents the entity that users interact with when they resize and move your app on-screen. If you're familiar with Win32, the *app window* can be seen as a high-level abstraction of the **HWND**. If you're familiar with UWP, then the *app window* can be seen as a replacement for **CoreWindow**/**ApplicationView**/**Windows.UI.WindowManagement.AppWindow**.

For the Windows App SDK version of **Microsoft.UI.Windowing.AppWindow** we're supporting only top-level **HWND**s. There's a 1:1 mapping between an **AppWindow** and a top-level **HWND**.

The lifetime of an **AppWindow** object and an **HWND** is the same—the **AppWindow** is available immediately after the window has been created; and it's destroyed when the window is closed.

The AppWindowPresenter class, and subclasses

Each **AppWindow** has an **AppWindowPresenter** (presenter) applied to it. If you're a UWP developer who's worked with **Windows.UI.WindowManagement.AppWindow**, then this will be familiar; even if it's not a 1:1 mapping of functionality and behavior. Also see See Windowing functionality migration.

As a new concept to the Win32 application model, a presenter is akin to (but not the same as) a combination of window state and styles. Some presenters also have UI/UX behaviors defined in them that aren't inspectable from classic window state and style properties (such as an auto-hiding titlebar).

By default, a presenter is created by the system, and applied to an **AppWindow** at creation time. In the Windows App SDK 1.0, on Windows desktop, the type of presenter is **OverlappedPresenter**, which is a subclass of **AppWindowPresenter**. There's no need for your app to stash it, nor to keep a reference to it in order to go back to the default presenter for a window after having applied another presenter. That's because the system keeps the same instance of this presenter around for the lifetime of the **AppWindow** for which it was created; and your app can reapply it by calling the **AppWindow.SetPresenter** method with **AppWindowPresenterKind.Default** as a parameter.

A presenter can be applied to only a single window at a time. Trying to apply the same presenter to a second window throws an exception. That means that if you have multiple windows, and you want to switch each one into a specific presentation mode, then you need to create multiple presenters of the same kind, and then apply each to its own window.

Some presenters have functionality that allows a user to make changes outside of your app's own control. When such a change happens, your app is notified by an **AppWindow.Changed** event on the affected **AppWindow**, with the **AppWindowChangedEventArgs.DidPresenterChange** property set to `true`. Your app should then inspect the property of the applied presenter to see what changed.

The applied presenter is a live object. A change to any property of the **AppWindow.Presenter** object takes effect immediately.

A presenter can't be destroyed while it's applied to a window. To destroy a presenter object, first apply a different presenter object to the window; that way, the presenter that you intend to destroy is removed from the window. You can do that either by applying another specific presenter to the window, or by calling the **AppWindow.SetPresenter** method with **AppWindowPresenterKind.Default** as an argument, which will reapply the default system-created presenter to the window. If you

happened to keep a reference to the system-created presenter for the window, then it will be valid at this point (that is, the instance that was first created for the window is re-applied).

Available presenters

These **AppWindowPresenter**-derived presenters are provided, and they're available on all of the supported OS versions.

- **CompactOverlayPresenter**. Creates an *always-on-top* window of a fixed size, with a 16:9 aspect ratio to allow for *picture-in-picture*-like experiences.
- **FullScreenPresenter**. Allows a window to go into a full-screen experience.
- **OverlappedPresenter**. The system-created default presenter, which allows you to request and react to minimize/maximize/restore operations and state changes.

UI framework and HWND interop

The **AppWindow** class is available for *any* top-level **HWND** in your app. That means that when you're working with a desktop UI framework (including WinUI 3), you can continue to use that framework's entry point for creating a window, and attaching its content. And once you've created a window with that UI framework, you can use the windowing interop functions (see below) provided in the Windows App SDK to access the corresponding **AppWindow** and its methods, properties, and events.

Some of the benefits of using **AppWindow** (even when working with a UI framework) are:

- Easy title bar customization; which by default maintains the Windows 11 UI (rounded corners, snap group flyout).
- System-provided full-screen and compact overlay (picture-in-picture) experiences.
- Windows Runtime (WinRT) API surface for some of the core Win32 windowing concepts.

Code example

This code example demonstrates how to retrieve a **Microsoft.UI.Windowing.AppWindow** from a WinUI 3 window by using the **Microsoft.UI.Xaml.Window.AppWindow** property. To use the example, create a new **Blank App, Packaged (WinUI 3 in Desktop)** project, and paste the code in.

For additional details on how to work with **AppWindow**, see the Windowing gallery sample.

C#

```
// MainWindow.xaml.cs
private void myButton_Click(object sender, RoutedEventArgs e)
{
    // Retrieve the AppWindow for the current (XAML) WinUI 3 window.
    Microsoft.UI.Windowing.AppWindow appWindow = this.AppWindow;

    if (appWindow != null)
    {
        // With a non-null AppWindow object, you can call its methods
        // to manipulate the window. As an example, let's change the title
        // text of the window.
        appWindow.Title = "Title text updated via AppWindow!";
    }
}
```

Code example for versions of Windows App SDK prior to 1.3 (or other desktop app frameworks)

The **Microsoft.UI.Xaml.Window.AppWindow** property (used in the code example above) is available in Windows App SDK version 1.3 and later. For earlier versions, you can use the functionally equivalent code example in this section.

C#. .NET wrappers for the windowing interop functions are implemented as methods of the **Microsoft.UI.Win32Interop** class. Also see Call interop APIs from a .NET app.

C++. The interop functions are defined in the winrt/Microsoft.ui.interop.h header file.

The Code example section below shows actual source code; but here's the recipe for retrieving an **AppWindow** object given an existing window:

1. Retrieve the **HWND** for your existing window object (for your UI framework), if you don't already have it.
2. Pass that **HWND** to the **GetWindowIdFromWindow** interop function to retrieve a **WindowId**.
3. Pass that **WindowId** to the static **AppWindow.GetFromWindowId** method to retrieve the **AppWindow**.

C#

```
// MainWindow.xaml.cs
private void myButton_Click(object sender, RoutedEventArgs e)
{
    // Retrieve the window handle (HWND) of the current (XAML) WinUI 3
window.
    var hWnd =
        WinRT.Interop.WindowNative.GetWindowHandle(this);

    // Retrieve the WindowId that corresponds to hWnd.
    Microsoft.UI.WindowId windowId =
        Microsoft.UI.Win32Interop.GetWindowIdFromWindow(hWnd);

    // Lastly, retrieve the AppWindow for the current (XAML) WinUI 3 window.
    Microsoft.UI.Windowing.AppWindow appWindow =
        Microsoft.UI.Windowing.AppWindow.GetFromWindowId(windowId);

    if (appWindow != null)
    {
        // You now have an AppWindow object, and you can call its methods to
manipulate the window.
        // As an example, let's change the title text of the window.
        appWindow.Title = "Title text updated via AppWindow!";
    }
}
```

Limitations

- **AppWindow** is a WinUI 3 API. That means that it's available only to desktop apps (both packaged and unpackaged); and isn't available to UWP apps.
- The Windows App SDK doesn't currently provide methods for attaching UI framework content to an **AppWindow**. But see the Code example section.
- Title bar customization is supported in Windows 11 and later; and in Windows 10 for version 1.2 and later of the Windows App SDK. For details, see Title bar customization.

Related topics

- Windowing functionality migration
- Call interop APIs from a .NET app
- Retrieve a window handle (HWND)
- Windowing gallery sample

Retrieve a window handle (HWND)

Article • 08/28/2024

This topic shows you how, in a desktop app, to retrieve the window handle for a window. The scope covers WinUI 3, Windows Presentation Foundation (WPF), and Windows Forms (WinForms) apps; code examples are presented in C# and C++/WinRT.

The development and UI frameworks listed above are (behind the scenes) built on the Win32 API. In Win32, a window object is identified by a value known as a window handle. And the type of a window handle is an **HWND** (although it surfaces in C# as an **IntPtr**). In any case, you'll hear the term **HWND** used as a shorthand for *window handle*.

There are several reasons to retrieve the **HWND** for a window in your WinUI 3, WPF, or WinForms desktop app. One example is to use the **HWND** to interoperate with certain Windows Runtime (WinRT) objects that depend on a **CoreWindow** to display a user-interface (UI). For more info, see Display WinRT UI objects that depend on CoreWindow.

WinUI 3 with C#

The C# code below shows how to retrieve the window handle (HWND) for a WinUI 3 Window object. This example calls the **GetWindowHandle** method on the **WinRT.Interop.WindowNative** C# interop class. For more info about the C# interop classes, see Call interop APIs from a .NET app.

C#

```
// MainWindow.xaml.cs
private async void myButton_Click(object sender, RoutedEventArgs e)
{
    // Retrieve the window handle (HWND) of the current WinUI 3 window.
    var hWnd = WinRT.Interop.WindowNative.GetWindowHandle(this);
}
```

WinUI 3 with C++

The C++/WinRT code below shows how to retrieve the window handle (HWND) for a WinUI 3 Window object. This example calls the **IWindowNative::get_WindowHandle** method.

C++/WinRT

```
// pch.h
...
#include <microsoft.ui.xaml.window.h>

// MainWindow.xaml.cpp
void MainWindow::myButton_Click(IInspectable const&, RoutedEventArgs const&)
{
    // Retrieve the window handle (HWND) of the current WinUI 3 window.
    auto windowNative{ this->m_inner.as<::IWindowNative>() };
    HWND hWnd{ 0 };
    windowNative->get_WindowHandle(&hWnd);
}
```

WPF with C#

The C# code below shows how to retrieve the window handle (HWND) for a WPF window object. This example uses the **WindowInteropHelper** class.

C#

```
// MainWindow.xaml.cs
private void Button_Click(object sender, RoutedEventArgs e)
{
    var wih = new System.Windows.Interop.WindowInteropHelper(this);
    var hWnd = wih.Handle;
}
```

WinForms with C#

The C# code below shows how to retrieve the window handle (HWND) for a WinForms form object. This example uses the **NativeWindow.Handle** property.

C#

```
// Form1.cs
private void button1_Click(object sender, EventArgs e)
{
    var hWnd = this.Handle;
}
```

Determining the window that's hosting a visual element

From a visual element, you can access UIElement.XamlRoot; then XamlRoot.ContentIslandEnvironment; then the ContentIslandEnvironment.AppWindowId property contains the ID of the top-level Win32 HWND.

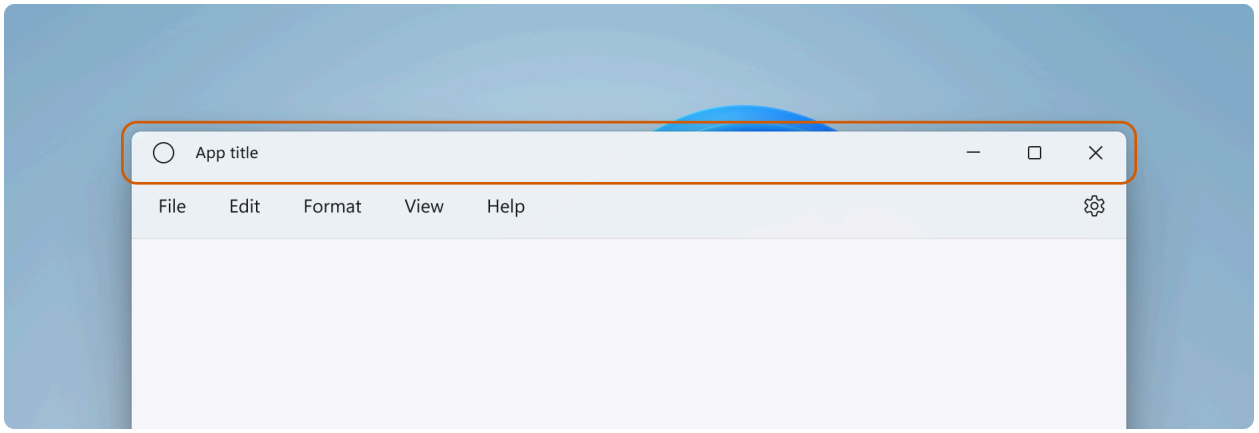
Related topics

- Display WinRT UI objects that depend on CoreWindow
- Call interop APIs from a .NET app
- WinUI 3
- Windows Presentation Foundation (WPF)
- Windows Forms (WinForms)
- C++/WinRT
- Win32 API

Title bar customization

Article • 11/21/2024

Windows provides a default title bar for every window and lets you customize it to match the personality of your app. The default title bar comes with some standard components and core functionality such as dragging and resizing the window.



See the Title bar design article for guidance on customizing your app's title bar, acceptable title bar area content, and recommended UI patterns.

See the Windows 11 Fluent Design guidance for title bar

> **Important**
>
> This article shows how to customize the title bar for apps that use the Windows App SDK, either with or without WinUI 3. For apps that use UWP and WinUI 2, see Title bar customization for UWP.

- **Applies to**: Windows App SDK, WinUI 3 desktop apps
- **Important APIs**: AppWindow.TitleBar property, AppWindowTitleBar class, AppWindow class, Microsoft.UI.Xaml.Window class, Window.ExtendsContentIntoTitleBar property, Window.SetTitleBar method

Title bar components

This list describes the components of the standard title bar.

- Title bar rectangle
- Title text
- System icon
- System menu - accessed by clicking the app icon or right-clicking the title bar

- Caption controls
  - Minimize button
  - Maximize/Restore button
  - Close button

Windowing

Windowing functionality in the Windows App SDK is through the Microsoft.UI.Windowing.AppWindow class, which is based on the Win32 HWND model. There's a 1:1 mapping between an AppWindow and a top-level HWND in your app. AppWindow and its related classes provide APIs that let you manage many aspects of your app's top-level windows, including customization of the title bar. You can modify the default title bar that Windows provides so that it blends with the rest of your UI, or extend your app canvas into the title bar area and provide your own title bar content.

Windowing functionality in WinUI 3 is through the Microsoft.UI.Xaml.Window class, which is also based on the Win32 HWND model. For XAML apps that use WinUI 3, XAML Window APIs provide a simpler way to customize the title bar, while still letting you access the AppWindow APIs when needed.

How to work with AppWindow

You can use AppWindow APIs with any UI framework that the Windows App SDK supports - Win32, WPF, WinForms, or WinUI 3 - and you can adopt them incrementally, using only the APIs you need.

If you use WinUI 3 XAML as your app's UI framework, both the Window and the AppWindow APIs are available to you. Starting in Windows App SDK 1.4, the XAML Window and AppWindow use the same AppWindowTitleBar object for title bar customization. Use the Window.AppWindow property to get an AppWindow object from an existing XAML window. With this AppWindow object you have access to the title bar customization APIs. To access additional features of the title bar, you can use the AppWindow APIs from your XAML Window like this: `AppWindow.TitleBar.ForegroundColor = Colors.White;`.

If you're not using WinUI 3 1.3 or later, use interop APIs to get the AppWindow and use the AppWindow APIs to customize the title bar. For more about the interop APIs, see Manage app windows - UI framework and HWND interop and the Windowing gallery sample.

How much to customize the title bar

There are two levels of customization that you can apply to the title bar: apply minor modifications to the default title bar, or extend your app canvas into the title bar area and provide completely custom content.

Simple

For simple customization, such as changing the title bar color, you can set properties on the AppWindowTitleBar object to specify the colors you want to use for title bar elements. In this case, the system retains responsibility for all other aspects of the title bar, such as drawing the app title and defining drag areas.

Full

Your other option is to hide the default system title bar and replace it with your own custom content. For example, you can place text, a search box, or custom menus in the title bar area. You will also need to use this option to extend a material backdrop, like Mica, into the title bar area.

When you opt for full customization, you are responsible for putting content in the title bar area, and you can define your own drag regions. The caption controls (system Close, Minimize, and Maximize buttons) are still available and handled by the system, but elements like the app title are not. You will need to create those elements yourself as needed by your app.

Simple customization

If you want only to customize the title bar title, colors, or icon, you can set properties on the title bar object for your app window.

Title

By default, the title bar shows the app type as the window title (for example, "WinUI Desktop"). You should update the window title to show a meaningful display name for your app.

A XAML app has a display name that's set in the `Package.appxmanifest` file. You can get this value and use it to set the `Title` property like this.

```
C#
```

```
Title = AppInfo.Current.DisplayInfo.DisplayName;
```

To change the window title, set the Window.Title property to a single-line text value, as shown here.

XAML

```
<Window
    ...
    Title="App title">
    ...
</Window>
```

C#

```
public MainWindow()
{
    InitializeComponent();
    Title = "App title";
}
```

To change the window title using AppWindow APIs, set the AppWindow.Title property to a single-line text value, as shown here. This example shows how to use interop APIs to get the AppWindow, which is needed of your app does not use WinUI 3 1.3 or later.

C#

```
using Microsoft.UI;           // Needed for WindowId.
using Microsoft.UI.Windowing; // Needed for AppWindow.
using WinRT.Interop;          // Needed for XAML/HWND interop.

private AppWindow m_AppWindow;

public MainWindow()
{
    this.InitializeComponent();

    m_AppWindow = GetAppWindowForCurrentWindow();
    m_AppWindow.Title = "App title";
}

private AppWindow GetAppWindowForCurrentWindow()
{
    IntPtr hWnd = WindowNative.GetWindowHandle(this);
    WindowId wndId = Win32Interop.GetWindowIdFromWindow(hWnd);
    return AppWindow.GetFromWindowId(wndId);
}
```

Colors

To customize the default title bar colors or to change the default window icon, you will need to use the AppWindow APIs or opt to fully customize your title bar.

This example shows how to get an instance of AppWindowTitleBar and set its color properties.

> **ⓘ Important**
>
> Color customization is ignored when the app runs on Windows 10.

C#

```
// Assumes "this" is a XAML Window. In projects that don't use
// WinUI 3 1.3 or later, use interop APIs to get the AppWindow.
AppWindow m_AppWindow = this.AppWindow;

private bool SetTitleBarColors()
{
    // Check to see if customization is supported.
    // The method returns true on Windows 10 since Windows App SDK 1.2,
    // and on all versions of Windows App SDK on Windows 11.
    if (AppWindowTitleBar.IsCustomizationSupported())
    {
        AppWindowTitleBar m_TitleBar = m_AppWindow.TitleBar;

        // Set active window colors.
        // Note: No effect when app is running on Windows 10
        // because color customization is not supported.
        m_TitleBar.ForegroundColor = Colors.White;
        m_TitleBar.BackgroundColor = Colors.Green;
        m_TitleBar.ButtonForegroundColor = Colors.White;
        m_TitleBar.ButtonBackgroundColor = Colors.SeaGreen;
        m_TitleBar.ButtonHoverForegroundColor = Colors.Gainsboro;
        m_TitleBar.ButtonHoverBackgroundColor = Colors.DarkSeaGreen;
        m_TitleBar.ButtonPressedForegroundColor = Colors.Gray;
        m_TitleBar.ButtonPressedBackgroundColor = Colors.LightGreen;

        // Set inactive window colors.
        // Note: No effect when app is running on Windows 10
        // because color customization is not supported.
        m_TitleBar.InactiveForegroundColor = Colors.Gainsboro;
        m_TitleBar.InactiveBackgroundColor = Colors.SeaGreen;
        m_TitleBar.ButtonInactiveForegroundColor = Colors.Gainsboro;
        m_TitleBar.ButtonInactiveBackgroundColor = Colors.SeaGreen;
        return true;
    }
```

```
    return false;
}
```

There are a few things to be aware of when setting title bar colors:

- The button background color is not applied to the close button *hover* and *pressed* states. The close button always uses the system-defined color for those states.
- Setting a color property to `null` resets it to the default system color.
- You can't set transparent colors. The color's alpha channel is ignored.

Windows gives a user the option to apply their selected accent color to the title bar. If you set any title bar color, we recommend that you explicitly set all the colors. This ensures that there are no unintended color combinations that occur because of user defined color settings.

Icon and system menu

You can hide the system icon, or replace it with a custom icon. The system icon shows the system menu when right clicked or tapped once. It closes the window when double clicked/tapped.

To show or hide the system icon and associated behaviors, set the title bar IconShowOptions property.

```
C#

m_TitleBar.IconShowOptions = IconShowOptions.HideIconAndSystemMenu;
```

To use a custom window icon, call one of the AppWindow.SetIcon methods to set the new icon.

- `SetIcon(String)`

  The SetIcon(String) method currently works only with .ico files. The string you pass to this method is the fully qualified path to the .ico file.

  ```
  C#

  m_AppWindow.SetIcon("iconPath/iconName.ico");
  ```

- `SetIcon(IconId)`

If you already have a handle to an icon (`HICON`) from one of the Icon functions like CreateIcon, you can use the GetIconIdFromIcon interop API to get an IconId. You can then pass the `IconId` to the SetIcon(IconId) method to set your window icon.

```
C#

m_AppWindow.SetIcon(iconId));
```

Full customization

When you opt-in to full title bar customization, your app's client area is extended to cover the entire window, including the title bar area. You are responsible for drawing and input-handling for the entire window except the caption buttons, which are still provided by the window.

To hide the system title bar and extend your content into the title bar area, set the property that extends app content into the title bar area to `true`. In a XAML app, this property can be set in your app's `OnLaunched` method (*App.xaml.cs*), or in your app's first page.

> **Tip**
>
> See the **Full customization example** section to see all the code at once.

This example shows how to set the Window.ExtendsContentIntoTitleBar property to `true`.

```
C#

public MainWindow()
{
    this.InitializeComponent();

    // Hide system title bar.
    ExtendsContentIntoTitleBar = true;
}
```

> **Caution**
>
> `ExtendsContentIntoTitleBar` shows in the XAML IntelliSense for `Window`, but setting it in XAML causes an error. Set this property in code instead.

This example shows how to get the AppWindowTitleBar and set the AppWindowTitleBar.ExtendsContentIntoTitleBar property to `true`. This example shows how to use interop APIs to get the AppWindow, which is needed if your app does not use WinUI 3 1.3 or later.

C#

```
using Microsoft.UI;           // Needed for WindowId.
using Microsoft.UI.Windowing; // Needed for AppWindow.
using WinRT.Interop;          // Needed for XAML/HWND interop.

private AppWindow m_AppWindow;

public MainWindow()
{
    this.InitializeComponent();

    m_AppWindow = GetAppWindowForCurrentWindow();
    var titleBar = m_AppWindow.TitleBar;
    // Hide system title bar.
    titleBar.ExtendsContentIntoTitleBar = true;
}

private AppWindow GetAppWindowForCurrentWindow()
{
    IntPtr hWnd = WindowNative.GetWindowHandle(this);
    WindowId wndId = Win32Interop.GetWindowIdFromWindow(hWnd);
    return AppWindow.GetFromWindowId(wndId);
}
```

Title bar content and the default drag region

When your app is extended into the title bar area, you're responsible for defining and managing the UI for the title bar. This typically includes, at a minimum, specifying title text and the drag region. The drag region of the title bar defines where the user can click and drag to move the window around. It's also where the user can right-click to show the system menu.

To learn more about acceptable title bar content and recommended UI patterns, see Title bar design.

This example shows the XAML for a custom title bar UI without interactive content.

XAML

```
<Grid x:Name="AppTitleBar"
      Height="32">
```

```
    <Grid.ColumnDefinitions>
        <ColumnDefinition x:Name="LeftPaddingColumn" Width="0"/>
        <ColumnDefinition/>
        <ColumnDefinition x:Name="RightPaddingColumn" Width="0"/>
    </Grid.ColumnDefinitions>
    <Image x:Name="TitleBarIcon" Source="ms-appx:///Assets/StoreLogo.png"
           Grid.Column="1"
           HorizontalAlignment="Left"
           Width="16" Height="16"
           Margin="8,0,0,0"/>
    <TextBlock x:Name="TitleBarTextBlock"
               Text="App title"
               Style="{StaticResource CaptionTextBlockStyle}"
               Grid.Column="1"
               VerticalAlignment="Center"
               Margin="28,0,0,0"/>
</Grid>
```

> **ⓘ Important**
>
> The `LeftPaddingColumn` and `RightPaddingColumn` are used to reserve space for the caption buttons. The `Width` values for these columns are set in code, which is shown later. See the ***System caption buttons*** section for the code and explanation.

A XAML app has a display name that's set in the Package.appxmanifest file. You can get this value and use it in your custom title bar like this.

C#

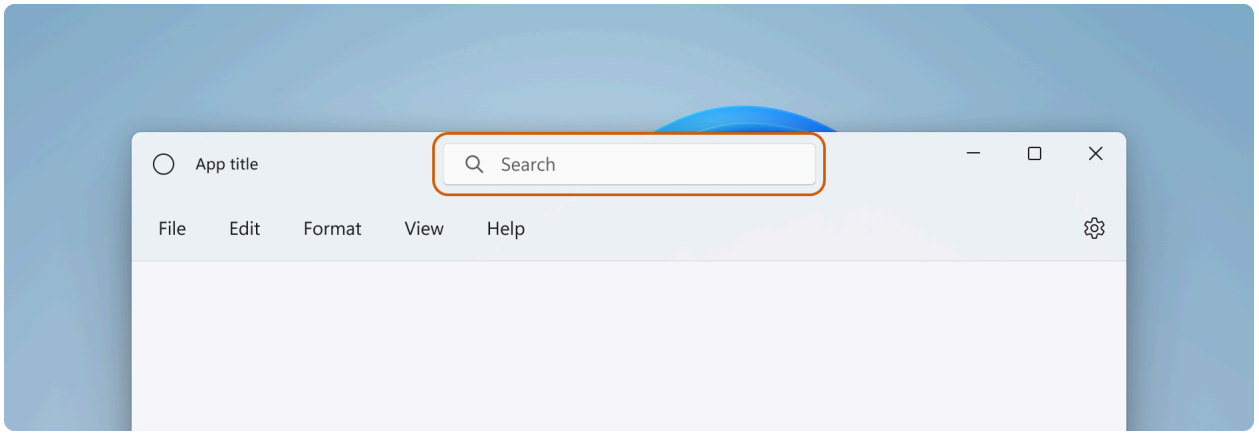
```
TitleBarTextBlock.Text = AppInfo.Current.DisplayInfo.DisplayName;
```

When you extend your content into the title bar area, the system title bar is hidden and a default AppWindowTitleBar is created that provides caption buttons and a drag region across the width of the screen, identical to the system title bar. If you don't place interactive content in your title bar, you can leave this default drag region as-is. If you place interactive content in your title bar, you need to specify the regions that are interactive, which we cover in the next section.

> **⊗ Caution**
>
> When you define custom drag regions, they don't have to be at the top of the window in the default title bar area; you can define any part of your UI as a drag region. However, placing drag regions in different places might make it difficult for your users to discover them.

Interactive content

You can place interactive controls, like buttons, menus, or a search box, in the top part of the app so they appear to be in the title bar. However, you need to specify which regions are interactive to ensure that your interactive elements receive user input while still allowing users to move your window around.



When you add interactive content in the title bar area, you need to use the InputNonClientPointerSource class to specify areas where input is passed through to the interactive control, rather than handled by the title bar. To set the interactive regions, call the InputNonClientPointerSource.SetRegionRects method. This method takes a value that specifies the kind of region being set (in this case, `Passthrough`), and an array of rectangles, each of which defines a `Passthrough` region. When the size of the title bar changes, you need to recalculate the interactive regions to match the new size, and call `SetRegionRects` with the new values.

This example shows a custom title bar UI with a search box and a PersonPicture account control. It demonstrates how to calculate and set the interactive rectangles for these controls so that input is passed through to them.

Here are a few important points to notice about this code:

- Set the `AppTitleBar` Grid height to 48 to follow the Title bar design guidance for interactive content.
- Set PreferredHeightOption to `Tall` so the caption buttons are the same height as the title bar.
- To make resizing controls and calculating regions easier, use a `Grid` with multiple named columns for the layout.
- Use *star* (*) sizing with `MinWidth` for the column that contains the `AutoSuggestBox` so that it will automatically resize with the window.
- Set `MinWidth` on the `RightDragColumn` to reserve a small area that is always draggable even when the window is resized.

- Set `ExtendsContentIntoTitleBar` to `true` in the MainWindow constructor. If you set it in code that gets called later, the default system title bar might be shown first, and then hidden.
- Make the initial call to calculate interactive regions after the `AppTitleBar` element has loaded. Otherwise, there's no guarantee that the elements used for the calculation will have their correct values.
- Update the interactive rectangle calculations only after the `AppTitleBar` element has changed size (`AppTitleBar_SizeChanged`). If you depend on the window `Changed` event, there will be situations (like window maximize/minimize) where the event occurs before `AppTitleBar` is resized and the calculations will use incorrect values.
- Set your custom drag/interactive areas only after you check `ExtendsContentIntoTitleBar` to confirm that a custom title bar is being used.

XAML

```
<Grid x:Name="AppTitleBar"
      Height="48">
    <Grid.ColumnDefinitions>
        <ColumnDefinition x:Name="LeftPaddingColumn" Width="0"/>
        <ColumnDefinition x:Name="IconColumn" Width="Auto"/>
        <ColumnDefinition x:Name="TitleColumn" Width="Auto"/>
        <ColumnDefinition x:Name="LeftDragColumn" Width="*"/>
        <ColumnDefinition x:Name="SearchColumn" Width="4*" MinWidth="220"/>
        <ColumnDefinition x:Name="RightDragColumn" Width="*" MinWidth="48"/>
        <ColumnDefinition x:Name="AccountColumn" Width="Auto"/>
        <ColumnDefinition x:Name="RightPaddingColumn" Width="0"/>
    </Grid.ColumnDefinitions>
    <Image x:Name="TitleBarIcon"
           Source="ms-appx:///Assets/StoreLogo.png"
           Grid.Column="1"
           Width="16" Height="16"
           Margin="8,0,4,0"/>
    <TextBlock x:Name="TitleBarTextBlock"
               Text="App title"
               Style="{StaticResource CaptionTextBlockStyle}"
               Grid.Column="2"
               VerticalAlignment="Center">
    </TextBlock>
    <AutoSuggestBox x:Name="TitleBarSearchBox"
                    Grid.Column="4"
                    QueryIcon="Find"
                    PlaceholderText="Search"
                    VerticalAlignment="Center"
                    MaxWidth="600"/>
    <PersonPicture x:Name="PersonPic"
                   Grid.Column="6"
```

```
                    Height="32" Margin="0,0,16,0"/>
</Grid>
```

This code shows how to calculate and set the interactive regions that correspond to the AutoSuggestBox and PersonPicture controls.

C#

```
public sealed partial class MainWindow : Window
{
    public MainWindow()
    {
        this.InitializeComponent();

        // Assumes "this" is a XAML Window. In projects that don't use
        // WinUI 3 1.3 or later, use interop APIs to get the AppWindow.
        m_AppWindow = this.AppWindow;
        AppTitleBar.Loaded += AppTitleBar_Loaded;
        AppTitleBar.SizeChanged += AppTitleBar_SizeChanged;
        ExtendsContentIntoTitleBar = true;
        TitleBarTextBlock.Text = AppInfo.Current.DisplayInfo.DisplayName;
    }

    private void AppTitleBar_Loaded(object sender, RoutedEventArgs e)
    {
        if (ExtendsContentIntoTitleBar == true)
        {
            // Set the initial interactive regions.
            SetRegionsForCustomTitleBar();
        }
    }

    private void AppTitleBar_SizeChanged(object sender, SizeChangedEventArgs
e)
    {
        if (ExtendsContentIntoTitleBar == true)
        {
            // Update interactive regions if the size of the window changes.
            SetRegionsForCustomTitleBar();
        }
    }

    private void SetRegionsForCustomTitleBar()
    {
        // Specify the interactive regions of the title bar.

        double scaleAdjustment = AppTitleBar.XamlRoot.RasterizationScale;

        RightPaddingColumn.Width = new
GridLength(m_AppWindow.TitleBar.RightInset / scaleAdjustment);
        LeftPaddingColumn.Width = new
GridLength(m_AppWindow.TitleBar.LeftInset / scaleAdjustment);
```

```
        GeneralTransform transform =
TitleBarSearchBox.TransformToVisual(null);
        Rect bounds = transform.TransformBounds(new Rect(0, 0,

TitleBarSearchBox.ActualWidth,

TitleBarSearchBox.ActualHeight));
        Windows.Graphics.RectInt32 SearchBoxRect = GetRect(bounds,
scaleAdjustment);

        transform = PersonPic.TransformToVisual(null);
        bounds = transform.TransformBounds(new Rect(0, 0,
                                                    PersonPic.ActualWidth,

PersonPic.ActualHeight));
        Windows.Graphics.RectInt32 PersonPicRect = GetRect(bounds,
scaleAdjustment);

        var rectArray = new Windows.Graphics.RectInt32[] { SearchBoxRect,
PersonPicRect };

        InputNonClientPointerSource nonClientInputSrc =
            InputNonClientPointerSource.GetForWindowId(this.AppWindow.Id);
        nonClientInputSrc.SetRegionRects(NonClientRegionKind.Passthrough,
rectArray);
    }

    private Windows.Graphics.RectInt32 GetRect(Rect bounds, double scale)
    {
        return new Windows.Graphics.RectInt32(
            _X: (int)Math.Round(bounds.X * scale),
            _Y: (int)Math.Round(bounds.Y * scale),
            _Width: (int)Math.Round(bounds.Width * scale),
            _Height: (int)Math.Round(bounds.Height * scale)
        );
    }
}
```

⚠ Warning

`AppWindow` uses physical pixels for compatibility with UI frameworks that don't use logical coordinates. If you use WPF or WinUI 3, `RightInset`, `LeftInset`, and the values used to calculate regions need to be adjusted if the display scale is not 100%. In this example, we get a `scaleAdjustment` value to account for the display scale setting.

- For WinUI 3, use the XamlRoot.RasterizationScale property to get the scale adjustment.

- For WPF, you can handle the Window.DpiChanged event to get the NewDpi value and calculate the scale adjustment.

System caption buttons

The system reserves the upper-left or upper-right corner of the app window for the system caption buttons (minimize, maximize/restore, close). The system retains control of the caption button area to guarantee that minimum functionality is provided for dragging, minimizing, maximizing, and closing the window. The system draws the Close button in the upper-right for left-to-right languages and the upper-left for right-to-left languages.

You can draw content underneath the caption control area, such as your app background, but you should not put any UI that you expect the user to be able to interact with. It does not receive any input because input for the caption controls is handled by the system.

These lines from the previous example show the padding columns in the XAML that defines the title bar. Using padding columns instead of margins ensures that the background paints the area under the caption control buttons (for transparent buttons). Using both right and left padding columns ensures that your title bar behaves correctly in both right-to-left and left-to-right layouts.

XAML

```
<Grid.ColumnDefinitions>
    <ColumnDefinition x:Name="LeftPaddingColumn" Width="0"/>
    <ColumnDefinition/>
    <ColumnDefinition x:Name="RightPaddingColumn" Width="0"/>
</Grid.ColumnDefinitions>
```

The dimensions and position of the caption control area is communicated by the AppWindowTitleBar class so that you can account for it in the layout of your title bar UI. The width of the reserved region on each side is given by the LeftInset or RightInset properties, and its height is given by the Height property.

Here's how the width of the padding columns is specified when the drag regions are calculated and set.

C#

```
RightPaddingColumn.Width =
    new GridLength(m_AppWindow.TitleBar.RightInset / scaleAdjustment);
```

```
LeftPaddingColumn.Width =
    new GridLength(m_AppWindow.TitleBar.LeftInset / scaleAdjustment);
```

> **ⓘ Important**
>
> See important information in the ***Interactive content*** section about how display scaling affects these values.

Tall title bar support for custom title bars

When you add interactive content like a search box or person picture in the title bar, we recommend that you increase the height of your title bar to provide more room for these elements. A taller title bar also makes touch manipulation easier. The AppWindowTitleBar.PreferredHeightOption property gives you the option of increasing your title bar height from the standard height, which is the default, to a taller height. When you select `Tall` title bar mode, the caption buttons that the system draws as an overlay in the client area are rendered taller with their min/max/close glyphs centered. If you haven't specified a drag region, the system will draw one that extends the width of your window and the height determined by the `PreferredHeightOption` value that you set.

This example shows how you can set the `PreferredHeightOption` property.

C#

```
// A taller title bar is only supported when drawing a fully custom title
bar.
if (ExtendsContentIntoTitleBar == true)
{
    m_AppWindow.TitleBar.PreferredHeightOption = TitleBarHeightOption.Tall;
}
```

> **⊗ Caution**
>
> The **AppWindowTitleBar.ExtendsContentIntoTitleBar** property must be `true` before you set the `PreferredHeightOption` property. If you attempt to set `PreferredHeightOption` while `ExtendsContentIntoTitleBar` is `false`, an exception is thrown.

Color and transparency in caption buttons

When you extend your app content into the title bar area, you can make the background of the caption buttons transparent to let your app background show through. You typically set the background to Colors.Transparent for full transparency. For partial transparency, set the alpha channel for the Color you set the property to.

These title bar properties can be transparent:

- ButtonBackgroundColor
- ButtonHoverBackgroundColor
- ButtonPressedBackgroundColor
- ButtonInactiveBackgroundColor

All other color properties will continue to ignore the alpha channel. If `ExtendsContentIntoTitleBar` is set to `false`, the alpha channel is always ignored for all `AppWindowTitleBar` color properties.

The button background color is not applied to the Close button *hover* and *pressed* states. The close button always uses the system-defined color for those states.

> **Tip**
>
> Mica is a delightful material that helps distinguish the window that's in focus. We recommend it as the background for long-lived windows in Windows 11. If you have applied Mica in the client area of your window, you can extend it into the title bar area and make your caption buttons transparent for the Mica to show through. See Mica material for more info.

Dim the title bar when the window is inactive

You should make it obvious when your window is active or inactive. At a minimum, you should change the color of the text, icons, and buttons in your title bar.

For XAML apps, handle the Window.Activated event to determine the activation state of the window, and update your title bar UI as needed.

C#

```
public MainWindow()
{
    ...
    Activated += MainWindow_Activated;
}

private void MainWindow_Activated(object sender, WindowActivatedEventArgs
```

```
args)
{
    if (args.WindowActivationState == WindowActivationState.Deactivated)
    {
        TitleBarTextBlock.Foreground =

(SolidColorBrush)App.Current.Resources["WindowCaptionForegroundDisabled"];
    }
    else
    {
        TitleBarTextBlock.Foreground =

(SolidColorBrush)App.Current.Resources["WindowCaptionForeground"];
    }
}
```

For other UI frameworks, handle an event to determine the activation state of the window, and update your title bar UI as needed. How you determine the state of the window depends on the UI framework you use for your app.

- **Win32**: Listen and respond to the WM_ACTIVATE message.
- **WPF**: Handle Window.Activated, Window.Deactivated.
- **WinForms**: Handle Form.Activated, Form.Deactivate.

Reset the title bar

To reset or switch to the system title bar while your app is running, you can call AppWindowTitleBar.ResetToDefault.

C#

```
m_AppWindow.TitleBar.ResetToDefault();
```

For XAML apps, you can also reset the title bar is these ways:

- Call SetTitleBar to switch to a new title bar element while your app is running.
- Call `SetTitleBar` with `null` as the parameter to reset to the default `AppWindowTitleBar` drag regions.
- Call `SetTitleBar` with `null` as the parameter and set ExtendsContentIntoTitleBar to `false` to revert to the default system title bar.

Show and hide the title bar

If you add support for *full screen* or *compact overlay* modes to your app, you might need to make changes to your title bar when your app switches between these modes. The

XAML Window does not provide any APIs to support full screen mode; you can use AppWindow APIs for this.

When your app runs in *full screen* mode, the system hides the title bar and caption control buttons. You can handle the AppWindow.Changed event and check the event args DidPresenterChange property to determine if you should show, hide, or change the title bar in response to a new window presentation.

This example shows how to handle the `Changed` event to show and hide the `AppTitleBar` element from previous examples. If the window is put in *compact overlay* mode, the title bar is reset to the default system title bar (or you could provide a custom title bar optimized for compact overlay).

C#

```
public MainWindow()
{
    this.InitializeComponent();

    m_AppWindow = this.AppWindow;
    m_AppWindow.Changed += AppWindow_Changed;
}

private void AppWindow_Changed(AppWindow sender, AppWindowChangedEventArgs
args)
{
    if (args.DidPresenterChange)
    {
        switch (sender.Presenter.Kind)
        {
            case AppWindowPresenterKind.CompactOverlay:
                // Compact overlay - hide custom title bar
                // and use the default system title bar instead.
                AppTitleBar.Visibility = Visibility.Collapsed;
                sender.TitleBar.ResetToDefault();
                break;

            case AppWindowPresenterKind.FullScreen:
                // Full screen - hide the custom title bar
                // and the default system title bar.
                AppTitleBar.Visibility = Visibility.Collapsed;
                sender.TitleBar.ExtendsContentIntoTitleBar = true;
                break;

            case AppWindowPresenterKind.Overlapped:
                // Normal - hide the system title bar
                // and use the custom title bar instead.
                AppTitleBar.Visibility = Visibility.Visible;
                sender.TitleBar.ExtendsContentIntoTitleBar = true;
                break;
```

```
            default:
                // Use the default system title bar.
                sender.TitleBar.ResetToDefault();
                break;
        }
    }
}
```

> **Note**
>
> *Full screen* and *compact overlay* modes can be entered only if supported by your app. See Manage app windows, FullScreenPresenter, and CompactOverlayPresenter for more info.

Do's and don'ts

- Do make it obvious when your window is active or inactive. At a minimum, change the color of the text, icons, and buttons in your title bar.
- Do define a drag region along the top edge of the app canvas. Matching the placement of system title bars makes it easier for users to find.
- Do define a drag region that matches the visual title bar (if any) on the app's canvas.

Full customization example

This examples shows all the code described in the Full customization section.

XAML

```
<Window
    x:Class="WinUI3_CustomTitleBar.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>

        <Grid x:Name="AppTitleBar"
      Height="48">
```

```
            <Grid.ColumnDefinitions>
                <ColumnDefinition x:Name="LeftPaddingColumn" Width="0"/>
                <ColumnDefinition x:Name="IconColumn" Width="Auto"/>
                <ColumnDefinition x:Name="TitleColumn" Width="Auto"/>
                <ColumnDefinition x:Name="LeftDragColumn" Width="*"/>
                <ColumnDefinition x:Name="SearchColumn" Width="4*"
MinWidth="220"/>
                <ColumnDefinition x:Name="RightDragColumn" Width="*"
MinWidth="48"/>
                <ColumnDefinition x:Name="AccountColumn" Width="Auto"/>
                <ColumnDefinition x:Name="RightPaddingColumn" Width="0"/>
            </Grid.ColumnDefinitions>
            <Image x:Name="TitleBarIcon"
           Source="ms-appx:///Assets/StoreLogo.png"
           Grid.Column="1"
           Width="16" Height="16"
           Margin="8,0,4,0"/>
            <TextBlock x:Name="TitleBarTextBlock"
                       Text="App title"
                       Style="{StaticResource CaptionTextBlockStyle}"
                       Grid.Column="2"
                       VerticalAlignment="Center">
            </TextBlock>
            <AutoSuggestBox x:Name="TitleBarSearchBox"
                            Grid.Column="4"
                            QueryIcon="Find"
                            PlaceholderText="Search"
                            VerticalAlignment="Center"
                            MaxWidth="600"/>
            <PersonPicture x:Name="PersonPic"
                           Grid.Column="6"
                           Height="32" Margin="0,0,16,0"/>
        </Grid>

        <NavigationView Grid.Row="1"
                        IsBackButtonVisible="Collapsed"
                        IsSettingsVisible="False">
            <StackPanel>
                <TextBlock Text="Content"
                           Style="{ThemeResource TitleTextBlockStyle}"
                           Margin="32,0,0,0"/>
                <StackPanel Grid.Row="1" VerticalAlignment="Center">
                    <Button Margin="4" x:Name="CompactoverlaytBtn"
                            Content="Enter CompactOverlay"
                            Click="SwitchPresenter"/>
                    <Button Margin="4" x:Name="FullscreenBtn"
                            Content="Enter FullScreen"
                            Click="SwitchPresenter"/>
                    <Button Margin="4" x:Name="OverlappedBtn"
                            Content="Revert to default (Overlapped)"
                            Click="SwitchPresenter"/>
                </StackPanel>
            </StackPanel>
        </NavigationView>
```

```
    </Grid>
</Window>
```

C#

```
using Microsoft.UI.Input;
using Microsoft.UI.Windowing;
using Microsoft.UI.Xaml;
using Microsoft.UI.Xaml.Controls;
using Microsoft.UI.Xaml.Media;
using System;
using Windows.ApplicationModel;
using Rect = Windows.Foundation.Rect;

public sealed partial class MainWindow : Window
{
    private AppWindow m_AppWindow;

    public MainWindow()
    {
        this.InitializeComponent();

        // Assumes "this" is a XAML Window. In projects that don't use
        // WinUI 3 1.3 or later, use interop APIs to get the AppWindow.
        m_AppWindow = this.AppWindow;
        m_AppWindow.Changed += AppWindow_Changed;
        Activated += MainWindow_Activated;
        AppTitleBar.SizeChanged += AppTitleBar_SizeChanged;
        AppTitleBar.Loaded += AppTitleBar_Loaded;

        ExtendsContentIntoTitleBar = true;
        if (ExtendsContentIntoTitleBar == true)
        {
            m_AppWindow.TitleBar.PreferredHeightOption =
TitleBarHeightOption.Tall;
        }
        TitleBarTextBlock.Text = AppInfo.Current.DisplayInfo.DisplayName;
    }

    private void AppTitleBar_Loaded(object sender, RoutedEventArgs e)
        {
            if (ExtendsContentIntoTitleBar == true)
            {
                // Set the initial interactive regions.
                SetRegionsForCustomTitleBar();
            }
        }

    private void AppTitleBar_SizeChanged(object sender, SizeChangedEventArgs
e)
        {
            if (ExtendsContentIntoTitleBar == true)
            {
```

```
                // Update interactive regions if the size of the window
changes.
                SetRegionsForCustomTitleBar();
            }
        }

    private void SetRegionsForCustomTitleBar()
    {
        // Specify the interactive regions of the title bar.

        double scaleAdjustment = AppTitleBar.XamlRoot.RasterizationScale;

        RightPaddingColumn.Width = new
GridLength(m_AppWindow.TitleBar.RightInset / scaleAdjustment);
        LeftPaddingColumn.Width = new
GridLength(m_AppWindow.TitleBar.LeftInset / scaleAdjustment);

        // Get the rectangle around the AutoSuggestBox control.
        GeneralTransform transform =
TitleBarSearchBox.TransformToVisual(null);
        Rect bounds = transform.TransformBounds(new Rect(0, 0,

TitleBarSearchBox.ActualWidth,

TitleBarSearchBox.ActualHeight));
        Windows.Graphics.RectInt32 SearchBoxRect = GetRect(bounds,
scaleAdjustment);

        // Get the rectangle around the PersonPicture control.
        transform = PersonPic.TransformToVisual(null);
        bounds = transform.TransformBounds(new Rect(0, 0,
                                                    PersonPic.ActualWidth,

PersonPic.ActualHeight));
        Windows.Graphics.RectInt32 PersonPicRect = GetRect(bounds,
scaleAdjustment);

        var rectArray = new Windows.Graphics.RectInt32[] { SearchBoxRect,
PersonPicRect };

        InputNonClientPointerSource nonClientInputSrc =
            InputNonClientPointerSource.GetForWindowId(this.AppWindow.Id);
        nonClientInputSrc.SetRegionRects(NonClientRegionKind.Passthrough,
rectArray);
    }

    private Windows.Graphics.RectInt32 GetRect(Rect bounds, double scale)
    {
        return new Windows.Graphics.RectInt32(
            _X: (int)Math.Round(bounds.X * scale),
            _Y: (int)Math.Round(bounds.Y * scale),
            _Width: (int)Math.Round(bounds.Width * scale),
            _Height: (int)Math.Round(bounds.Height * scale)
        );
    }
```

```
    private void MainWindow_Activated(object sender,
WindowActivatedEventArgs args)
    {
        if (args.WindowActivationState == WindowActivationState.Deactivated)
        {
            TitleBarTextBlock.Foreground =

(SolidColorBrush)App.Current.Resources["WindowCaptionForegroundDisabled"];
        }
        else
        {
            TitleBarTextBlock.Foreground =

(SolidColorBrush)App.Current.Resources["WindowCaptionForeground"];
        }
    }

    private void AppWindow_Changed(AppWindow sender,
AppWindowChangedEventArgs args)
    {
        if (args.DidPresenterChange)
        {
            switch (sender.Presenter.Kind)
            {
                case AppWindowPresenterKind.CompactOverlay:
                    // Compact overlay - hide custom title bar
                    // and use the default system title bar instead.
                    AppTitleBar.Visibility = Visibility.Collapsed;
                    sender.TitleBar.ResetToDefault();
                    break;

                case AppWindowPresenterKind.FullScreen:
                    // Full screen - hide the custom title bar
                    // and the default system title bar.
                    AppTitleBar.Visibility = Visibility.Collapsed;
                    sender.TitleBar.ExtendsContentIntoTitleBar = true;
                    break;

                case AppWindowPresenterKind.Overlapped:
                    // Normal - hide the system title bar
                    // and use the custom title bar instead.
                    AppTitleBar.Visibility = Visibility.Visible;
                    sender.TitleBar.ExtendsContentIntoTitleBar = true;
                    break;

                default:
                    // Use the default system title bar.
                    sender.TitleBar.ResetToDefault();
                    break;
            }
        }
    }

    private void SwitchPresenter(object sender, RoutedEventArgs e)
```

```
    {
        if (AppWindow != null)
        {
            AppWindowPresenterKind newPresenterKind;
            switch ((sender as Button).Name)
            {
                case "CompactoverlaytBtn":
                    newPresenterKind =
AppWindowPresenterKind.CompactOverlay;
                    break;

                case "FullscreenBtn":
                    newPresenterKind = AppWindowPresenterKind.FullScreen;
                    break;

                case "OverlappedBtn":
                    newPresenterKind = AppWindowPresenterKind.Overlapped;
                    break;

                default:
                    newPresenterKind = AppWindowPresenterKind.Default;
                    break;
            }

            // If the same presenter button was pressed as the
            // mode we're in, toggle the window back to Default.
            if (newPresenterKind == AppWindow.Presenter.Kind)
            {
                AppWindow.SetPresenter(AppWindowPresenterKind.Default);
            }
            else
            {
                // Else request a presenter of the selected kind
                // to be created and applied to the window.
                AppWindow.SetPresenter(newPresenterKind);
            }
        }
    }
}
```

Related articles

- Title bar customization (UWP)
- Acrylic
- Mica
- Color

Display WinRT UI objects that depend on CoreWindow

Article • 03/01/2023

Certain pickers, popups, dialogs, and other Windows Runtime (WinRT) objects depend on a CoreWindow; typically to display a user-interface (UI). Even though **CoreWindow** isn't supported in desktop apps (see Core unsupported classes), you can still use many of those WinRT classes in your desktop app by adding a little bit of interoperation code.

Your desktop app can be WinUI 3, Windows Presentation Foundation (WPF), or Windows Forms (WinForms) apps. Code examples are presented in C# and C++/WinRT.

Set the owner window handle (HWND) for a WinRT UI object

For classes that implement the **IInitializeWithWindow** interface (or the equivalent **IDataTransferManagerInterop** interface), you can use that interface to set an owner window on the object before you display it. It's a two-step process.

1. Decide which window will be the owner of the UI object that you want to display, and retrieve that window's HWND. For more details and code examples for this step, see Retrieve a window handle (HWND).
2. Then call the appropriate interoperatability API (for C# or C++/WinRT) to set an owner window handle (HWND) for the WinRT UI object.

For classes that implement IInitializeWithWindow

These classes implement **IInitializeWithWindow**:

- **Windows.ApplicationModel.Contacts.PinnedContactManager**
- **Windows.ApplicationModel.Payments.PaymentMediator**
- **Windows.Devices.Enumeration.DevicePicker**
- **Windows.Graphics.Capture.GraphicsCapturePicker**
- **Windows.Media.Casting.CastingDevicePicker**
- **Windows.Media.DialProtocol.DialDevicePicker**
- **Windows.Networking.NetworkOperators.ProvisioningAgent**
- **Windows.Security.Authentication.OnlineId.OnlineIdAuthenticator**

- Windows.Services.Store.StoreContext
- Windows.Storage.Pickers.FileOpenPicker
- Windows.Storage.Pickers.FileSavePicker
- Windows.Storage.Pickers.FolderPicker
- Windows.System.FolderLauncherOptions—Windows 10, version 1903 (10.0; Build 18362) and later
- Windows.System.LauncherOptions—Windows 10, version 1903 (10.0; Build 18362) and later
- Windows.UI.Core.CoreWindowDialog
- Windows.UI.Core.CoreWindowFlyout
- Windows.UI.Popups.MessageDialog. But for new apps we recommend using the ContentDialog control instead.
- Windows.UI.Popups.PopupMenu
- Windows.UI.StartScreen.SecondaryTile
- Windows.Web.Http.Filters.HttpBaseProtocolFilter

> ⓘ Note
>
> The list above is necessarily incomplete—refer to a type's documentation to see whether it implements **IInitializeWithWindow** (or an equivalent interop interface).

The next sections contain code examples to display a FolderPicker. But it's the same technique to display any of the APIs listed above.

WinUI 3 with C# (also WPF/WinForms with .NET 6 or later)

> ⓘ Note
>
> The code examples in this section use the **WinRT.Interop.WindowNative** C# interop class. If you target .NET 6 or later, then you can use that class in a WPF or WinForms project. For info about setting up your project to do that, see Call interop APIs from a .NET app.

The C# code below expects that you've already used the pattern documented in Retrieve a window handle (HWND). Then, to set the owner window for the UI object that you want to display, the code calls the **Initialize** method on the **WinRT.Interop.InitializeWithWindow** C# interop class. For more info about the C# interop classes, see Call interop APIs from a .NET app.

C#

```
// MainWindow.xaml.cs
private async void ShowFolderPickerAsync(IntPtr hWnd)
{
    // Create a folder picker.
    var folderPicker = new Windows.Storage.Pickers.FolderPicker();

    // Initialize the folder picker with the window handle (HWND).
    WinRT.Interop.InitializeWithWindow.Initialize(folderPicker, hWnd);

    // Use the folder picker as usual.
    folderPicker.FileTypeFilter.Add("*");
    var folder = await folderPicker.PickSingleFolderAsync();
}
```

WinUI 3 with C++

The C++/WinRT code below expects that you've already used the pattern documented in Retrieve a window handle (HWND). Then, to set the owner window for the UI object that you want to display, the code calls the interoperatability method **IInitializeWithWindow::Initialize**.

C++/WinRT

```
// pch.h
...
#include <microsoft.ui.xaml.window.h>
#include <Shobjidl.h>
#include <winrt/Windows.Storage.Pickers.h>

// MainWindow.xaml.cpp
winrt::fire_and_forget ShowFolderPickerAsync(HWND hWnd)
{
    // Create a folder picker.
    Windows::Storage::Pickers::FolderPicker folderPicker;

    // Initialize the folder picker with the window handle (HWND).
    auto initializeWithWindow{ folderPicker.as<::IInitializeWithWindow>() };
    initializeWithWindow->Initialize(hWnd);

    // Use the folder picker as usual.
    folderPicker.FileTypeFilter().Append(L"*");
    auto folder{ co_await folderPicker.PickSingleFolderAsync() };
}
```

For classes that implement IDataTransferManagerInterop

The **Windows.ApplicationModel.DataTransfer.DataTransferManager** class implements the **IDataTransferManagerInterop** interface (which, like **IInitializeWithWindow**, lets you set an owner window).

In a desktop app, instead of calling the **DataTransferManager.ShowShareUI** method, you call **IDataTransferManagerInterop::ShowShareUIForWindow**, as shown in the code examples below.

WinUI 3 with C# (also WPF/WinForms with .NET 6 or later)

C#

```
// MainWindow.xaml.cs
...
public sealed partial class MainWindow : Window
{
    ...

    [System.Runtime.InteropServices.ComImport]
    [System.Runtime.InteropServices.Guid("3A3DCD6C-3EAB-43DC-BCDE-
45671CE800C8")]
    [System.Runtime.InteropServices.InterfaceType(

System.Runtime.InteropServices.ComInterfaceType.InterfaceIsIUnknown)]
    interface IDataTransferManagerInterop
    {
        IntPtr GetForWindow([System.Runtime.InteropServices.In] IntPtr
appWindow,
            [System.Runtime.InteropServices.In] ref Guid riid);
        void ShowShareUIForWindow(IntPtr appWindow);
    }

    static readonly Guid _dtm_iid =
        new Guid(0xa5caee9b, 0x8708, 0x49d1, 0x8d, 0x36, 0x67, 0xd2, 0x5a,
0x8d, 0xa0, 0x0c);

    private void myButton_Click(object sender, RoutedEventArgs e)
    {
        // Retrieve the window handle (HWND) of the current WinUI 3 window.
        var hWnd = WinRT.Interop.WindowNative.GetWindowHandle(this);

        IDataTransferManagerInterop interop =
        Windows.ApplicationModel.DataTransfer.DataTransferManager.As
            <IDataTransferManagerInterop>();
```

```
        IntPtr result = interop.GetForWindow(hWnd, _dtm_iid);
        var dataTransferManager = WinRT.MarshalInterface

<Windows.ApplicationModel.DataTransfer.DataTransferManager>.FromAbi(result);

        dataTransferManager.DataRequested += (sender, args) =>
        {
            args.Request.Data.Properties.Title = "In a desktop app...";
            args.Request.Data.SetText("...display WinRT UI objects that
depend on CoreWindow.");
            args.Request.Data.RequestedOperation =

Windows.ApplicationModel.DataTransfer.DataPackageOperation.Copy;
        };

        // Show the Share UI
        interop.ShowShareUIForWindow(hWnd);
    }
}
...
```

WinUI 3 with C++

C++/WinRT

```
// pch.h in a Windows App SDK app
...
#include <shobjidl_core.h>
#include <microsoft.ui.xaml.window.h>
#include <winrt/Windows.ApplicationModel.DataTransfer.h>
...

// MainWindow.xaml.cpp
...
void MainWindow::myButton_Click(IInspectable const&, RoutedEventArgs const&)
{
    // Retrieve the window handle (HWND) of the current WinUI 3 window.
    auto windowNative{ this->m_inner.as<::IWindowNative>() };
    HWND hWnd{ 0 };
    windowNative->get_WindowHandle(&hWnd);

    winrt::com_ptr<IDataTransferManagerInterop> interop =

winrt::get_activation_factory<Windows::ApplicationModel::DataTransfer::DataT
ransferManager,
        IDataTransferManagerInterop>();

    winrt::guid _dtm_iid{ 0xa5caee9b, 0x8708, 0x49d1, { 0x8d, 0x36, 0x67,
0xd2, 0x5a, 0x8d, 0xa0, 0x0c } };
    Windows::ApplicationModel::DataTransfer::DataTransferManager
dataTransferManager{ nullptr };
    interop->GetForWindow(hWnd, _dtm_iid,
```

```
winrt::put_abi(dataTransferManager));

    dataTransferManager.DataRequested([]
(Windows::ApplicationModel::DataTransfer::DataTransferManager const& /*
sender */,
        Windows::ApplicationModel::DataTransfer::DataRequestedEventArgs
const& args)
    {
        args.Request().Data().Properties().Title(L"In a desktop app...");
        args.Request().Data().SetText(L"...display WinRT UI objects that
depend on CoreWindow.");

args.Request().Data().RequestedOperation(Windows::ApplicationModel::DataTran
sfer::DataPackageOperation::Copy);
    });

    interop->ShowShareUIForWindow(hWnd);
}
...
```

For classes that implement IUserConsentVerifierInterop

The **Windows.Security.Credentials.UI.UserConsentVerifier** class implements the **IUserConsentVerifierInterop** interface (which, like **IInitializeWithWindow**, lets you set an owner window).

In a desktop app, instead of calling the **UserConsentVerifier.RequestVerificationAsync** method:

- **C#**. Call the **RequestVerificationForWindowAsync** method of the **Windows.Security.Credentials.UI.UserConsentVerifierInterop** C# interop class. For more info about the C# interop classes, see Call interop APIs from a .NET app.
- **C++/WinRT**. Call **IUserConsentVerifierInterop::RequestVerificationForWindowAsync**.

For more info, and code examples, see **UserConsentVerifier**.

For classes that implement other interop interfaces

These interfaces have **XxxForWindow** methods, which let you set an owner window handle (HWND). You can use these interfaces directly from C++/WinRT. Versions of the

interfaces also exist in the form of C# classes—for more details, see Call interop APIs from a .NET app.

- IAccountsSettingsPaneInterop
- IDragDropManagerInterop
- IInputPaneInterop
- IPlayToManagerInterop
- IPrintManagerInterop
- IRadialControllerConfigurationInterop
- IRadialControllerIndependentInputSourceInterop
- IRadialControllerInterop
- ISpatialInteractionManagerInterop
- ISystemMediaTransportControlsInterop
- IUIViewSettingsInterop
- IWebAuthenticationCoreManagerInterop

Related topics

- Retrieve a window handle (HWND)
- WinUI 3
- Windows Presentation Foundation (WPF)
- Windows Forms (WinForms)
- C++/WinRT
- Call interop APIs from a .NET app

Choose the right visual tree viewer for your Windows app

Article • 03/03/2025

A visual tree viewer, also known as a UI visualizer, is a tool used to inspect and interact with UI components in a Windows app at run time.

This can be helpful for prototyping, improving user experiences, and debugging UI issues through capabilities such as viewing and traversing component hierarchy, component highlighting, getting and setting state, and deep-linking to associated code.

Recommended tools

The following table identifies several UI visualization tools and the UI frameworks and Windows app platforms they support. A summary of each tool can be found after the table.

Expand table

UI platform/framework	Visual Studio - Live Visual Tree	Spy++	Accessibility Insights	Chromium UI DevTools
WinUI in the Windows App SDK	**Supported**	Not supported	**Supported**	Not supported
WPF	**Supported**	Not supported	**Supported**	Not supported
React Native for Desktop	**Supported**	Not supported	**Supported**	**Supported**
.NET MAUI	**Supported**	Not supported	**Supported**	Not supported
WinForms	**Supported**	**Supported**	**Supported**	Not supported
WinUI 2 for UWP	**Supported**	Not supported	**Supported**	Not supported
Classic Visual Basic apps	Not supported	**Supported**	Not supported	Not supported
Classic Win32 apps	Not supported	**Supported**	Not supported	Not supported
Chromium-based apps	Not supported	Not supported	Not supported	**Supported**

Visual Studio - Live Visual Tree

The Live Visual Tree and Live Property Explorer features ship with Visual Studio and work in tandem to provide an interactive runtime view of the UI elements in your app.

When to use Live Visual Tree

Use these tools when building apps with WinUI in the Windows App SDK, WinUI 2 for UWP, WPF, .NET MAUI, WinForms, or React Native for Desktop.

- For more information on WinUI in the Windows App SDK, WinUI 2 for UWP, and WPF, see Inspect XAML properties while debugging.
- For more information on .NET MAUI, see Inspect the visual tree of a .NET MAUI app.

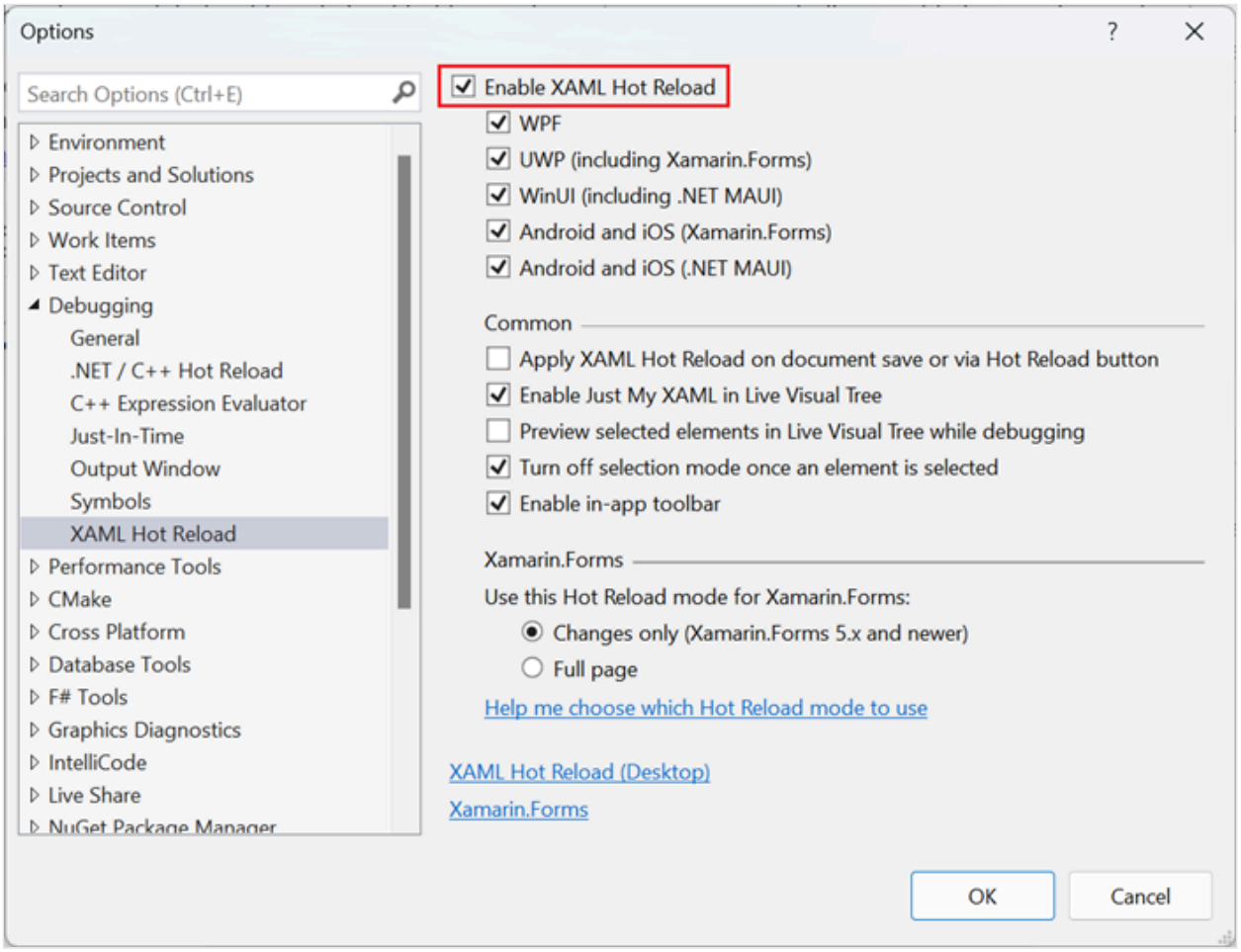
> **Note**
>
> The **WPF Tree Visualizer** is a legacy feature and is not in active development. You can use the WPF Tree visualizer to explore the visual tree of a WPF object, and to view the WPF dependency properties for the objects that are contained in that tree.

How to use Live Visual Tree

XAML Hot Reload must be enabled to view the Live Visual Tree (this feature is enabled by default in Visual Studio 2019 and later).

To check if XAML Hot Reload is enabled, run your app with the Visual Studio debugger attached (**F5** or **Debug -> Start Debugging**). When the app starts, you should see the in-app toolbar, which confirms that XAML Hot Reload is available (as shown in the following image).

If you don't see the in-app toolbar, select **Debug** > **Options** > **XAML Hot Reload** from the Visual Studio menu bar. In the **Options** dialog box, make sure that the **Enable XAML Hot Reload** option is selected.



Once your app is running in debug configuration (with the debugger attached), navigate to the Visual Studio menu bar (**Debug** > **Windows** > **Live Visual Tree**). This opens the Live Visual Tree pane with a real-time view of your XAML code.

By default, the **Just My XAML** option for the Live Visual Tree is selected. This provides a simplified view of the XAML element collection in your app and can be toggled on or off through the **Show Just My XAML** button as shown in the following image.

Capabilities of Live Visual Tree

The Live Visual Tree toolbar provides several options for selecting elements to examine through your application's UI at runtime.

- **Select Element in the Running Application**. With this mode on, when you select a UI element in the application, the Live Visual Tree automatically updates to show the node and it's properties in the tree corresponding to that element.

- **Display Layout Adorners in the Running Application**. With this mode on, the application window shows horizontal and vertical lines along the bounds of a selected object and a rectangle outlining its margins.

- **Preview selection**. With this mode on, Visual Studio shows the XAML where the element is declared (if you have access to the application source code).

Spy++

Spy++ (SPYXX.EXE) is a Win32-based utility that ships with Visual Studio and provides a graphical view of the system's processes, threads, windows, and window messages.

When to use Spy++

Use Spy++ when building a classic Win32 application or one that uses Win32 APIs to draw its UI elements, such as WinForms and Classic Visual Basic apps.

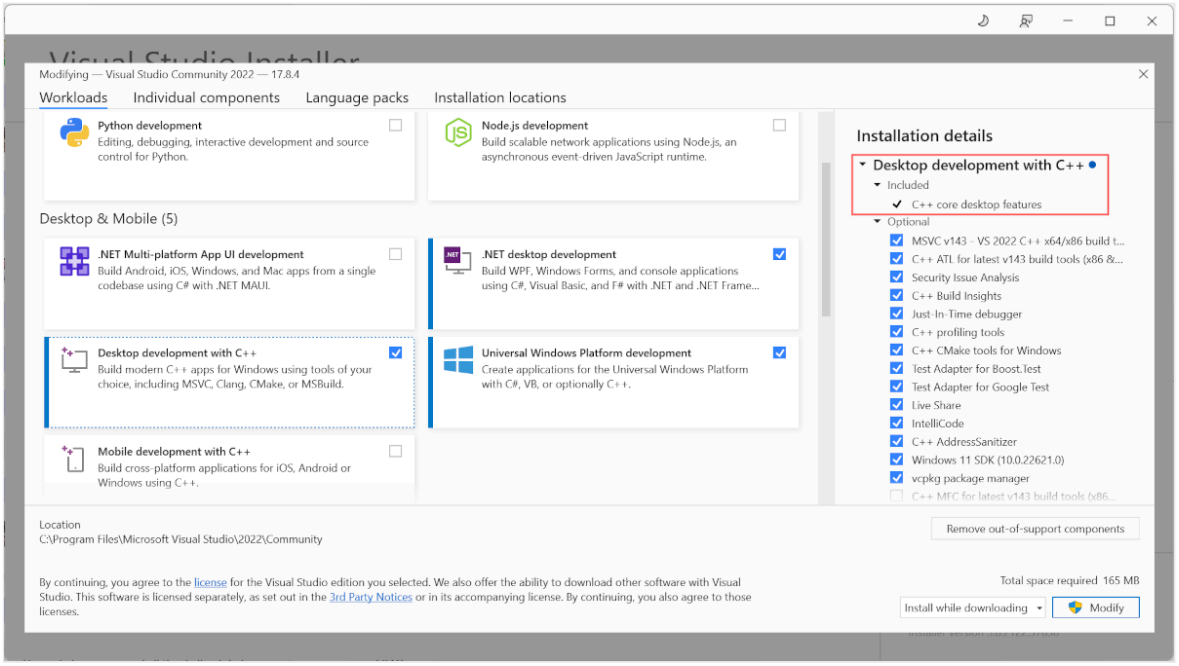
> **Note**
>
> For .NET framework apps, Spy++ is of limited usefulness as the window messages and classes intercepted by Spy++ don't correspond to managed events and property values.

How to use Spy++

Spy++ can be started from either Visual Studio or the Developer Command Prompt for Visual Studio.

To start Spy++ from Visual Studio:

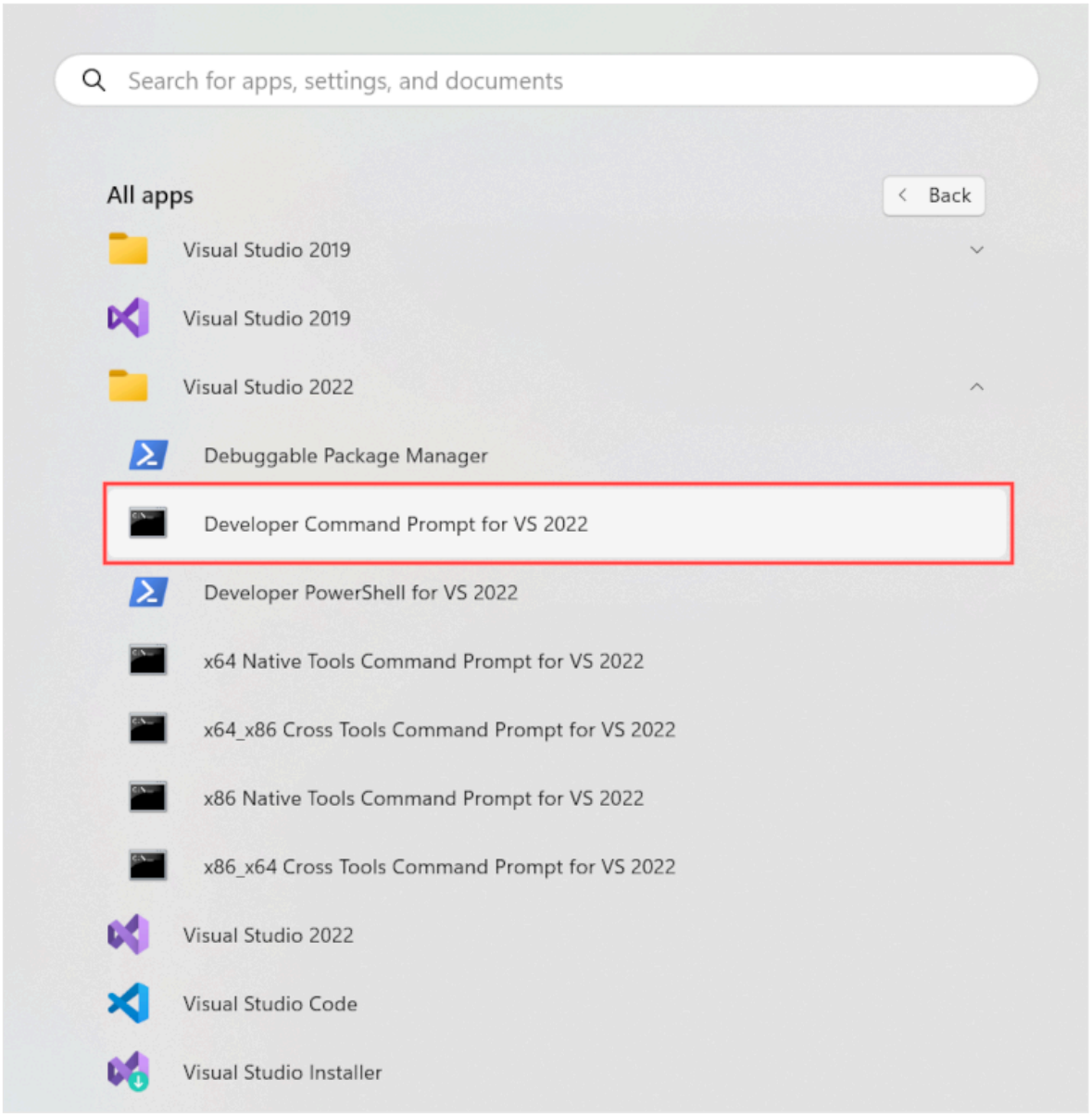
- Confirm that your Visual Studio installation includes the **C++ core desktop features** component from the **Desktop development with C++** workload. (This is installed by default with Visual Studio 2022.)



- When installed, Spy++ is available from the **Tools** menu.
- Spy++ runs independently of Visual Studio, which can be closed if no longer required.

To start Spy++ from the Developer Command Prompt for Visual Studio:

- Launch **Developer Command Prompt for Visual Studio** from the Windows **Start** menu.

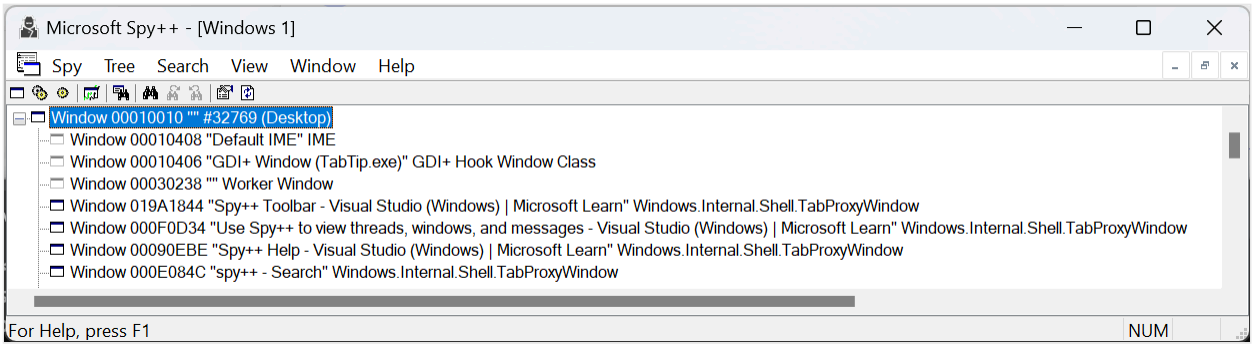


- At the command prompt, enter spyxx.exe (or spyxx_amd64.exe for the 64-bit version) and press Enter.

For more specific information on how to use Spy++ from Visual Studio, see Spy++ Toolbar.

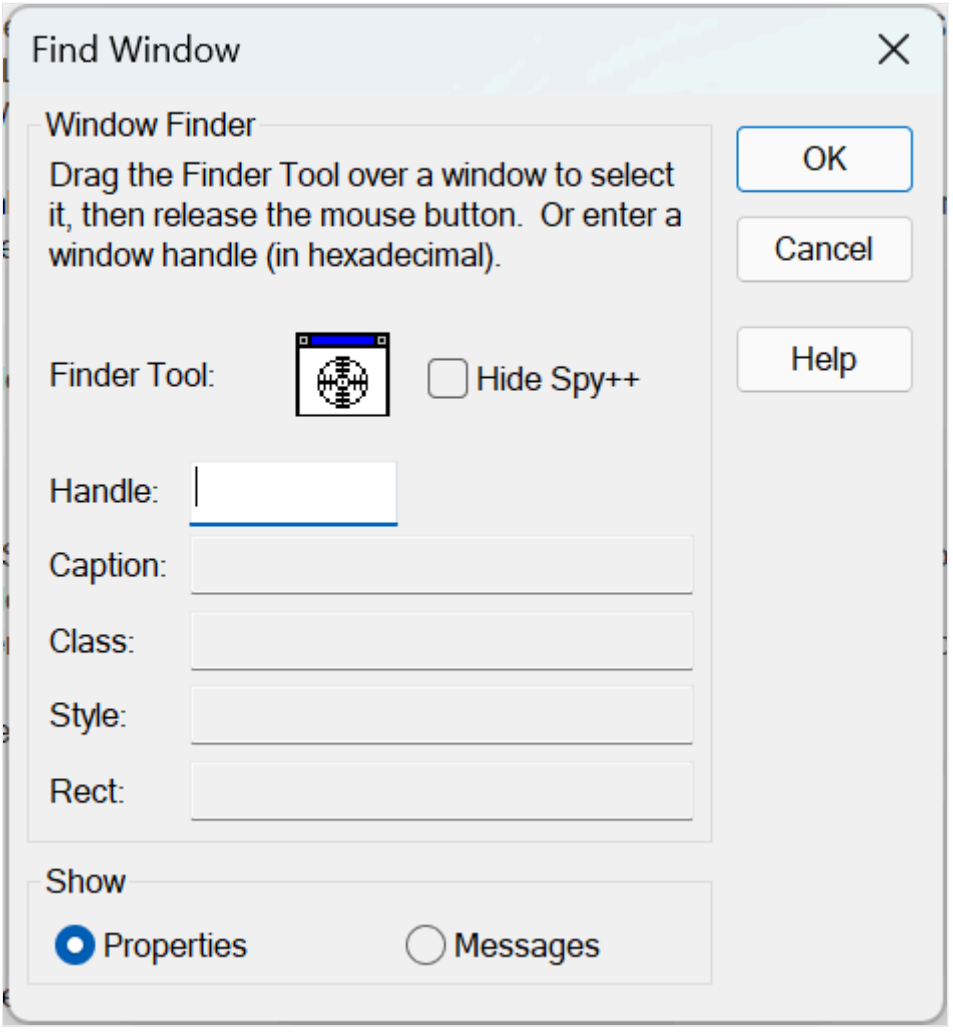
Capabilities of Spy++

Spy++ displays a graphical tree of relationships among system objects, with the current desktop window at the top of the tree and child nodes representing all other windows listed according to the standard window hierarchy. It can provide valuable insight into an application's behavior that is not available through the Visual C++ debugger.



With Spy++ you can:

- Search for specific windows, threads, processes, or messages.
- View the properties of selected threads processes or messages.
- Select a window, thread, process, or message directly in the tree view.
- Use the **Finder Tool** to select a window using the mouse pointer (**Search** -> **Find Window**).



- Set a message option by using a complex message log selection parameter.

For Spy++ documentation, see Spy++ Help.

Accessibility Insights for Windows - Live Inspect

Accessibility Insights for Windows - Live Inspect is a downloadable Microsoft application that can help developers find and fix accessibility issues in Windows apps that support UI Automation. It helps developers verify that an element in an app has the correct UI Automation properties simply by hovering over the element or setting keyboard focus to it.

When to use Accessibility Insights - Live Inspect

Live Inspect is typically used in conjunction with Live Visual Tree, Spy++, and other tools when building apps with WinUI in the Windows App SDK, WinUI 2 for UWP, WPF, .NET MAUI, WinForms, or React Native for Desktop.

How to use Accessibility Insights - Live Inspect

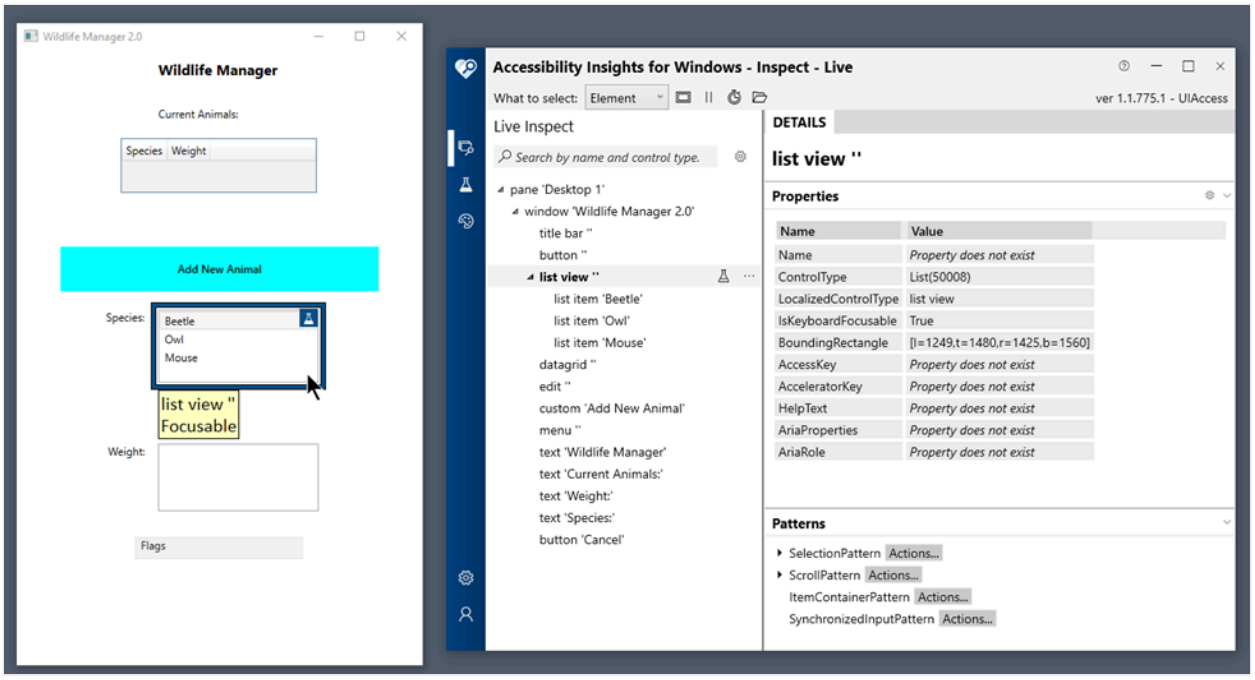
Accessibility Insights must be downloaded and installed from Download Accessibility Insights.

Capabilities of Accessibility Insights - Live Inspect

Live Inspect displays a graphical tree of relationships among system objects with detail panes containing the UI Automation properties and patterns corresponding to the selected element. The current desktop window is displayed at the top of the tree and child nodes representing all other windows listed according to the standard window hierarchy.

With Live Inspect you can:

- Verify that an element in an app has the right UI Automation properties simply by hovering over the element or setting keyboard focus on it.
- Visually highlights elements in the target application.
- Test controls or patterns with manual or automated checks for compliance with numerous accessibility rules and guidelines.



To learn more about UI Automation, see UI Automation Overview.

To learn more about Accessibility Insights, see Accessibility Insights for Windows

Chromium UI DevTools for Windows

Chromium UI DevTools for Windows is a tool from Google that lets you inspect the UI system like a webpage using the frontend DevTools Inspector.

When to use Chromium UI DevTools for Windows

Use Chrome UI DevTools if you're developing a Chromium project, including progressive web apps or Electron desktop apps. For more information on Electron, see the DevTools extension on GitHub.

How to use Chromium UI DevTools for Windows

The Chromium UI DevTools for Windows codebase must be downloaded from GitHub and built with Visual Studio. For more information, see the UI DevTools Overview.

Capabilities of Chromium UI DevTools for Windows

Chromium UI DevTools for Windows uses a webpage front end to display and traverse views, windows, and other UI elements.

With Chromium UI DevTools for Windows you can:

- Use a hierarchical UI element tree to inspect UI elements and their properties.
- Use Inspect mode, which highlights a UI element when you hover over it and reveals the element's nodes in the UI element tree.
- Use the Elements panel to open a search bar and find an element in the UI element tree using name, tag, and style properties.
- Use the Sources panel to open the class header file in Chromium code search and pull in the code from local files.

Related content

An overview of Package Identity in Windows apps

Article • 02/10/2023

Package identity is a unique identifier across space and time. Just as your DNA uniquely identifies you, package identity uniquely identifies a package.

A package has an associated set of bits (files etc). No two packages have the same identity, and any changes to the bits associated with a package requires a different identity.

What is package identity?

A package identity is a logical construct, uniquely identifying a package. The identity has 5 parts:

- **Name:** This is a name chosen by the app developer. The Microsoft Store enforces uniqueness of all app names across all app developers within the Store, but names are not guaranteed to be unique in the general ecosystem.
- **Version:** Version number of the package. The app developer can choose arbitrary version numbers but must ensure version numbers increase with updates.
- **Architecture:** The processor architecture being targeted by the package. The same app can be built targeting different processor architectures, with each build residing in its own package.
- **ResourceId:** A string chosen by the app developer to uniquely identify resource packages, for example different languages or different display scales. Resource packages are typically architecture-neutral. For bundles, the **ResourceId** is always `~`.
- **Publisher:** The app developer's subject name as identified by their signing certificate. This is theoretically unique for each app developer, because reputable certification authorities use unique real-world names and identities to populate the certificate's subject name field.

This construct is sometimes referred to as the *5-part tuple*.

> **Note**
>
> **Unsigned packages** (1) still require a **Publisher**, (2) the **Publisher** must contain the Unsigned marker (OID.2.25.311729368913984317654407730594956997722=1), (3)

the Unsigned marker *must* be the last field in the **Publisher** string, and (4) there is no certificate or signing for an Unsigned package.

Package identity fields limits

Field	Data type	Limits	Comments
Name	Package String	Min: 3 Max: 50	Allowed values per Validate API (see Package String)
Version	DotQuad	Min: 0.0.0.0 Max: 65535.65535.65535.65535	String form uses base-10 dotted notation, "Major.Minor.Build.Revision"
Architecture	Enumeration	Min: N/A Max: N/A	Allowed values are "neutral", "x86", "x64", "arm", "arm64", "x86a64"
ResourceId	Package String	Min: 0 Max: 30	Allowed values per Validate API (see Package String)
Publisher	String	Min: 1 Max: 8192	Allowed values per X.509
PublisherId	String	Min: 13 Max: 13	Base32 encoded, Crockford variant, i.e. [a-hjkmnp-tv-z0-9]

What is a 'Package String'?

A **package string** is a string that allows the following characters:

- Allowed Input Characters (ASCII subset)
  - Uppercase letters (U+0041 thru U+005A)
  - Lowercase letters (U+0061 thru U+007A)
  - Numbers (U+0030 thru U+0039)
  - Dot (U+002E)
  - Dash (U+002D)

The following values are prohibited from being used as package strings:

Condition	Prohibited values
Cannot equal	".", "..", "con", "prn", "aux", "nul", "com1", "com2", "com3", "com4", "com5", "com6", "com7", "com8", "com9", "lpt1", "lpt2", "lpt3", "lpt4", "lpt5", "lpt6", "lpt7", "lpt8", "lpt9"

Condition	Prohibited values
Cannot begin with	"con.", "prn.", "aux.", "nul.", "com1.", "com2.", "com3.", "com4.", "com5.", "com6.", "com7.", "com8.", "com9.", "lpt1.", "lpt2.", "lpt3.", "lpt4.", "lpt5.", "lpt6.", "lpt7.", "lpt8.", "lpt9.", "xn--"
Cannot end with	"."
Cannot contain	".xn--"

A **package string** must be compared using an ordinal case-insensitive string comparison APIs (e.g. _wcsicmp).

Package Identity's `name` and `resourceid` fields are package strings.

PackageId object

A PackageId is an object containing the 5-part tuple as individual fields (`Name`, `Version`, `Architecture`, `ResourceId`, `Publisher`).

Package Full Name

A **Package Full Name** is an opaque string derived from all 5 part of a package's identity (name, version, architecture, resourceid, publisher)

`<Name>_<Version>_<Architecture>_<ResourceId>_<PublisherId>`

For example, one package full name for the Windows Photos app is "Microsoft.Windows.Photos_2020.20090.1002.0_x64__8wekyb3d8bbwe", where "Microsoft.Windows.Photos" is the name, "2020.20090.1002.0" is the version number, "x64" is the target processor architecture, the resource ID is empty (no content between the last two underscores), and "8wekyb3d8bbwe" is the publisher ID for Microsoft.

The **Package Full Name** uniquely identifies an **MSIX** package or bundle. It is an error to have two packages or bundles with different contents but with the same Package Full Name.

> ⓘ **Note**
>
> **MSIX** is the new name for the previous term **APPX**. For more information, see **What is MSIX?**

Package Family Name

A **Package Family Name** is an opaque string derived from only two parts of a package identity - *name* and *publisher*:

`<Name>_<PublisherId>`

For example, the Package Family Name of the Windows Photos app is "Microsoft.Windows.Photos_8wekyb3d8bbwe", where "Microsoft.Windows.Photos" is the name and "8wekyb3d8bbwe" is the publisher ID for Microsoft.

Package Family Name is often referred to as a 'version-less Package Full Name'.

> ⓘ **Note**
>
> This isn't strictly true, as Package Family Name also lacks architecture and Resource ID.

> ⓘ **Note**
>
> Data and security are typically scoped to a package family. For example, it would be a poor experience if you configured the Notepad app installed from a Notepad version 1.0.0.0 package to enable Wordwrap. Then Notepad was subsequently updated to 1.0.0.1 and your configuration data wasn't carried over to the newer version of the package.

Publisher Id

A Package Family Name is a string with the format:

`<name>_<publisherid>`

where Publisher Id has some very specific properties:

- Derived from Publisher
- MinLength = MaxLength = 13 chars [fixed-size]
- Allowed Characters (as regex) = a-hj-km-np-tv-z0-9
  - Base-32, Crockford Variant, i.e. alphanumeric (A-Z0-9) except no I (eye), L (ell), O (oh) or U (you)
- Ordinal case-insensitive for comparisons --- ABCDEFABCDEFG == abcdefabcdefg

So you'll never see % : \ / " ? or other characters in a Publisher Id.

See PackageFamilyNameFromId and PackageNameAndPublisherIdFromFamilyName for more details.

Publisher Id is often referred to as PublisherId.

Why does Publisher Id exist?

Publisher Id exists because Publisher needs to match your cert's X.509 name/signer, thus:

- It can be very big (length <= 8192 chars)
- It can include characters that are awkward or restricted (backslash, etc.)

These issues can make some X.509 strings awkward or impossible to use in the filesystem, registry, URLs, and other contexts.

How do I create a PublisherId?

Use PackageNameAndPublisherIdFromFamilyName to extract the `PublisherId` from a `PackageFamilyName`.

Use PackageIdFromFullName to extract the `PublisherId` from a `PackageFullName`.

It's rare to need to create a `PublisherId` from `Publisher`, but it can be done with the use of available APIs:

C++

```
#include <appmodel.h>

HRESULT PublisherIdFromPublisher(
    _In_ PCWSTR publisher,
    _Out_writes_(PACKAGE_PUBLISHERID_MAX_LENGTH + 1) PWSTR publisherId)
{
    PCWSTR name{ L"xyz" };
    const size_t nameLength{ ARRAYSIZE(L"xyz") - 1 };
    const size_t offsetToPublisherId{ name + 1 }; // xyz_...publisherid...
    PACKAGE_ID id{};
    id.name = name;
    id.publisher = publisher;

    WCHAR familyName[PACKAGE_PUBLISHERID_MAX_LENGTH + 1]{};
    UINT32 n{ ARRAYSIZE(familyName) };
    RETURN_IF_WIN32_ERROR(PackageFamilyNameFromId(&id, &n, familyName);
    RETURN_IF_FAILED(StringCchCopyW(publisherId,
PACKAGE_PUBLISHERID_MAX_LENGTH + 1, familyName + offsetToPublisherId));
    return S_OK;
}
```

The following is a classic Windows C implementation of the same operation:

C

```
#include <appmodel.h>

HRESULT PublisherIdFromPublisher(
    _In_ PCWSTR publisher,
    _Out_writes_(PACKAGE_PUBLISHERID_MAX_LENGTH + 1) PWSTR publisherId)
{
    const WCHAR c_name[]{ L"xyz" };
    const UINT32 c_nameLength{ ARRAYSIZE(c_nameForPublisherToPublisherId) -
1 };

    PACKAGE_ID id{};
    id.name = c_name;
    id.publisher = publisher;
    WCHAR familyName[PACKAGE_PUBLISHERID_MAX_LENGTH + 1]{};
    UINT32 n{ ARRAYSIZE(familyName) };
    RETURN_IF_WIN32_ERROR(PackageFamilyNameFromId(&id, &n, familyName));
    RETURN_IF_FAILED(StringCchCopyW(publisherId,
PACKAGE_PUBLISHERID_MAX_LENGTH + 1,  familyName + c_nameLength + 1);
    return S_OK;
}
```

This creates the PublisherId by converting a Package Id to a Package Family Name with the resulting format `xyz_<publisherid>`. This recipe is stable and reliable.

This only requires you compile with appmodel.h from the SDK and link with kernel32.lib (or kernelbase.lib, onecore.lib, or api-ms-win-appmodel-runtime-l1.lib if using APIsets).

Understanding processor architecture in package identity

A common misunderstanding is that `Architecture=x64` means the package can only contain x64 code. This is not true. It means that the package works on systems supporting x64 code, and can be used by x64 apps. You could make a package containing only PDF files but declare it with `<Identity Architecture=x64...>` because it is only intended to be installed on x64-compatible systems (e.g. x64 packages can only be installed on x64 and (as of Windows 11) Arm64 systems, because x86, Arm, and Windows 10 Arm64 systems don't support x64).

Even more misunderstood, `Architecture=neutral` does *not* mean the package contains no executable code. It means the package works on all architectures. For example, you could make a package containing an AES encryption API written in JavaScript, Python,

C#, etc. but the performance is not acceptable on Arm64 systems. So, you include an optimized Arm64 binary and implement the API to handle it:

C++

```
void Encrypt(...)
{
    HANDLE h{};
    if (GetCpu() == arm64)
    {
        h = LoadLibrary(GetCurrentPackagePath() + "\bin\encrypt-arm64.dll")
        p = GetProcAddress(h, "Encrypt")
        return (*p)(...)
    }
    else
    {
        // ...call other implementation...
    }
}
```

Or you can make a neutral package with multiple variants:

YAML

```
\
    bin\
        encrypt-x86.dll
        encrypt-x64.dll
        encrypt-arm.dll
        encrypt-arm64.dll
```

Developers can then `LoadLibrary("bin\encrypt-" + cpu + ".dll")` to get the appropriate binary for their process at runtime.

Typically, neutral packages have no per-architecture content, but they can. There are limits to what one can do (e.g. you could make a Notepad package containing notepad.exe compiled for x86 + x64 + arm + arm64 but **appxmanifest.xml** can only declare `<Application Executable=...>` pointing to one of them). Given that there are bundles that let you install only the needed bits, this is a very uncommon thing to do. It's not illegal, just advanced and exotic.

Also, `Architecture=x86` (or x64|arm|arm64) doesn't mean the package only contains executable code for the specified architecture. It's just the overwhelmingly common case.

> ⓘ Note

When discussing "code" or "executable code" in this context, we are referring to portable executable (PE) files.

Is package identity case sensitive?

Mostly, no, but `Publisher` is case-sensitive.

The remaining fields (`Name`, `ResourceId`, `PublisherId`, `PackageFullName` and `PackageFamilyName`) are not. These fields will preserve case but compare case-insensitively.

See also

Package identity

PackageFamilyNameFromId

PackageNameAndPublisherIdFromFamilyName

Windows apps: packaging, deployment, and process

Article • 08/19/2024

> **Note**
>
> **Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.**

This topic discusses your options concerning:

- Whether or not your app will be packaged.
- How you'll deploy/distribute your app, and how it'll be installed.
- Your app's run-time process—how isolated it'll be; and what APIs will be available to it.

You can make those decisions for both new and existing apps. But if you're still in the planning stage for a new app, then before you start to think about the considerations above, first decide what development platform and user interface (UI) framework you'll use for your app. And for that decision, see An overview of Windows development options.

Packaged or unpackaged

The decision to make your app either packaged or unpackaged is first determined by a concept known as *package identity*, which we'll describe in this section. If you don't need that, then the decision comes down to the desired installer experience for yourself and for your users. Let's drill into the details of those things.

Many Windows extensibility features—including background tasks, notifications, live tiles, custom context menu extensions, and share targets—can be used by an app only if that app has package identity at runtime. That's because the operating system (OS) needs to be able to identify the caller of the corresponding API. See Features that require package identity.

- If you need to use any of those features, then your app needs package identity. And therefore it *needs* to be a packaged app (packaged apps are the only kind that

have package identity). A packaged app is packaged by using MSIX technology (see What is MSIX?).
    - For a new app, the packaging process is straightfoward (and at the end of this section there's info about how to do it).
    - For some existing apps, you can follow the same packaging process as for a new app. But because some existing apps aren't yet ready for all of their content to be present inside an MSIX package, there's an option for your app to be *packaged with external location*. That enables your app to have package identity; thereby being able to use those features that require it. For more info, see Grant package identity by packaging with external location.
- Even if you *don't* need to use any of those features, then creating a packaged app is still a good idea. It gives your users the easiest way to install, uninstall, and update your app. For more info, see Deployment/distribution/installation in this topic.
- But creating an unpackaged app is an option.

The takeaway is that packaged apps are the only kind that have package identity (and they have the best install experience). An unpackaged app doesn't have package identity; so it can't use the APIs/features mentioned above.

For more details about packaged versus unpackaged, see Deployment overview; in particular, the **Advantages and disadvantages of packaging your app** section in that topic. That topic also mentions the *packaged with external location* option.

For info about how to configure your app as packaged or unpackaged:

- **WinUI 3 apps (Windows App SDK)**. See the `AppxPackage` Visual Studio project property in Project properties; and see Create your first WinUI 3 (Windows App SDK) project.
- **Desktop apps**. See Set up your desktop app for MSIX packaging.
- **Universal Windows Platform (UWP) apps**. UWP apps are already configured as packaged; and that configuration can't be changed.

Also see the Windows Package Manager and the WinGet client section in this topic.

Deployment/distribution/installation

- A packaged app is packaged by using MSIX technology.
    - A packaged app is *installed* by using MSIX, also. But if you choose to *package with external location*, then you can think of that as a "bring-your-own-installer" model. So there *will* be some installer work for you to do with that option. For more info, see Grant package identity by packaging with external location.

- An unpackaged app doesn't involve MSIX at all.

So, why does it matter whether or not your app is packaged?

- Well, MSIX gives your users an easy way to install, uninstall, and update your app. Uninstall is clean—when your app is uninstalled, the system is restored to the same state it was in before installation; no artifacts are left behind.
- This kind of app also supports incremental and automatic updates.
- The Microsoft Store optimizes for apps of this kind (although they can be used in or out of the Store).
- It's an easy path for use via MSIX app attach (for Azure Virtual Desktop virtual machines). For more info, see What is MSIX app attach?.
- A signed package benefits from strong anti-tampering. This benefit is even greater than for an unpackaged app installed under **Program Files**.

Also see the Windows Package Manager and the WinGet client section in this topic.

AppContainer or Medium IL

The option to run your app in an AppContainer, or not, is a question of security. An AppContainer app's process and its child processes run inside a lightweight app container where they can access only the resources that are specifically granted to them. And they're isolated using file system and registry virtualization. As a result, apps implemented in an AppContainer can't be hacked to allow malicious actions outside of the limited assigned resources.

Packaged or unpackaged apps can be configured to run in an AppContainer. But the process is more straightforward for packaged apps. If an app isn't an AppContainer app, then it's a *Medium IL* app.

For more info, see AppContainer for legacy apps and MSIX AppContainer apps.

For info about how to configure your app to run in an AppContainer or Medium IL:

- **WinUI 3 apps (Windows App SDK)**. See the `uap10:TrustLevel` app package manifest attribute in Configure a WinUI 3 project for AppContainer.
- **Desktop apps**. See the `TrustLevel` Visual Studio project property in MSIX AppContainer apps (in the section that's appropriate for your kind of app).
- **Universal Windows Platform (UWP) apps**. UWP apps are already configured to run in an AppContainer; and that configuration can't be changed.

Remember that unpackaged apps don't have an app package manifest. So for unpackaged apps, you declare your AppContainer-or-Medium-IL decision in your

project file instead of in an app package manifest.

Win32 app isolation

> **Important**
>
> The feature described in this section is available in pre-release versions of the Windows Insider Preview.

Win32 app isolation is an upcoming security feature in Windows which, in the event of an app being compromised, helps contain the damage, and safeguard user privacy choices. This feature is built on the foundation of AppContainers and components that virtualize resources and provide brokered access to other resources. For documentation and tools to help you isolate your apps, see Welcome to the Win32 app isolation repo.

App capabilities

App capabilities (for example, internetClient, location, microphone, and bluetooth) are relevant mostly to *packaged apps that run in an AppContainer*. So that includes *all* Universal Windows Platform (UWP) apps, and *some* desktop apps.

But there are some scenarios where even a Medium IL app (that is, *not* an AppContainer app) should declare a capability. One example is the *runFullTrust* restricted capability.

For more details about app capabilities, what kinds of apps they apply to, and how to configure them, see App capability declarations. You configure capabilities in your app package manifest; and that's why they apply only to packaged apps.

Kinds of apps

Desktop apps and Universal Windows Platform (UWP) apps are the two main kinds of apps—although, there are several kinds of apps in the *desktop apps* family. Choosing a user interface (UI) framework—WinForms, WPF, Win32, Direct 2D/3D, UWP, or WinUI 3—is one option that's to some degree independent of the configurations described in this topic.

But let's take a look at how those app kinds can differ from one another in terms of packaging, deployment, and process.

First off, all UWP apps are packaged, and run in an AppContainer. But for desktop apps, things are more flexible. You can choose to package your desktop app, or not. And, independently of that decision, you can choose to configure your desktop app as either an AppContainer or a Medium IL app.

Expand table

	Packaged	Unpackaged
AppContainer	Desktop apps UWP apps	Desktop apps
Medium IL	Desktop apps	Desktop apps

For packaged apps, to configure the kind of app you want, you use the `uap10:RuntimeBehavior` attribute in your app package manifest (see Application (Windows 10)).

- **Desktop apps** are Windows `.exe`s, typically with a **main** or **WinMain** entry-point function. To configure your app as a desktop app, set `uap10:RuntimeBehavior` to either "packagedClassicApp" or "win32App".
  - The value "packagedClassicApp" indicates either a WinUI 3 app (Windows App SDK) or a Desktop Bridge app (Centennial). The difference is that a Centennial app runs in an AppContainer.
  - And "win32App" indicates any other kind of Win32 app (including an app packaged with external location).
- Lastly, setting `uap10:RuntimeBehavior` to "windowsApp" gives you a UWP app.

For all of the options for the kinds of apps you can develop, see Windows app development: options and features.

Windows App SDK—framework-dependent or self-contained

If you're developing or maintaining an app that makes use of the Windows App SDK, then you have a further decision to make. Because there are the following two ways in which you can deploy the Windows App SDK that your app depends on:

- Framework-dependent (the default). Your app needs the Windows App SDK runtime and/or Framework package to be present on the target machine.
- Self-contained. Your app carries with it its Windows App SDK dependencies.

For more info, see Windows App SDK deployment overview.

Windows Package Manager and the WinGet client

A package manager can help your users to install/upgrade/configure your software by automating the workflow. Package managers can help install any software, but they tend to be used mostly to install developer tools. So if you're building a developer tool, then you might be particuarly interested in this option. But here's how it works:

- You, as the software developer, define to the package manager (in the form of declarative instructions) all of the pieces necessary for a successful install of your product.
- And then when a user installs your software, the package manager follows your declarative instructions to automate the install-and-configure workflow.

The result is a reduction in time spent getting a user's environment ready, and better compatibility between the components installed. And you can use Windows Package Manager to distribute your packaged or unpackaged apps in formats such as `.msix`, `.msi`, and `.exe`.

For more info, see Windows Package Manager.

Features that require package identity

Article • 05/26/2023

Some modern Windows experiences require your app to have package identity at runtime (in other words, your app needs to be *packaged*). Those experiences include certain Windows features, certain Windows Runtime APIs, package extensions, and UWP components.

Universal Windows Platform (UWP) apps receive package identity by default because they can be distributed only via MSIX packages. Other types of Windows apps, including WPF apps, can also be deployed via MSIX packages to obtain package identity. But apps *packaged with external location* also has package identity. For more info about these terms, see Advantages and disadvantages of packaging your app.

Only packaged apps (including apps packaged with external location) have package identity at runtime. If your app has package identity, then you can use the following features in your app.

Notifications

The Windows App SDK notifications APIs require your app to have package identity.

Integrate with package extensions

If your app needs to integrate with the system (for example, establish firewall rules), then describe those things in the package manifest of your app, and the system will do the rest. For most of these tasks, you won't have to write any code at all. With a bit of XML in the manifest, you can do things such as: start a process when the user logs on; integrate your app into File Explorer; and add your app a list of print targets that appear in other apps.

For more info, see Integrate your desktop app with package extensions.

Get activation info for packaged apps

Starting in Windows 10, version 1809, packaged apps can retrieve certain kinds of activation info during startup. For example, you can get info related to app activation from opening a file, from clicking an interactive toast, or from using a protocol.

For more info, see Get activation info for packaged apps.

Extend with UWP components

Some Windows experiences (for example, a touch-enabled UI page) must run inside of an AppContainer. In general, you should first determine whether you can add your experience by enhancing your existing desktop app with Windows Runtime APIs. If you have to use a UWP component to achieve the experience, then you can add a UWP project to your solution, and use app services to communicate between your desktop app and the UWP component.

For more information, see Extend your desktop app with UWP components.

Distribute

If you package your app in an MSIX package, then it's very easy to distribute it by publishing it the Microsoft Store, or by sideloading it onto systems.

For more info, see Distribute your packaged desktop app.

Integrate your desktop app with Windows by using packaging extensions

Article • 05/07/2024

If your desktop app is packaged (has package identity at runtime), then you can use extensions to integrate your app with Windows by using predefined extensions in the package manifest. Also see Features that require package identity.

For example, use an extension to create a firewall exception; make your app the default application for a file type; or point Start tiles to your app. To use an extension, just add some XML to your app's package manifest file. No code is required.

This topic describes those extensions and the tasks that you can perform by using them.

> **Note**
>
> The features described in this topic require that your app is packaged (has package identity at runtime). That includes packaged apps (see Create a new project for a packaged WinUI 3 desktop app) and packaged apps with external location (see Grant package identity by packaging with external location). Also see Features that require package identity.

Transition users to your app

Help users transition to your packaged app.

- Redirect your existing desktop app to your packaged app
- Point existing Start tiles and taskbar buttons to your packaged app
- Make your packaged app open files instead of your desktop app
- Associate your packaged app with a set of file types
- Add options to the context menus of files that have a certain file type
- Open certain types of files directly by using a URL

Redirect your existing desktop app to your packaged app

When users start your existing unpackaged desktop app, you can configure your packaged app to be opened instead.

> **Note**

This feature is supported in Windows Insider Preview Build 21313 and later versions.

To enable this behavior:

1. Add registry entries to redirect your unpackaged desktop app executable to your packaged app.
2. Register your packaged app to be launched when your unpackaged desktop app executable is launched.

Add registry entries to redirect your unpackaged desktop app executable

1. In the registry, create a subkey with the name of your desktop app executable file under the **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options** key.
2. Under this subkey, add the following values:

    - **AppExecutionAliasRedirect** (DWORD): If set to 1, the system will check for an AppExecutionAlias package extension with the same name as the executable. If the **AppExecutionAlias** extension is enabled, the packaged app will be activated using that value.
    - **AppExecutionAliasRedirectPackages** (REG_SZ): The system will redirect only to the listed packages. Packages are listed by their package family name, separated by semicolons. If the special value * is used, the system will redirect to an **AppExecutionAlias** from any package.

For example:

```ini
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\contosoapp.exe
    AppExecutionAliasRedirect = 1
    AppExecutionAliasRedirectPackages = "Microsoft.WindowsNotepad_8weky8webbe"
```

Register your packaged app to be launched

In your package manifest, add an AppExecutionAlias extension that registers the name of your unpackaged desktop app executable. For example:

XML

```
<Package
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
  IgnorableNamespaces="uap3">
  <Applications>
    <Application>
      <Extensions>
        <uap3:Extension Category="windows.appExecutionAlias" EntryPoint="Win
dows.FullTrustApplication">
          <uap3:AppExecutionAlias>
            <desktop:ExecutionAlias Alias="contosoapp.exe" />
          </uap3:AppExecutionAlias>
        </uap3:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Disable the redirection

Users can turn off the redirection and launch your unpackaged app executable via these options:

- They can uninstall the packaged version of your app.
- The user can disable the **AppExecutionAlias** entry for your packaged app in the **App execution aliases** page in **Settings**.

XML namespaces

- `http://schemas.microsoft.com/appx/manifest/uap/windows10/3`
- `http://schemas.microsoft.com/appx/manifest/desktop/windows10`

Elements and attributes of this extension

XML

```
<uap3:Extension
    Category="windows.appExecutionAlias"
    EntryPoint="Windows.FullTrustApplication">
    <uap3:AppExecutionAlias>
        <desktop:ExecutionAlias Alias="[AliasName]" />
    </uap3:AppExecutionAlias>
</uap3:Extension>
```

Expand table

Name	Description
Category	Always `windows.appExecutionAlias`.
Executable	The relative path to the executable to start when the alias is invoked.
Alias	The short name for your app. It must always end with the ".exe" extension.

Point existing Start tiles and taskbar buttons to your packaged app

Your users might have pinned your desktop application to the taskbar or the Start menu. You can point those shortcuts to your new packaged app.

XML namespace

`http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities/3`

Elements and attributes of this extension

XML

```xml
<Extension Category="windows.desktopAppMigration">
    <DesktopAppMigration>
        <DesktopApp AumId="[your_app_aumid]" />
        <DesktopApp ShortcutPath="[path]" />
    </DesktopAppMigration>
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.desktopAppMigration`.
AumID	The Application User Model ID of your packaged app.
ShortcutPath	The path to .lnk files that start the desktop version of your app.

Example

XML

```
<Package

xmlns:rescap3="http://schemas.microsoft.com/appx/manifest/foundation/windows
10/restrictedcapabilities/3"
  IgnorableNamespaces="rescap3">
  <Applications>
    <Application>
      <Extensions>
        <rescap3:Extension Category="windows.desktopAppMigration">
          <rescap3:DesktopAppMigration>
            <rescap3:DesktopApp AumId="[your_app_aumid]" />
            <rescap3:DesktopApp ShortcutPath="%USERPROFILE%\Desktop\
[my_app].lnk" />
            <rescap3:DesktopApp
ShortcutPath="%APPDATA%\Microsoft\Windows\Start Menu\Programs\[my_app].lnk"
/>
            <rescap3:DesktopApp
ShortcutPath="%PROGRAMDATA%\Microsoft\Windows\Start Menu\Programs\
[my_app_folder]\[my_app].lnk"/>
          </rescap3:DesktopAppMigration>
        </rescap3:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Related sample

WPF picture viewer with transition/migration/uninstallation

Make your packaged application open files instead of your desktop app

You can make sure that users open your new packaged application by default for specific types of files instead of opening the desktop version of your app.

To do that, you'll specify the programmatic identifier (ProgID) of each application from which you want to inherit file associations.

XML namespaces

- `http://schemas.microsoft.com/appx/manifest/uap/windows10/3`

- `http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities/3`

Elements and attributes of this extension

XML

```
<Extension Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="[Name]">
         <MigrationProgIds>
            <MigrationProgId>"[ProgID]"</MigrationProgId>
        </MigrationProgIds>
    </FileTypeAssociation>
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.fileTypeAssociation`.
Name	The name of the file type association. You can use this name to organize and group file types. The name must be all lower case characters with no spaces.
MigrationProgId	The programmatic identifier (ProgID) that describes the application, component, and version of the desktop application from which you want to inherit file associations.

Example

XML

```
<Package
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"

xmlns:rescap3="http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities/3"
  IgnorableNamespaces="uap3, rescap3">
  <Applications>
    <Application>
      <Extensions>
        <uap:Extension Category="windows.fileTypeAssociation">
          <uap3:FileTypeAssociation Name="myfiletypes">
            <rescap3:MigrationProgIds>
              <rescap3:MigrationProgId>Foo.Bar.1</rescap3:MigrationProgId>
              <rescap3:MigrationProgId>Foo.Bar.2</rescap3:MigrationProgId>
```

```
            </rescap3:MigrationProgIds>
          </uap3:FileTypeAssociation>
        </uap:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Related sample

WPF picture viewer with transition/migration/uninstallation

Associate your packaged application with a set of file types

You can associate your packaged application with file type extensions. If a user right-clicks a file in File Explorer and then selects the **Open with** option, your application appears in the list of suggestions. For more information about using this extension, see Integrate a packaged desktop app with File Explorer.

XML namespaces

- `http://schemas.microsoft.com/appx/manifest/uap/windows10`
- `http://schemas.microsoft.com/appx/manifest/uap/windows10/3`

Elements and attributes of this extension

XML

```
<Extension Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="[Name]">
        <SupportedFileTypes>
            <FileType>"[file extension]"</FileType>
        </SupportedFileTypes>
    </FileTypeAssociation>
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.fileTypeAssociation`.
Name	The name of the file type association. You can use this name to organize and group file types. The name must be all lower case characters with no spaces.
FileType	The file extension supported by your app.

Example

XML

```
<Package
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
  IgnorableNamespaces="uap, uap3">
  <Applications>
    <Application>
      <Extensions>
        <uap:Extension Category="windows.fileTypeAssociation">
          <uap3:FileTypeAssociation Name="mediafiles">
            <uap:SupportedFileTypes>
            <uap:FileType>.avi</uap:FileType>
            </uap:SupportedFileTypes>
          </uap3:FileTypeAssociation>
        </uap:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Related sample

WPF picture viewer with transition/migration/uninstallation

Add options to the context menus of files that have a certain file type

This extension enables you to add options to the context menu that displays when users right-click a file in File Explorer.These options give users other ways to interact with your file such as print, edit, or preview the file. For more information about using this extension, see Integrate a packaged desktop app with File Explorer.

XML namespaces

- `http://schemas.microsoft.com/appx/manifest/uap/windows10`
- `http://schemas.microsoft.com/appx/manifest/uap/windows10/2`
- `http://schemas.microsoft.com/appx/manifest/uap/windows10/3`

Elements and attributes of this extension

XML

```
<Extension Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="[Name]">
        <SupportedVerbs>
           <Verb Id="[ID]" Extended="[Extended]" Parameters="[parameters]">"
[verb label]"</Verb>
        </SupportedVerbs>
    </FileTypeAssociation>
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.fileTypeAssociation`.
Name	The name of the file type association. You can use this name to organize and group file types. The name must be all lower case characters with no spaces.
Verb	The name that appears in the File Explorer context menu. This string is localizable that uses `ms-resource`.
Id	The unique Id of the verb. If your application is a UWP app, this is passed to your app as part of its activation event args so it can handle the user's selection appropriately. If your application is a full-trust packaged app, it receives parameters instead (see the next bullet).
Parameters	The list of argument parameters and values associated with the verb. If your application is a full-trust packaged app, these parameters are passed to the application as event args when the application is activated. You can customize the behavior of your application based on different activation verbs. If a variable can contain a file path, wrap the parameter value in quotes. That will avoid any issues that happen in cases where the path includes spaces. If your application is a UWP app, you can't pass parameters. The app receives the Id instead (see the previous bullet).
Extended	Specifies that the verb appears only if the user shows the context menu by holding the **Shift** key before right-clicking the file. This attribute is optional and defaults to a

Name	Description
	value of **False** (for example, always show the verb) if not listed. You specify this behavior individually for each verb (except for "Open," which is always **False**).

Example

XML

```
<Package
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
  xmlns:uap2="http://schemas.microsoft.com/appx/manifest/uap/windows10/2"
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"

  IgnorableNamespaces="uap, uap2, uap3">
  <Applications>
    <Application>
      <Extensions>
        <uap:Extension Category="windows.fileTypeAssociation">
          <uap3:FileTypeAssociation Name="myfiletypes">
            <uap2:SupportedVerbs>
              <uap3:Verb Id="Edit" Parameters="/e
&amp;quot;%1&amp;quot;">Edit</uap3:Verb>
              <uap3:Verb Id="Print" Extended="true" Parameters="/p
&amp;quot;%1&amp;quot;">Print</uap3:Verb>
            </uap2:SupportedVerbs>
          </uap3:FileTypeAssociation>
        </uap:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Related sample

WPF picture viewer with transition/migration/uninstallation

Open certain types of files directly by using a URL

You can make sure that users open your new packaged application by default for specific types of files instead of opening the desktop version of your app.

XML namespaces

- `http://schemas.microsoft.com/appx/manifest/uap/windows10`
- `http://schemas.microsoft.com/appx/manifest/uap/windows10/3`

Elements and attributes of this extension

XML

```
<Extension Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="[Name]" UseUrl="true" Parameters="%1">
        <SupportedFileTypes>
            <FileType>"[FileExtension]"</FileType>
        </SupportedFileTypes>
    </FileTypeAssociation>
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.fileTypeAssociation`.
Name	The name of the file type association. You can use this name to organize and group file types. The name must be all lower case characters with no spaces.
UseUrl	Indicates whether to open files directly from a URL target. If you do not set this value, attempts by your application to open a file by using a URL cause the system to first download the file locally.
Parameters	Optional parameters.
FileType	The relevant file extensions.

Example

XML

```
<Package
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
  IgnorableNamespaces="uap, uap3">
  <Applications>
      <Application>
        <Extensions>
          <uap:Extension Category="windows.fileTypeAssociation">
            <uap3:FileTypeAssociation Name="myfiletypes" UseUrl="true"
Parameters="%1">
              <uap:SupportedFileTypes>
                <uap:FileType>.txt</uap:FileType>
                <uap:FileType>.doc</uap:FileType>
              </uap:SupportedFileTypes>
```

```
            </uap3:FileTypeAssociation>
          </uap:Extension>
        </Extensions>
      </Application>
    </Applications>
</Package>
```

Perform setup tasks

- Create firewall exception for your app
- Place your DLL files into any folder of the package

Create firewall exception for your app

If your application requires communication through a port, you can add your application to the list of firewall exceptions.

> ⓘ Note
>
> To use the "windows.firewallRules" extension category (see below), your package needs the **Full Trust Permission Level** restricted capability. See Restricted capability list.

XML namespace

`http://schemas.microsoft.com/appx/manifest/desktop/windows10/2`

Elements and attributes of this extension

XML

```
<Extension Category="windows.firewallRules">
  <FirewallRules Executable="[executable file name]">
    <Rule
      Direction="[Direction]"
      IPProtocol="[Protocol]"
      LocalPortMin="[LocalPortMin]"
      LocalPortMax="LocalPortMax"
      RemotePortMin="RemotePortMin"
      RemotePortMax="RemotePortMax"
      Profile="[Profile]"/>
  </FirewallRules>
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.firewallRules`
Executable	The name of the executable file that you want to add to the list of firewall exceptions
Direction	Indicates whether the rule is an inbound or outbound rule
IPProtocol	The communication protocol
LocalPortMin	The lower port number in a range of local port numbers.
LocalPortMax	The highest port number of a range of local port numbers.
RemotePortMax	The lower port number in a range of remote port numbers.
RemotePortMax	The highest port number of a range of remote port numbers.
Profile	The network type

Example

XML

```
<Package

xmlns:desktop2="http://schemas.microsoft.com/appx/manifest/desktop/windows10
/2"
  IgnorableNamespaces="desktop2">
  <Extensions>
    <desktop2:Extension Category="windows.firewallRules">
      <desktop2:FirewallRules Executable="Contoso.exe">
          <desktop2:Rule Direction="in" IPProtocol="TCP" Profile="all"/>
          <desktop2:Rule Direction="in" IPProtocol="UDP" LocalPortMin="1337"
LocalPortMax="1338" Profile="domain"/>
          <desktop2:Rule Direction="in" IPProtocol="UDP" LocalPortMin="1337"
LocalPortMax="1338" Profile="public"/>
          <desktop2:Rule Direction="out" IPProtocol="UDP"
LocalPortMin="1339" LocalPortMax="1340" RemotePortMin="15"
                         RemotePortMax="19" Profile="domainAndPrivate"/>
          <desktop2:Rule Direction="out" IPProtocol="GRE"
Profile="private"/>
      </desktop2:FirewallRules>
  </desktop2:Extension>
```

```
</Extensions>
</Package>
```

Place your DLL files into any folder of the package

Use the uap6:LoaderSearchPathOverride extension to declare up to five folder paths in the app package, relative to the app package root path, to be used in the loader search path for the app's processes.

The DLL search order for Windows apps includes packages in the package dependency graph if the packages have execution rights. By default, this includes main, optional and framework packages, although this can be overwritten by the uap6:AllowExecution element in the package manifest.

A package that is included in the DLL search order will, by default, include its *effective path*. For more information about effective paths, see the EffectivePath property (WinRT) and the PackagePathType enumeration (Win32).

If a package specifies uap6:LoaderSearchPathOverride, then this information is used instead of the package's effective path.

Each package can contain only one uap6:LoaderSearchPathOverride extension. That means that you can add one of them to your main package, and then add one to each of your optional packages, and related sets.

XML namespace

`http://schemas.microsoft.com/appx/manifest/uap/windows10/6`

Elements and attributes of this extension

Declare this extension at the package-level of your app manifest.

XML

```
<Extension Category="windows.loaderSearchPathOverride">
  <LoaderSearchPathOverride>
    <LoaderSearchPathEntry FolderPath="[path]"/>
  </LoaderSearchPathOverride>
</Extension>
```

Expand table

Name	Description
Category	Always `windows.loaderSearchPathOverride`.
FolderPath	The path of the folder that contains your DLL files. Specify a path that is relative to the root folder of the package. You can specify up to five paths in one extension. If you want the system to search for files in the root folder of the package, use an empty string for one of these paths. Don't include duplicate paths, and make sure that your paths don't contain leading and trailing slashes or backslashes. The system won't search subfolders, so make sure to explicitly list each folder that contains DLL files that you want the system to load.

Example

XML

```
<Package
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/6"
  IgnorableNamespaces="uap6">
  ...
    <Extensions>
      <uap6:Extension Category="windows.loaderSearchPathOverride">
        <uap6:LoaderSearchPathOverride>
          <uap6:LoaderSearchPathEntry FolderPath=""/>
          <uap6:LoaderSearchPathEntry FolderPath="folder1/subfolder1"/>
          <uap6:LoaderSearchPathEntry FolderPath="folder2/subfolder2"/>
        </uap6:LoaderSearchPathOverride>
      </uap6:Extension>
    </Extensions>
...
</Package>
```

Integrate with File Explorer

Help users organize your files and interact with them in familiar ways.

- Define how your application behaves when users select and open multiple files at the same time
- Show file contents in a thumbnail image within File Explorer
- Show file contents in a Preview pane of File Explorer
- Enable users to group files by using the Kind column in File Explorer
- Make file properties available to search, index, property dialogs, and the details pane

- Specify a context menu handler for a file type
- Make files from your cloud service appear in File Explorer

Define how your application behaves when users select and open multiple files at the same time

Specify how your application behaves when a user opens multiple files simultaneously.

XML namespaces

- `http://schemas.microsoft.com/appx/manifest/uap/windows10`
- `http://schemas.microsoft.com/appx/manifest/uap/windows10/2`
- `http://schemas.microsoft.com/appx/manifest/uap/windows10/3`

Elements and attributes of this extension

XML

```
<Extension Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="[Name]" MultiSelectModel="[SelectionModel]">
        <SupportedVerbs>
            <Verb Id="Edit" MultiSelectModel="[SelectionModel]">Edit</Verb>
        </SupportedVerbs>
        <SupportedFileTypes>
            <FileType>"[FileExtension]"</FileType>
        </SupportedFileTypes>
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.fileTypeAssociation`.
Name	The name of the file type association. You can use this name to organize and group file types. The name must be all lower case characters with no spaces.
MultiSelectModel	See below
FileType	The relevant file extensions.

MultiSelectModel

packaged desktop apps have the same three options as regular desktop apps.

- `Player`: Your application is activated one time. All of the selected files are passed to your application as argument parameters.
- `Single`: Your application is activated one time for the first selected file. Other files are ignored.
- `Document`: A new, separate instance of your application is activated for each selected file.

You can set different preferences for different file types and actions. For example, you may wish to open *Documents* in *Document* mode and *Images* in *Player* mode.

Example

XML

```
<Package
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
  xmlns:uap2="http://schemas.microsoft.com/appx/manifest/uap/windows10/2"
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
  IgnorableNamespaces="uap, uap2, uap3">
  <Applications>
    <Application>
      <Extensions>
        <uap:Extension Category="windows.fileTypeAssociation">
          <uap3:FileTypeAssociation Name="myfiletypes"
MultiSelectModel="Document">
            <uap2:SupportedVerbs>
              <uap3:Verb Id="Edit"
MultiSelectModel="Player">Edit</uap3:Verb>
              <uap3:Verb Id="Preview"
MultiSelectModel="Document">Preview</uap3:Verb>
            </uap2:SupportedVerbs>
            <uap:SupportedFileTypes>
              <uap:FileType>.txt</uap:FileType>
            </uap:SupportedFileTypes>
        </uap:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

If the user opens 15 or fewer files, the default choice for the **MultiSelectModel** attribute is *Player*. Otherwise, the default is *Document*. UWP apps are always started as *Player*.

Show file contents in a thumbnail image within File Explorer

Enable users to view a thumbnail image of the file's contents when the icon of the file appears in the medium, large, or extra large size.

XML namespace

- `http://schemas.microsoft.com/appx/manifest/uap/windows10`
- `http://schemas.microsoft.com/appx/manifest/uap/windows10/2`
- `http://schemas.microsoft.com/appx/manifest/uap/windows10/3`
- `http://schemas.microsoft.com/appx/manifest/desktop/windows10/2`

Elements and attributes of this extension

XML

```
<Extension Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="[Name]">
        <SupportedFileTypes>
            <FileType>"[FileExtension]"</FileType>
        </SupportedFileTypes>
        <ThumbnailHandler
            Clsid  ="[Clsid  ]" />
    </FileTypeAssociation>
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.fileTypeAssociation`.
Name	The name of the file type association. You can use this name to organize and group file types. The name must be all lower case characters with no spaces.
FileType	The relevant file extensions.
Clsid	The class ID of your app.

Example

XML

```
<Package
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
```

```
  xmlns:uap2="http://schemas.microsoft.com/appx/manifest/uap/windows10/2"
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"

xmlns:desktop2="http://schemas.microsoft.com/appx/manifest/desktop/windows10
/2"
  IgnorableNamespaces="uap, uap2, uap3, desktop2">
  <Applications>
    <Application>
      <Extensions>
        <uap:Extension Category="windows.fileTypeAssociation">
          <uap3:FileTypeAssociation Name="myfiletypes">
            <uap:SupportedFileTypes>
              <uap:FileType>.bar</uap:FileType>
            </uap:SupportedFileTypes>
            <desktop2:ThumbnailHandler
              Clsid  ="20000000-0000-0000-0000-000000000001"  />
            </uap3:FileTypeAssociation>
         </uap::Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Show file contents in the Preview pane of File Explorer

Enable users to preview a file's contents in the Preview pane of File Explorer.

XML namespace

- `http://schemas.microsoft.com/appx/manifest/uap/windows10`
- `http://schemas.microsoft.com/appx/manifest/uap/windows10/2`
- `http://schemas.microsoft.com/appx/manifest/uap/windows10/3`
- `http://schemas.microsoft.com/appx/manifest/desktop/windows10/2`

Elements and attributes of this extension

XML

```
<Extension Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="[Name]">
        <SupportedFileTypes>
            <FileType>"[FileExtension]"</FileType>
        </SupportedFileTypes>
        <DesktopPreviewHandler Clsid  ="[Clsid  ]" />
    </FileTypeAssociation>
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.fileTypeAssociation`.
Name	The name of the file type association. You can use this name to organize and group file types. The name must be all lower case characters with no spaces.
FileType	The relevant file extensions.
Clsid	The class ID of your app.

Example

XML

```
<Package
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
  xmlns:uap2="http://schemas.microsoft.com/appx/manifest/uap/windows10/2"
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"

xmlns:desktop2="http://schemas.microsoft.com/appx/manifest/desktop/windows10
/2"
  IgnorableNamespaces="uap, uap2, uap3, desktop2">
  <Applications>
    <Application>
      <Extensions>
        <uap:Extension Category="windows.fileTypeAssociation">
          <uap3:FileTypeAssociation Name="myfiletypes">
            <uap:SupportedFileTypes>
              <uap:FileType>.bar</uap:FileType>
                </uap:SupportedFileTypes>
              <desktop2:DesktopPreviewHandler Clsid ="20000000-0000-0000-
0000-000000000001" />
           </uap3:FileTypeAssociation>
        </uap:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Enable users to group files by using the Kind column in File Explorer

You can associate one or more predefined values for your file types with the **Kind** field.

In File Explorer, users can group those files by using that field. System components also use this field for various purposes such as indexing.

For more information about the **Kind** field and the values that you can use for this field, see Using Kind Names.

XML namespaces

- `http://schemas.microsoft.com/appx/manifest/uap/windows10`
- `http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities/3`

Elements and attributes of this extension

XML

```
<Extension Category="windows.fileTypeAssociation">
    <FileTypeAssociation Name="[Name]">
        <SupportedFileTypes>
            <FileType>"[FileExtension]"</FileType>
        </SupportedFileTypes>
        <KindMap>
            <Kind value="[KindValue]">
        </KindMap>
    </FileTypeAssociation>
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.fileTypeAssociation`.
Name	The name of the file type association. You can use this name to organize and group file types. The name must be all lower case characters with no spaces.
FileType	The relevant file extensions.
value	A valid Kind value

Example

XML

```
<Package
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"

xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities/3"
  IgnorableNamespaces="uap, rescap">
  <Applications>
    <Application>
      <Extensions>
        <uap:Extension Category="windows.fileTypeAssociation">
           <uap:FileTypeAssociation Name="mediafiles">
             <uap:SupportedFileTypes>
               <uap:FileType>.m4a</uap:FileType>
               <uap:FileType>.mta</uap:FileType>
             </uap:SupportedFileTypes>
             <rescap:KindMap>
               <rescap:Kind value="Item">
               <rescap:Kind value="Communications">
               <rescap:Kind value="Task">
             </rescap:KindMap>
          </uap:FileTypeAssociation>
      </uap:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Make file properties available to search, index, property dialogs, and the details pane

XML namespace

- `http://schemas.microsoft.com/appx/manifest/uap/windows10`
- `http://schemas.microsoft.com/appx/manifest/uap/windows10/3`
- `http://schemas.microsoft.com/appx/manifest/desktop/windows10/2`

Elements and attributes of this extension

XML

```
<uap:Extension Category="windows.fileTypeAssociation">
    <uap:FileTypeAssociation Name="[Name]">
        <SupportedFileTypes>
            <FileType>.bar</FileType>
        </SupportedFileTypes>
        <DesktopPropertyHandler Clsid ="[Clsid]"/>
```

```
    </uap:FileTypeAssociation>
</uap:Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.fileTypeAssociation`.
Name	The name of the file type association. You can use this name to organize and group file types. The name must be all lower case characters with no spaces.
FileType	The relevant file extensions.
Clsid	The class ID of your app.

Example

XML

```
<Package
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"

xmlns:desktop2="http://schemas.microsoft.com/appx/manifest/desktop/windows10
/2"
  IgnorableNamespaces="uap, uap3, desktop2">
  <Applications>
    <Application>
      <Extensions>
        <uap:Extension Category="windows.fileTypeAssociation">
          <uap3:FileTypeAssociation Name="myfiletypes">
            <uap:SupportedFileTypes>
              <uap:FileType>.bar</uap:FileType>
            </uap:SupportedFileTypes>
            <desktop2:DesktopPropertyHandler Clsid ="20000000-0000-0000-
0000-000000000001"/>
          </uap3:FileTypeAssociation>
        </uap:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Specify a context menu handler for a file type

If your desktop application defines a context menu handler, use this extension to register the menu handler.

XML namespaces

- `http://schemas.microsoft.com/appx/manifest/foundation/windows10`
- `http://schemas.microsoft.com/appx/manifest/desktop/windows10/4`

Elements and attributes of this extension

XML

```
<Extensions>
    <com:Extension Category="windows.comServer">
        <com:ComServer>
            <com:SurrogateServer AppId="[AppID]" DisplayName="
[DisplayName]">
                <com:Class Id="[Clsid]" Path="[Path]" ThreadingModel="
[Model]"/>
            </com:SurrogateServer>
        </com:ComServer>
    </com:Extension>
    <desktop4:Extension Category="windows.fileExplorerContextMenus">
        <desktop4:FileExplorerContextMenus>
            <desktop4:ItemType Type="[Type]">
                <desktop4:Verb Id="[ID]" Clsid="[Clsid]" />
            </desktop4:ItemType>
        </desktop4:FileExplorerContextMenus>
    </desktop4:Extension>
</Extensions>
```

Find the complete schema reference here: com:ComServer and desktop4:FileExplorerContextMenus.

Instructions

To register your context menu handler, follow these instructions.

1. In your desktop application, implement a context menu handler by implementing the IExplorerCommand or IExplorerCommandState interface. For a sample, see the ExplorerCommandVerb code sample. Make sure that you define a class GUID for each of your implementation objects. For example, the following code defines a class ID for an implementation of IExplorerCommand.

   C++

```
class __declspec(uuid("00001111-aaaa-2222-bbbb-3333cccc4444"))
CExplorerCommandVerb;
```

2. In your package manifest, specify a com:ComServer application extension that registers a COM surrogate server with the class ID of your context menu handler implementation.

```
XML

<com:Extension Category="windows.comServer">
    <com:ComServer>
        <com:SurrogateServer AppId="00001111-aaaa-2222-bbbb-
3333cccc4444" DisplayName="ContosoHandler">
            <com:Class Id="00001111-aaaa-2222-bbbb-3333cccc4444"
Path="ExplorerCommandVerb.dll" ThreadingModel="STA"/>
        </com:SurrogateServer>
    </com:ComServer>
</com:Extension>
```

3. In your package manifest, specify a desktop4:FileExplorerContextMenus application extension that registers your context menu handler implementation.

```
XML

<desktop4:Extension Category="windows.fileExplorerContextMenus">
    <desktop4:FileExplorerContextMenus>
        <desktop4:ItemType Type=".rar">
            <desktop4:Verb Id="Command1" Clsid="00001111-aaaa-2222-
bbbb-3333cccc4444" />
        </desktop4:ItemType>
    </desktop4:FileExplorerContextMenus>
</desktop4:Extension>
```

Example

```
XML

<Package
  xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"

xmlns:desktop4="http://schemas.microsoft.com/appx/manifest/desktop/windows10
/4"
  xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"
  IgnorableNamespaces="desktop4">
  <Applications>
    <Application>
      <Extensions>
```

```
        <com:Extension Category="windows.comServer">
          <com:ComServer>
            <com:SurrogateServer AppId="00001111-aaaa-2222-bbbb-
3333cccc4444" DisplayName="ContosoHandler">
              <com:Class Id="00001111-aaaa-2222-bbbb-3333cccc4444"
Path="ExplorerCommandVerb.dll" ThreadingModel="STA"/>
            </com:SurrogateServer>
          </com:ComServer>
        </com:Extension>
        <desktop4:Extension Category="windows.fileExplorerContextMenus">
          <desktop4:FileExplorerContextMenus>
            <desktop4:ItemType Type=".contoso">
              <desktop4:Verb Id="Command1" Clsid="00001111-aaaa-2222-bbbb-
3333cccc4444" />
            </desktop4:ItemType>
          </desktop4:FileExplorerContextMenus>
        </desktop4:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Make files from your cloud service appear in File Explorer

Register the handlers that you implement in your application. You can also add context menu options that appear when you users right-click your cloud-based files in File Explorer.

XML namespace

- `http://schemas.microsoft.com/appx/manifest/desktop/windows10`

Elements and attributes of this extension

XML

```
<Extension Category="windows.cloudfiles" >
    <CloudFiles IconResource="[Icon]">
        <CustomStateHandler Clsid ="[Clsid]"/>
        <ThumbnailProviderHandler Clsid ="[Clsid]"/>
        <ExtendedPropertyhandler Clsid ="[Clsid]"/>
        <CloudFilesContextMenus>
            <Verb Id ="Command3" Clsid= "[GUID]">[Verb Label]</Verb>
        </CloudFilesContextMenus>
    </CloudFiles>
</Extension>
```

Expand table

Name	Description
Category	Always `windows.cloudfiles`.
iconResource	The icon that represents your cloud file provider service. This icon appears in the Navigation pane of File Explorer. Users choose this icon to show files from your cloud service.
CustomStateHandler Clsid	The class ID of the application that implements the CustomStateHandler. The system uses this Class ID to request custom states and columns for cloud files.
ThumbnailProviderHandler Clsid	The class ID of the application that implements the ThumbnailProviderHandler. The system uses this Class ID to request thumbnail images for cloud files.
ExtendedPropertyHandler Clsid	The class ID of the application that implements the ExtendedPropertyHandler. The system uses this Class ID to request extended properties for a cloud file.
Verb	The name that appears in the File Explorer context menu for files provided by your cloud service.
Id	The unique ID of the verb.

Example

XML

```
<Package

xmlns:desktop="http://schemas.microsoft.com/appx/manifest/desktop/windows10"
    IgnorableNamespaces="desktop">
  <Applications>
    <Application>
      <Extensions>
        <Extension Category="windows.cloudfiles" >
            <CloudFiles IconResource="images\Wide310x150Logo.png">
                <CustomStateHandler Clsid ="20000000-0000-0000-0000-
000000000001"/>
                <ThumbnailProviderHandler Clsid ="20000000-0000-0000-0000-
000000000001"/>
                <ExtendedPropertyhandler Clsid ="20000000-0000-0000-0000-
000000000001"/>
                <desktop:CloudFilesContextMenus>
                    <desktop:Verb Id ="keep" Clsid=
                       "20000000-0000-0000-0000-000000000001">
                       Always keep on this device</desktop:Verb>
                </desktop:CloudFilesContextMenus>
```

```
            </CloudFiles>
          </Extension>
        </Extensions>
      </Application>
    </Applications>
</Package>
```

Start your application in different ways

- Start your application by using a protocol
- Start your application by using an alias
- Start an executable file when users log into Windows
- Enable users to start your application when they connect a device to their PC
- Restart automatically after receiving an update from the Microsoft Store

Start your application by using a protocol

Protocol associations can enable other programs and system components to interoperate with your packaged app. When your packaged application is started by using a protocol, you can specify specific parameters to pass to its activation event arguments so it can behave accordingly. Parameters are supported only for packaged, full-trust apps. UWP apps can't use parameters.

XML namespace

`http://schemas.microsoft.com/appx/manifest/uap/windows10/3`

Elements and attributes of this extension

XML

```
<Extension
    Category="windows.protocol">
  <Protocol
      Name="[Protocol name]"
      Parameters="[Parameters]" />
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.protocol`.
Name	The name of the protocol.
Parameters	The list of parameters and values to pass to your application as event arguments when the application is activated. If a variable can contain a file path, wrap the parameter value in quotes. That will avoid any issues that happen in cases where the path includes spaces.

Example

XML

```
<Package
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"

xmlns:desktop="http://schemas.microsoft.com/appx/manifest/desktop/windows10"
  IgnorableNamespaces="uap3, desktop">
  <Applications>
    <Application>
      <Extensions>
        <uap3:Extension
          Category="windows.protocol">
          <uap3:Protocol
            Name="myapp-cmd"
            Parameters="/p &amp;quot;%1&amp;quot;" />
        </uap3:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Start your application by using an alias

Users and other processes can use an alias to start your application without having to specify the full path to your app. You can specify that alias name.

XML namespaces

- `http://schemas.microsoft.com/appx/manifest/uap/windows10/3`
- `http://schemas.microsoft.com/appx/manifest/desktop/windows10`

Elements and attributes of this extension

XML

```xml
<uap3:Extension
    Category="windows.appExecutionAlias"
    Executable="[ExecutableName]"
    EntryPoint="Windows.FullTrustApplication">
    <uap3:AppExecutionAlias>
        <desktop:ExecutionAlias Alias="[AliasName]" />
    </uap3:AppExecutionAlias>
</uap3:Extension>
```

Expand table

Name	Description
Category	Always `windows.appExecutionAlias`.
Executable	The relative path to the executable to start when the alias is invoked.
Alias	The short name for your app. It must always end with the ".exe" extension. You can only specify a single app execution alias for each application in the package. If multiple apps register for the same alias, the system will invoke the last one that was registered, so make sure to choose a unique alias other apps are unlikely to override.

Example

XML

```xml
<Package
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
  IgnorableNamespaces="uap3">
  <Applications>
    <Application>
      <Extensions>
         <uap3:Extension
                Category="windows.appExecutionAlias"
                Executable="exes\launcher.exe"
                EntryPoint="Windows.FullTrustApplication">
            <uap3:AppExecutionAlias>
                <desktop:ExecutionAlias Alias="Contoso.exe" />
            </uap3:AppExecutionAlias>
        </uap3:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Find the complete schema reference here.

Start an executable file when users log into Windows

Startup tasks allow your application to run an executable automatically whenever a user logs on.

> **Note**
>
> The user has to start your application at least one time to register this startup task.

Your application can declare multiple startup tasks. Each task starts independently. All startup tasks will appear in Task Manager under the **Startup** tab with the name that you specify in your app's manifest and your app's icon. Task Manager will automatically analyze the startup impact of your tasks.

Users can manually disable your app's startup task by using Task Manager. If a user disables a task, you can't programmatically re-enable it.

XML namespace

`http://schemas.microsoft.com/appx/manifest/desktop/windows10`

Elements and attributes of this extension

XML

```xml
<Extension
    Category="windows.startupTask"
    Executable="[ExecutableName]"
    EntryPoint="Windows.FullTrustApplication">
  <StartupTask
      TaskId="[TaskID]"
      Enabled="true"
      DisplayName="[DisplayName]" />
</Extension>
```

Expand table

Name	Description
Category	Always `windows.startupTask`.
Executable	The relative path to the executable file to start.

Name	Description
TaskId	A unique identifier for your task. Using this identifier, your application can call the APIs in the Windows.ApplicationModel.StartupTask class to programmatically enable or disable a startup task.
Enabled	Indicates whether the task first starts enabled or disabled. Enabled tasks will run the next time the user logs on (unless the user disables it).
DisplayName	The name of the task that appears in Task Manager. You can localize this string by using `ms-resource`.

Example

XML

```
<Package

xmlns:desktop="http://schemas.microsoft.com/appx/manifest/desktop/windows10"
  IgnorableNamespaces="desktop">
  <Applications>
    <Application>
      <Extensions>
        <desktop:Extension
          Category="windows.startupTask"
          Executable="bin\MyStartupTask.exe"
          EntryPoint="Windows.FullTrustApplication">
        <desktop:StartupTask
          TaskId="MyStartupTask"
          Enabled="true"
          DisplayName="My App Service" />
        </desktop:Extension>
      </Extensions>
    </Application>
  </Applications>
 </Package>
```

Enable users to start your application when they connect a device to their PC

AutoPlay can present your application as an option when a user connects a device to their PC.

XML namespace

`http://schemas.microsoft.com/appx/manifest/desktop/windows10/3`

Elements and attributes of this extension

XML

```
<Extension Category="windows.autoPlayHandler">
  <AutoPlayHandler>
    <InvokeAction ActionDisplayName="[action string]" ProviderDisplayName="
[name of your app/service]">
      <Content ContentEvent="[Content event]" Verb="[any string]"
DropTargetHandler="[Clsid]" />
      <Content ContentEvent="[Content event]" Verb="[any string]"
Parameters="[Initialization parameter]"/>
      <Device DeviceEvent="[Device event]" HWEventHandler="[Clsid]"
InitCmdLine="[Initialization parameter]"/>
    </InvokeAction>
  </AutoPlayHandler>
```

Expand table

Name	Description
Category	Always `windows.autoPlayHandler`.
ActionDisplayName	A string that represents the action that users can take with a device that they connect to a PC (For example: "Import files", or "Play video").
ProviderDisplayName	A string that represents your application or service (For example: "Contoso video player").
ContentEvent	The name of a content event that causes users to be prompted with your `ActionDisplayName` and `ProviderDisplayName`. A content event is raised when a volume device such as a camera memory card, thumb drive, or DVD is inserted into the PC. You can find the full list of those events here.
Verb	The Verb setting identifies a value that is passed to your application for the selected option. You can specify multiple launch actions for an AutoPlay event and use the Verb setting to determine which option a user has selected for your app. You can tell which option the user selected by checking the verb property of the startup event arguments passed to your app. You can use any value for the Verb setting except, open, which is reserved.
DropTargetHandler	The class ID of the application that implements the IDropTarget interface. Files from the removable media are passed to the Drop method of your IDropTarget implementation.
Parameters	You don't have to implement the IDropTarget interface for all content events. For any of the content events, you could provide command line parameters instead of implementing the IDropTarget interface. For those events, AutoPlay will start your application by using those command line

Name	Description
	parameters. You can parse those parameters in your app's initialization code to determine if it was started by AutoPlay and then provide your custom implementation.
DeviceEvent	The name of a device event that causes users to be prompted with your `ActionDisplayName` and `ProviderDisplayName`. A device event is raised when a device is connected to the PC. Device events begin with the string `WPD` and you can find them listed here.
HWEventHandler	The Class ID of the application that implements the IHWEventHandler interface.
InitCmdLine	The string parameter that you want to pass into the Initialize method of the IHWEventHandler interface.

Example

XML

```
<Package

xmlns:desktop3="http://schemas.microsoft.com/appx/manifest/desktop/windows10
/3"
  IgnorableNamespaces="desktop3">
  <Applications>
    <Application>
      <Extensions>
        <desktop3:Extension Category="windows.autoPlayHandler">
          <desktop3:AutoPlayHandler>
            <desktop3:InvokeAction ActionDisplayName="Import my files"
ProviderDisplayName="ms-resource:AutoPlayDisplayName">
              <desktop3:Content ContentEvent="ShowPicturesOnArrival"
Verb="show" DropTargetHandler="CD041BAE-0DEA-4472-9B7B-C98043D26EA8"/>
              <desktop3:Content ContentEvent="PlayVideoFilesOnArrival"
Verb="play" Parameters="%1" />
              <desktop3:Device DeviceEvent="WPD\ImageSource"
HWEventHandler="CD041BAE-0DEA-4472-9B7B-C98043D26EA8"
InitCmdLine="/autoplay"/>
            </desktop3:InvokeAction>
          </desktop3:AutoPlayHandler>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Restart automatically after receiving an update from the Microsoft Store

If your application is open when users install an update to it, the application closes.

If you want that application to restart after the update completes, call the RegisterApplicationRestart function in every process that you want to restart.

Each active window in your application receives a WM_QUERYENDSESSION message. At this point, your application can call the RegisterApplicationRestart function again to update the command line if necessary.

When each active window in your application receives the WM_ENDSESSION message, your application should save data and shut down.

> **Note**
>
> Your active windows also receive the **WM_CLOSE** message in case the application doesn't handle the **WM_ENDSESSION** message.

At this point, your application has 30 seconds to close it's own processes or the platform terminates them forcefully.

After the update is complete, your application restarts.

Work with other applications

Integrate with other apps, start other processes or share information.

- Make your application appear as the print target in applications that support printing
- Share fonts with other Windows applications
- Start a Win32 process from a Universal Windows Platform (UWP) app

Make your application appear as the print target in applications that support printing

When users want to print data from another application such as Notepad, you can make your application appear as a print target in the app's list of available print targets.

You'll have to modify your application so that it receives print data in XML Paper Specification (XPS) format.

XML namespaces

`http://schemas.microsoft.com/appx/manifest/desktop/windows10/2`

Elements and attributes of this extension

XML

```
<Extension Category="windows.appPrinter">
    <AppPrinter
        DisplayName="[DisplayName]"
        Parameters="[Parameters]" />
</Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.appPrinter`.
DisplayName	The name that you want to appear in the list of print targets for an app.
Parameters	Any parameters that your application requires to properly handle the request.

Example

XML

```
<Package

xmlns:desktop2="http://schemas.microsoft.com/appx/manifest/desktop/windows10
/2"
  IgnorableNamespaces="desktop2">
  <Applications>
  <Application>
    <Extensions>
      <desktop2:Extension Category="windows.appPrinter">
        <desktop2:AppPrinter
          DisplayName="Send to Contoso"
          Parameters="/insertdoc %1" />
      </desktop2:Extension>
    </Extensions>
  </Application>
</Applications>
</Package>
```

Find a sample that uses this extension Here

Share fonts with other Windows applications

Share your custom fonts with other Windows applications.

> **Note**
>
> Before you can submit an app that uses this extension to the Store, you must first obtain approval from the Store team. To obtain approval, go to https://aka.ms/storesupport, click **Contact us**, and choose options relevant to submitting apps to the dashboard. This approval process helps to ensure that there are no conflicts between fonts installed by your app and fonts that are installed with the OS. If you do not obtain approval, you will receive an error similar to the following when you submit your app: "Package acceptance validation error: You can't use extension windows.sharedFonts with this account. Contact our support team if you'd like to request permissions to use this extension."

XML namespaces

`http://schemas.microsoft.com/appx/manifest/uap/windows10/4`

Elements and attributes of this extension

XML

```xml
<Extension Category="windows.sharedFonts">
    <SharedFonts>
      <Font File="[FontFile]" />
    </SharedFonts>
  </Extension>
```

Find the complete schema reference here.

Expand table

Name	Description
Category	Always `windows.sharedFonts`.
File	The file that contains the fonts that you want to share.

Example

XML

```
<Package
  xmlns:uap4="http://schemas.microsoft.com/appx/manifest/uap/windows10/4"
  IgnorableNamespaces="uap4">
  <Applications>
    <Application>
      <Extensions>
        <uap4:Extension Category="windows.sharedFonts">
          <uap4:SharedFonts>
            <uap4:Font File="Fonts\JustRealize.ttf" />
            <uap4:Font File="Fonts\JustRealizeBold.ttf" />
          </uap4:SharedFonts>
        </uap4:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

Start a Win32 process from a Universal Windows Platform (UWP) app

Start a Win32 process that runs in full-trust.

XML namespaces

`http://schemas.microsoft.com/appx/manifest/desktop/windows10`

Elements and attributes of this extension

XML

```
<Extension Category="windows.fullTrustProcess" Executable="[executable
file]">
  <FullTrustProcess>
    <ParameterGroup GroupId="[GroupID]" Parameters="[Parameters]"/>
  </FullTrustProcess>
</Extension>
```

Expand table

Name	Description
Category	Always `windows.fullTrustProcess`.
GroupID	A string that identifies a set of parameters that you want to pass to the executable.

Name	Description
Parameters	Parameters that you want to pass to the executable.

Example

XML

```
<Package
xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"

xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
         xmlns:rescap=
"http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedc
apabilities"

xmlns:desktop="http://schemas.microsoft.com/appx/manifest/desktop/windows10"
>
  ...
  <Capabilities>
      <rescap:Capability Name="runFullTrust"/>
  </Capabilities>
  <Applications>
    <Application>
      <Extensions>
          <desktop:Extension Category="windows.fullTrustProcess"
Executable="fulltrustprocess.exe">
              <desktop:FullTrustProcess>
                  <desktop:ParameterGroup GroupId="SyncGroup"
Parameters="/Sync"/>
                  <desktop:ParameterGroup GroupId="OtherGroup"
Parameters="/Other"/>
              </desktop:FullTrustProcess>
           </desktop:Extension>
      </Extensions>
    </Application>
  </Applications>
</Package>
```

This extension might be useful if you want to create a Universal Windows Platform User interface that runs on all devices, but you want components of your Win32 application to continue running in full-trust.

Just create a Windows app package for your Win32 app. Then, add this extension to the package file of your UWP app. This extensions indicates that you want to start an executable file in the Windows app package. If you want to communicate between your UWP app and your Win32 app, you can set up one or more app services to do that. You can read more about this scenario here.

Next steps

Have questions? Ask us on Stack Overflow. Our team monitors these tags. You can also ask us here.

Integrate a packaged desktop app with File Explorer

Article • 05/03/2022

Some Windows apps define File Explorer extensions that add context menu entries that enable customers to perform options related to the app. Older Windows app deployment technologies such MSI and ClickOnce define File Explorer extensions through the registry. The registry has a series of hives which control File Explorer extensions and other types of Shell extensions. These installers typically create a series of registry keys to configure the various items to include in the context menu.

If you package your Windows app using MSIX, the registry is virtualized, and therefore your app can't register File Explorer extensions via the registry. Instead, you must define your File Explorer extensions via package extensions, which you define in the package manifest. This article describes several ways to do this.

You can find the complete sample code used in this article on GitHub.

Add a context menu entry that supports startup parameters

One of the simplest ways to integrate with File Explorer is to define a package extension that adds your app to the list of available apps in the context menu when a user right-clicks a specific file type in File Explorer. If the user opens your app, your extension can pass parameters to your app.

This scenario has several limitations:

- It works only in combination with the file type association feature. You can display additional options in the context menu only for file types which are associated with the main app (for example, your app supports opening a file by double-clicking it in File Explorer).
- The options in the context menu will be displayed only if your app is set as default for that file type.
- The only supported action is to launch the main executable of the app (that is, the same executable that is connected to the Start menu entry). However, every action can specify different parameters, which you can use when the apps start to understand which action triggered the execution and perform different tasks.

Despite these limitations, this approach is sufficient for many scenarios. For example, if you're building an image editor, you can easily add an entry in the context menu to resize an image, which will launch the image editor directly with a wizard to start the resizing process.

Implement the context menu entry

To support this scenario, add an Extension element with the category `windows.fileTypeAssociation` to your package manifest. This element must be added as a child of the Extensions element under the Application element.

The following example demonstrates a registration for an app that enables context menus for files with the `.foo` extension. This example specifies the `.foo` extension because this is a fake extension that is typically not registered to other apps on any given computer. If you need to manage a file type which might already be taken (like .txt or .jpg), remember that you won't be able to see the option until your app is set as default for that file type. This example is an excerpt from the Package.appxmanifest file in the related sample on GitHub.

XML

```
<Extensions>
  <uap3:Extension Category="windows.fileTypeAssociation">
    <uap3:FileTypeAssociation Name="foo" Parameters="&quot;%1&quot;">
      <uap:SupportedFileTypes>
        <uap:FileType>.foo</uap:FileType>
      </uap:SupportedFileTypes>
      <uap2:SupportedVerbs>
        <uap3:Verb Id="Resize" Parameters="&quot;%1&quot; /p">Resize
file</uap3:Verb>
      </uap2:SupportedVerbs>
    </uap3:FileTypeAssociation>
  </uap3:Extension>
</Extensions>
```

This example assumes that the following namespaces and aliases are declared in the root `<Package>` element in the manifest.

XML

```
<Package
  xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
  xmlns:uap2="http://schemas.microsoft.com/appx/manifest/uap/windows10/2"
  xmlns:uap3="http://schemas.microsoft.com/appx/manifest/uap/windows10/3"
```

```
xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities"
  IgnorableNamespaces="uap uap2 uap3 rescap">
  ...
</Package>
```

The FileTypeAssociation element associates your app with the file type(s) you want to support. For more details, see Associate your packaged application with a set of file types. Here are the most important items related to this element.

Attribute or element	Description
`Name` attribute	Matches the name of the extension you want to register minus the dot (in the previous example, `foo`).
`Parameters` attribute	Contains the parameters you want to pass to your application when the user double-clicks a file with such extension. Typically, at least, you pass `%1`, which is a special parameter that contains the path of the selected file. This way, when you double click on a file, the application knows its full path and can load it.
SupportedFileTypes element	Specifies the name(s) of the extension you want to register, including the dot (in this example, `.foo`). You can specify multiple `<FileType>` entries you want to support more file types.

To define the context menu integration, you must also add the SupportedVerbs child element. This element contains one or more Verb elements that define the options that will be listed when a user right-clicks a file with the .foo extension in File Explorer. For more details, see Add options to the context menus of files that have a certain file type. Here are the most important items related to the Verb element.

Attribute or element	Description
`Id` attribute	Specifies the unique identifier for the action.
`Parameters` attribute	Similar to the FileTypeAssociation element, this attribute for the Verb element contains the parameters that are passed to your application when the user clicks the context menu entry. Typically, other than the `%1` special parameter to get the path of the selected file, you pass also one or more parameters to get the context. This enables your app to understand that it was opened from a context menu entry.
Element value	The value of the Verb element contains the label to display in the context menu entry (in this example, **Resize file**).

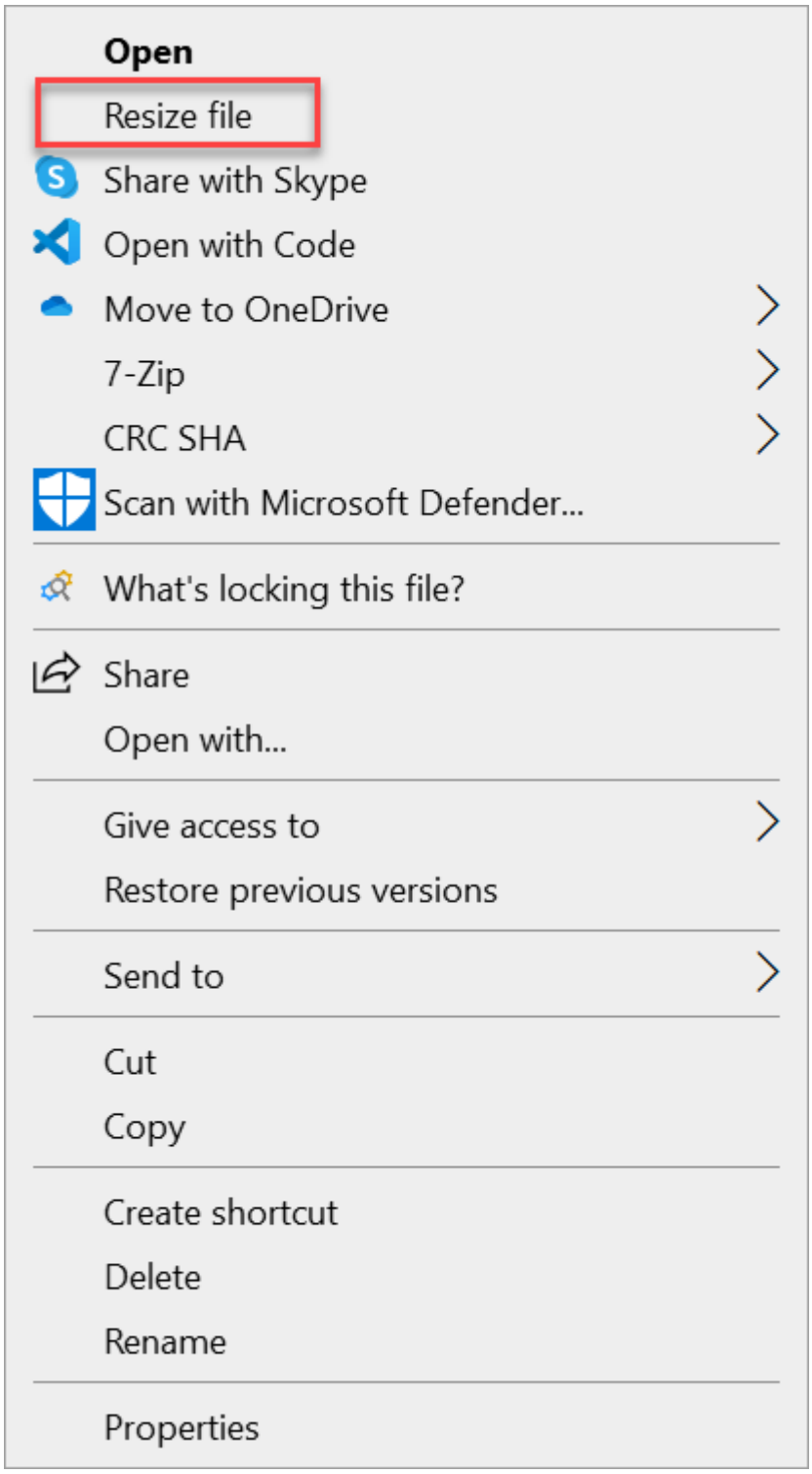
Access the startup parameters in your app code

The way your app receives the parameters depends on the type of app you have created. For example, a WPF app typically processes startup event args in the `OnStartup` method of the `App` class. You can check if there are startup parameters and, based on the outcome, take the most appropriate action (like opening a specific window of the application instead of the main one).

C#

```
public partial class App : Application
{
    protected override void OnStartup(StartupEventArgs e)
    {
        if (e.Args.Contains("Resize"))
        {
            // Open a specific window of the app.
        }
        else
        {
            MainWindow main = new MainWindow();
            main.Show();
        }
    }
}
```

The following screenshot demonstrates the **Resize File** context menu entry created by the previous example.



Support generic files or folders and perform complex tasks

Although using the FileTypeAssociation extension in the package manifest as described in the previous section is sufficient for many scenarios, you may find it limiting. The two biggest challenges are:

- You can handle only file types you are associated with. For example, you can't handle a generic folder.

- You can only launch the app with a series of parameters. You can't perform advanced operations, like launching another executable or performing a task without opening the main app.

To achieve these goals, you must create a Shell extension, which provides more powerful ways to integrate with File Explorer. In this scenario, you create a DLL that contains everything required to manage the file context menu, including the label, icon, state, and tasks to perform. Because this functionality is implemented in a DLL, you can do nearly everything that you can do with a normal app. After you implement the DLL, you must register it via extensions you define in your package manifest.

> **Note**
>
> The process described in this section has one limitation. After the MSIX package that contains the extension is installed on a target computer, File Explorer must be restarted before the Shell extension can be loaded. To accomplish this, the user can restart the computer, or they can restart the **explorer.exe** process using **Task Manager**.

Implement the Shell extension

Shell extensions are based on COM (Component Object Model). Your DLL exposes one or more COM objects that are registered in the system registry. Windows discovers these COM objects and integrates your extension with File Explorer. Because you're integrating your code with the Windows Shell, performance and memory footprint is important. Therefore, these kinds of extensions are typically built with C++.

For sample code that illustrates how to implement Shell extensions, see the ExplorerCommandVerb project in the related sample on GitHub. This project is based on this sample in the Windows desktop samples, and it has several revisions to make the sample more easy to use with the latest versions of Visual Studio.

This project contains a lot of boilerplate code for different tasks, such as dynamic vs static menus and manual registration of the DLL. Most of this code isn't needed if you're packaging your app using MSIX, because the packaging support will take care of these tasks for you. The ExplorerCommandVerb.cpp file contains the implementation of the context menu, and this is the main code file of interest for this walkthrough.

The key function is `CExplorerCommandVerb::Invoke`. This is the function that is invoked when a user clicks on the entry in the context menu. In the sample, to minimize the

impact on the performance, the operation is performed on another thread, so you will actually find the real implementation in `CExplorerCommandVerb::_ThreadProc`.

C++

```
DWORD CExplorerCommandVerb::_ThreadProc()
{
    IShellItemArray* psia;
    HRESULT hr = CoGetInterfaceAndReleaseStream(_pstmShellItemArray,
IID_PPV_ARGS(&psia));
    _pstmShellItemArray = NULL;
    if (SUCCEEDED(hr))
    {
        DWORD count;
        psia->GetCount(&count);

        IShellItem2* psi;
        HRESULT hr = GetItemAt(psia, 0, IID_PPV_ARGS(&psi));
        if (SUCCEEDED(hr))
        {
            PWSTR pszName;
            hr = psi->GetDisplayName(SIGDN_DESKTOPABSOLUTEPARSING,
&pszName);
            if (SUCCEEDED(hr))
            {
                WCHAR szMsg[128];
                StringCchPrintf(szMsg, ARRAYSIZE(szMsg), L"%d item(s), first
item is named %s", count, pszName);

                MessageBox(_hwnd, szMsg, L"ExplorerCommand Sample Verb",
MB_OK);

                CoTaskMemFree(pszName);
            }

            psi->Release();
        }
        psia->Release();
    }

    return 0;
}
```

When a user right-clicks on a file or folder, this function displays a message box with the full path of the selected file or folder. If you want to customize the Shell extension in other ways, you can extend the following functions in the sample:

- You can change the GetTitle function to customize the label of the entry in the context menu.

- You can change the GetIcon function to customize the icon displayed near the entry in the context menu.
- You can change the GetTooltip function to customize the tooltip which is displayed when you hover the entry in the context menu.

Register the Shell extension

Because the Shell extension is based on COM, the implementation DLL must be exposed as a COM server so that Windows can integrate it with File Explorer. Typically, this is done by assigning a unique ID (called CLSID) to the COM server and by registering it in a specific hive of the system registry. In the ExplorerCommandVerb project, the CLSID for the `CExplorerCommandVerb` extension is defined in the Dll.h file.

C++

```
class __declspec(uuid("CC19E147-7757-483C-B27F-3D81BCEB38FE"))
CExplorerCommandVerb;
```

When you package a Shell extension DLL in an MSIX package, you follow a similar approach. However, the GUID must registered inside the package manifest instead of the registry, as explained here.

In your package manifest, start by adding the following namespaces to your **Package** element.

XML

```
<Package

xmlns:desktop="http://schemas.microsoft.com/appx/manifest/desktop/windows10"

xmlns:desktop4="http://schemas.microsoft.com/appx/manifest/desktop/windows10
/4"

xmlns:desktop5="http://schemas.microsoft.com/appx/manifest/desktop/windows10
/5"
  xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"
  IgnorableNamespaces="desktop desktop4 desktop5 com">

    ...
</Package>
```

To register the CLSID, add a com.Extension element with the category `windows.comServer` to your package manifest. This element must be added as a child of

the Extensions element under the Application element. This example is an excerpt from the Package.appxmanifest file in the related sample on GitHub.

XML

```
<com:Extension Category="windows.comServer">
  <com:ComServer>
    <com:SurrogateServer DisplayName="ContextMenuSample">
      <com:Class Id="CC19E147-7757-483C-B27F-3D81BCEB38FE"
Path="ExplorerCommandVerb.dll" ThreadingModel="STA"/>
    </com:SurrogateServer>
  </com:ComServer>
</com:Extension>
```

There are two critical attributes to configure in the com:Class element.

Attribute	Description
`Id` attribute	This must match with the CLSID of the object you want to register. In this example, this is the CLSID declared in the `Dll.h` file associated to the `CExplorerCommandVerb` class.
`Path` attribute	This must contain the name of the DLL that exposes the COM object. This example includes the DLL in the root of the package, so it can just specify the name of the DLL generated by the `ExplorerCommandVerb` project.

Next, add another extension that registers the file context menu. To do this, add a desktop4:Extension element with the category `windows.fileExplorerContextMenus` to your package manifest. This element must also be added as a child of the Extensions element under the Application element.

XML

```
<desktop4:Extension Category="windows.fileExplorerContextMenus">
  <desktop4:FileExplorerContextMenus>
    <desktop5:ItemType Type="Directory">
      <desktop5:Verb Id="Command1" Clsid="CC19E147-7757-483C-B27F-
3D81BCEB38FE" />
    </desktop5:ItemType>
  </desktop4:FileExplorerContextMenus>
</desktop4:Extension>
```

There are two critical attributes to configure under the desktop4:Extension element.

Attribute or element	Description

Attribute or element	Description
`Type` attribute of desktop5:ItemType	This defines the type of items you want to associate with the context menu. It could be a star (`*`) if you want to display it for all the files; it could be a specific file extension (`.foo`); or it can be available for folders (`Directory`).
`Clsid` attribute of desktop5:Verb	This must match the CLSID that you have previously registered as COM server in the package manifest file.

Configure the DLL in the package

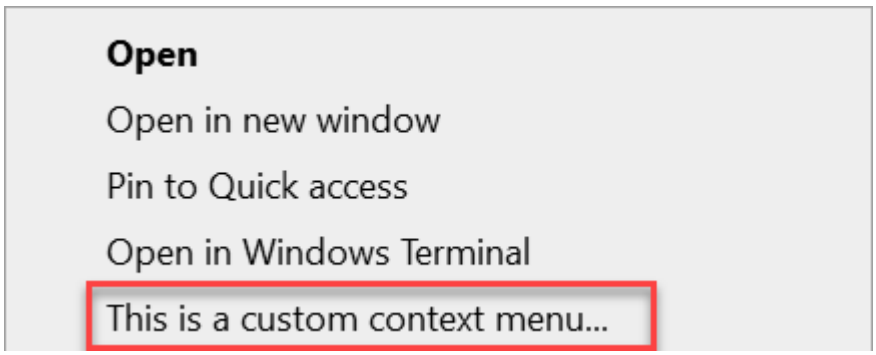
Include the DLL that implements the Shell extension (in this sample, **ExplorerCommandVerb.dll**) in the root of the MSIX package. If you're using the Windows Application Packaging Project, the easiest solution is to copy and paste the DLL into the project and make sure that the **Copy to Output Directory** option for the DLL file properties is set to **Copy if newer**.

To make sure that the package always includes the most recent version of the DLL, you can add a post-build event to the Shell extension project so that, every time you build it, the DLL is copied to the Windows Application Packaging Project.

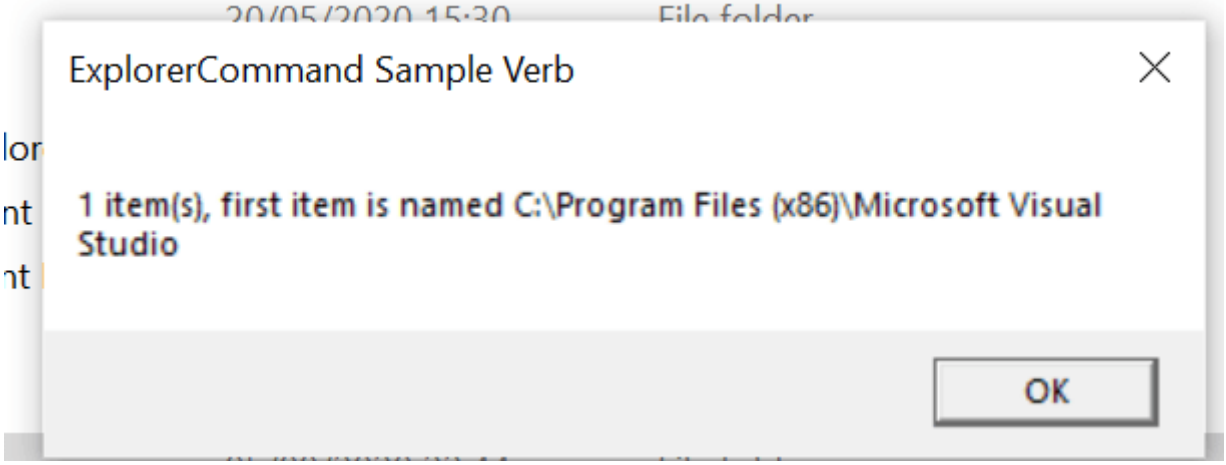
Restart File Explorer

After you install the Shell extension package, you must restart File Explorer before the Shell extension can be loaded. This is a limitation of Shell extensions that are deployed and registered via MSIX packages.

To test the Shell extension, restart your PC or restart the **explorer.exe** process using **Task Manager**. After you do to that, you should be able to see the entry in the context menu.



If you click on it, the `CExplorerCommandVerb::_ThreadProc` function will be called to display the message box with the path of the selected folder.


20/05/2020 15:30
File folder
ExplorerCommand Sample Verb
1 item(s), first item is named C:\Program Files (x86)\Microsoft Visual Studio
OK

Get activation info for packaged apps

Article • 10/24/2022

Starting in Windows 10, version 1809, packaged desktop apps can call the AppInstance.GetActivatedEventArgs method to retrieve certain kinds of app activation info during startup. For example, you can call this method to get info related to app activation from opening a file, clicking an interactive toast, or using a protocol. Starting in Windows 10, version 2004, this feature is also supported in packaged apps with external location (see Grant package identity by packaging with external location).

> **Note**
>
> In addition to retrieving certain types of activation info by using the AppInstance.GetActivatedEventArgs method as described in this article, you can also retrieve activation info for background tasks by defining a COM class. For more info, see Create and register a winmain COM background task.

Code example

The following code example demonstrates how to call the AppInstance.GetActivatedEventArgs method from the **Main** function in a Windows Forms app. For each activation type your app supports, cast the `args` return value to the corresponding event args type. In this code example, the `Handlexxx` methods are assumed to be dedicated activation handler code that you have defined elsewhere.

C#

```
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

    var args = AppInstance.GetActivatedEventArgs();
    switch (args.Kind)
    {
        case ActivationKind.Launch:
            HandleLaunch(args as LaunchActivatedEventArgs);
            break;
        case ActivationKind.ToastNotification:
            HandleToastNotification(args as
ToastNotificationActivatedEventArgs);
            break;
        case ActivationKind.VoiceCommand:
```

```
            HandleVoiceCommand(args as VoiceCommandActivatedEventArgs);
            break;
        case ActivationKind.File:
            HandleFile(args as FileActivatedEventArgs);
            break;
        case ActivationKind.Protocol:
            HandleProtocol(args as ProtocolActivatedEventArgs);
            break;
        case ActivationKind.StartupTask:
            HandleStartupTask(args as StartupTaskActivatedEventArgs);
            break;
        default:
            HandleLaunch(null);
            break;
    }
```

Supported activation types

You can use the AppInstance.GetActivatedEventArgs method to retrieve activation info from the supported set of event args objects listed in the following table. Some of these activation types require the use of a package extension in the package manifest.

ShareTargetActivatedEventArgs activation info is supported only on Windows 10, version 2004, and later. All other activation info types are supported on Windows 10, version 1809, and later.

Expand table

Event args type	Package extension	Related docs
ShareTargetActivatedEventArgs	uap:ShareTarget	Making your desktop application a share target
ProtocolActivatedEventArgs	uap:Protocol	Start your application by using a protocol
ToastNotificationActivatedEventArgs	desktop:ToastNotificationActivation	Toast notifications from desktop apps.
StartupTaskActivatedEventArgs	desktop:StartupTask	Start an executable file when users log into Windows
FileActivatedEventArgs	uap:FileTypeAssociation	Associate your packaged

Event args type	Package extension	Related docs
		application with a set of file types
VoiceCommandActivatedEventArgs	None	Activate a foreground app with voice commands through Cortana
LaunchActivatedEventArgs	None	

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Extend your desktop app with modern UWP components

Article • 01/03/2023

Some Windows experiences (For example, a touch-enabled UI page) must run inside of an AppContainer. If you want to add such experiences, then extend your desktop app with UWP projects and Windows Runtime components.

In many cases you can call Windows Runtime APIs directly from your desktop application, so before you review this guide, see Enhance for Windows.

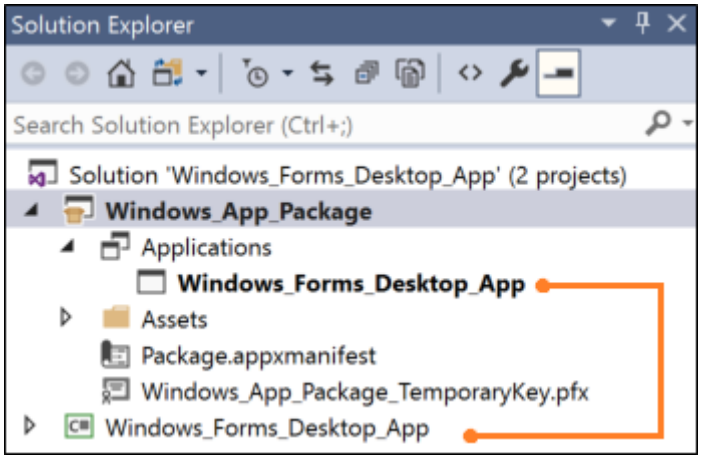
> **Note**
>
> The features described in this topic require that your app is packaged (has package identity at runtime). That includes packaged apps (see Create a new project for a packaged WinUI 3 desktop app) and packaged apps with external location (see Grant package identity by packaging with external location). Also see Features that require package identity.

First, set up your solution

Add one or more UWP projects and runtime components to your solution.

Start with a solution that contains a **Windows Application Packaging Project** with a reference to your desktop application.

This image shows an example solution.



If your solution doesn't contain a packaging project, see Package your desktop application by using Visual Studio.

Configure the desktop application

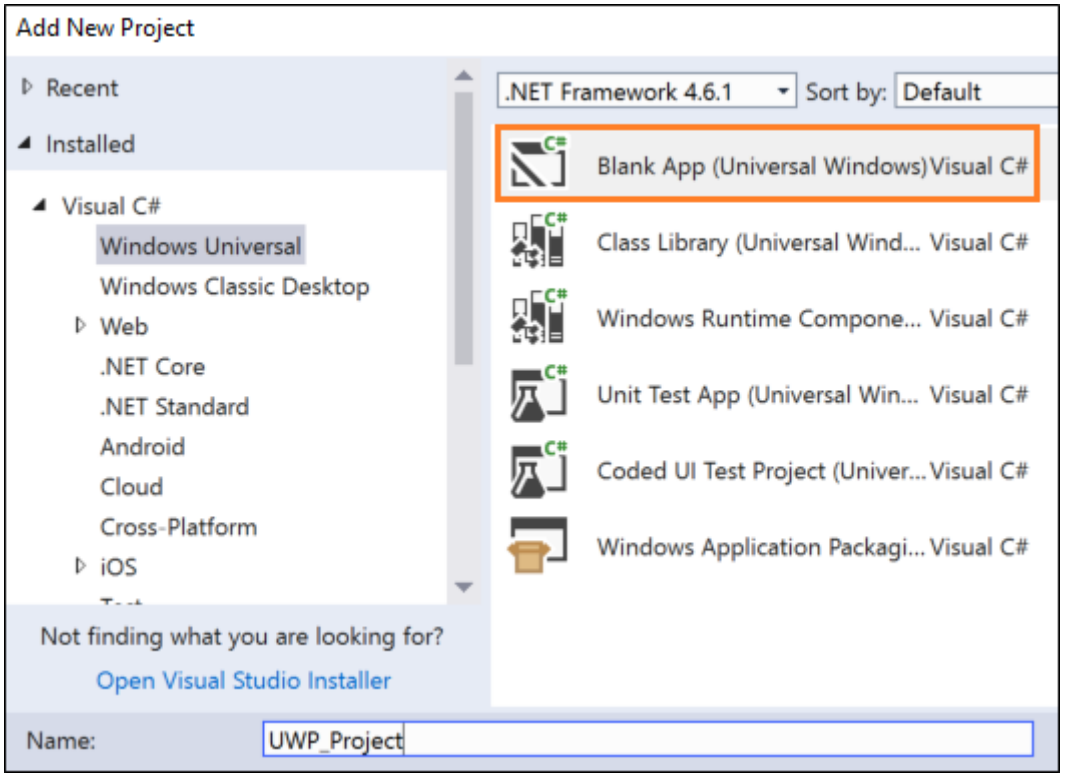
Make sure that your desktop application has references to the files that you need to call Windows Runtime APIs.

To do this, see Call Windows Runtime APIs in desktop apps.

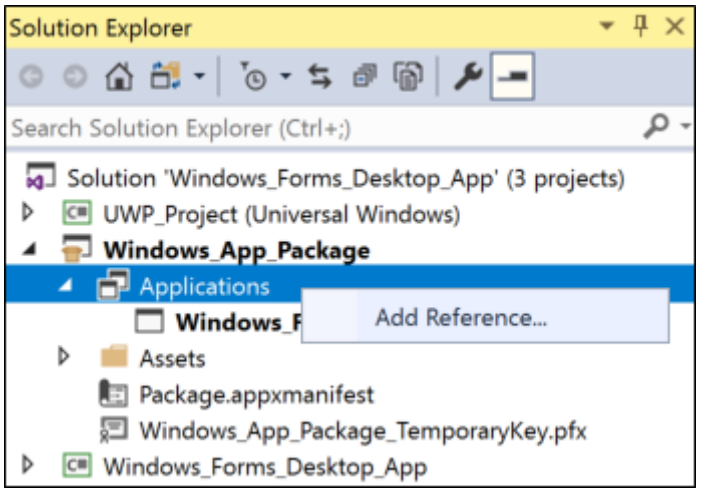
Add a UWP project

Add a **Blank App (Universal Windows)** to your solution.

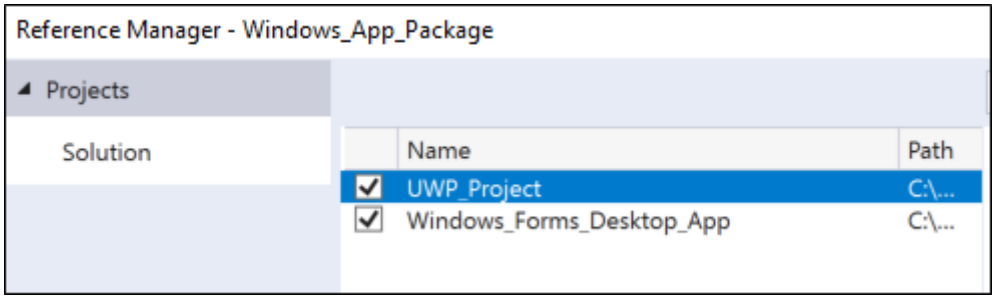
This is where you'll build a modern XAML UI or use APIs that run only within a UWP process.



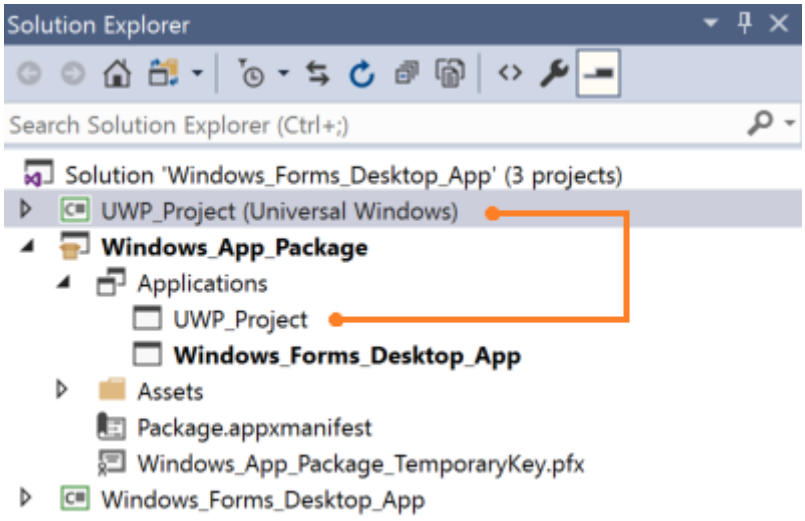
In your packaging project, right-click the **Applications** node, and then click **Add Reference**.



Then, add a reference the UWP project.

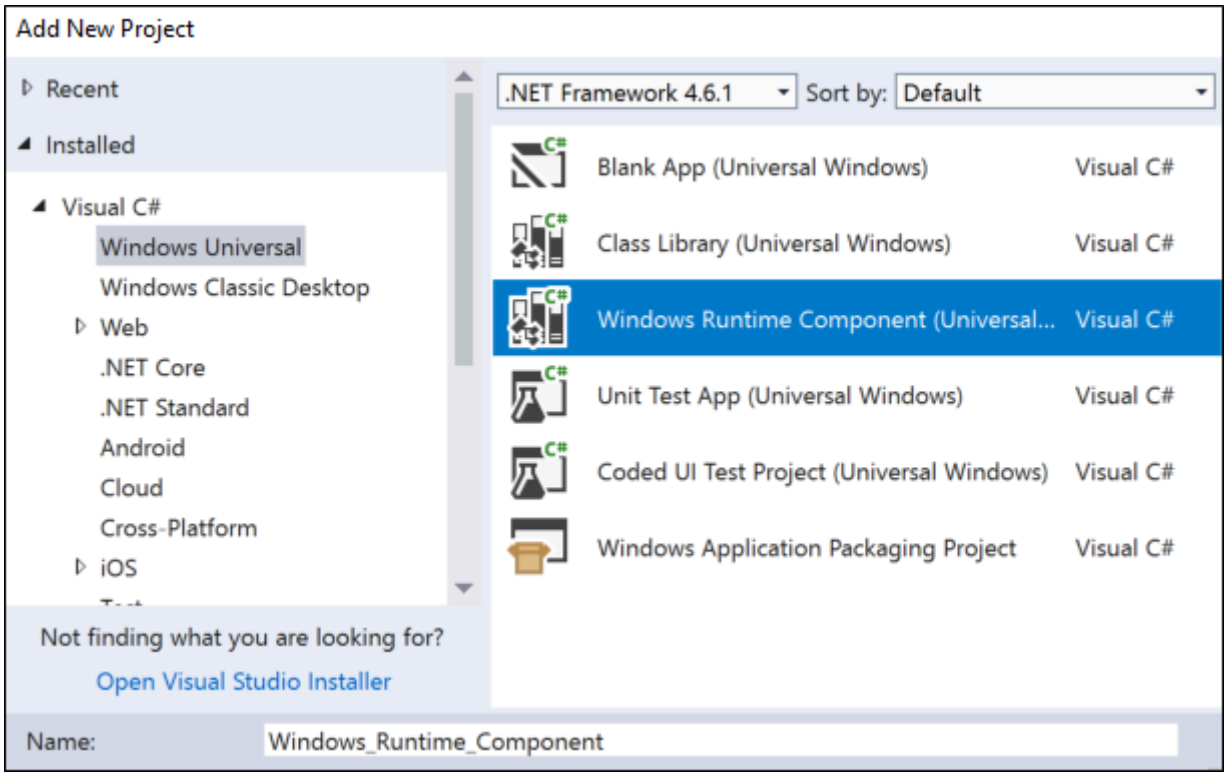


Your solution will look something like this:

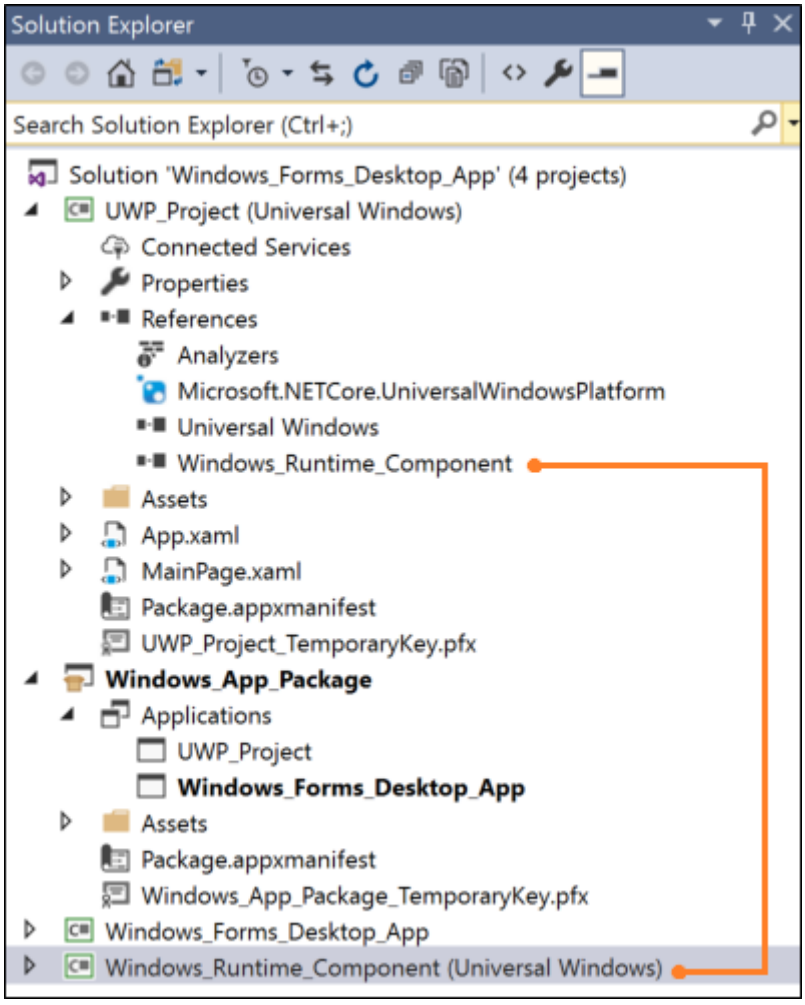


(Optional) Add a Windows Runtime component

To accomplish some scenarios, you'll have to add code to a Windows Runtime component.

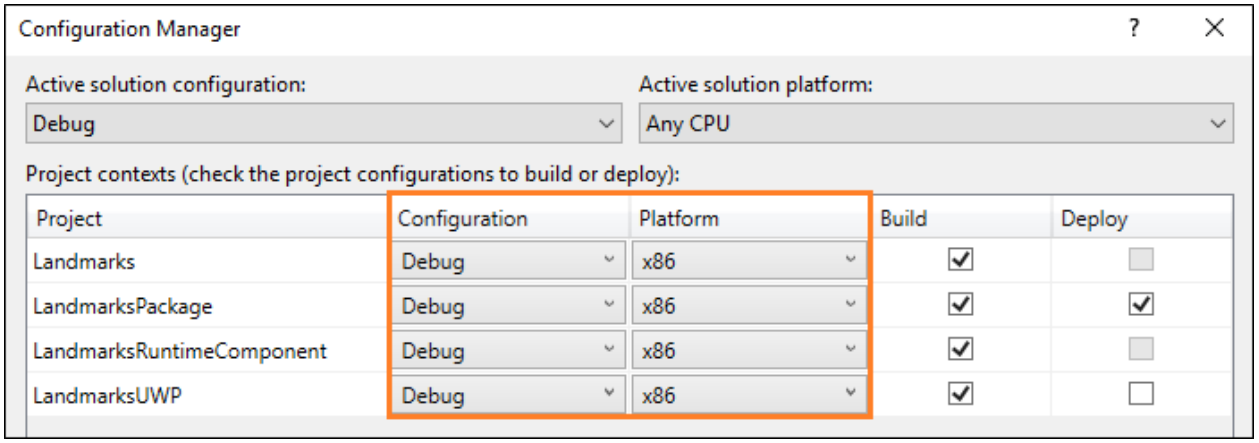


Then, from your UWP project, add a reference to the runtime component. Your solution will look something like this:



Build your solution

Build your solution to ensure that no errors appear. If you receive errors, open **Configuration Manager** and ensure that your projects target the same platform.



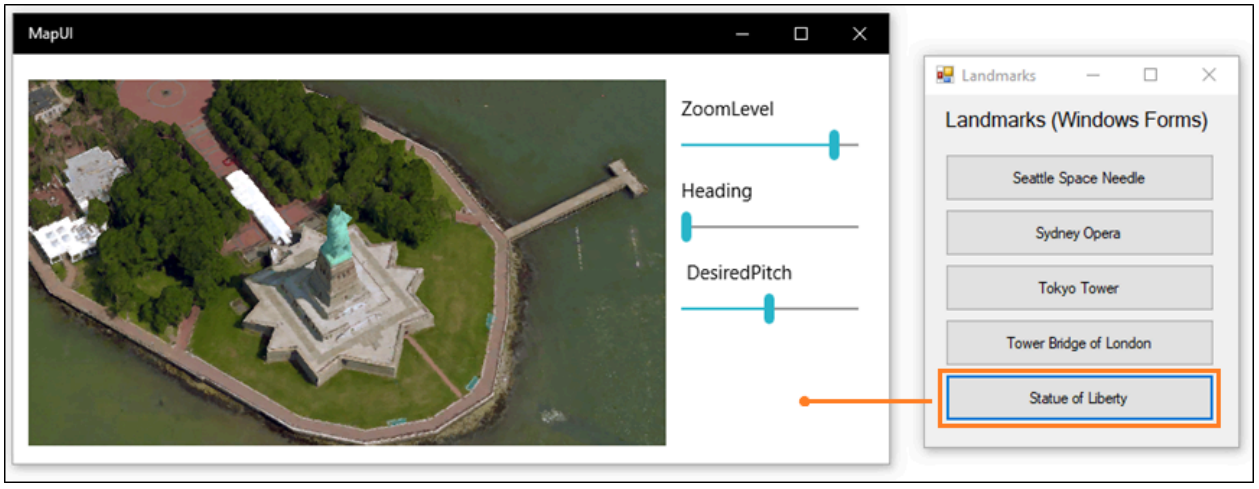
Let's take a look at a few things you can do with your UWP projects and runtime components.

Show a modern XAML UI

As part of your application flow, you can incorporate modern XAML-based user interfaces into your desktop application. These user interfaces are naturally adaptive to different screen sizes and resolutions and support modern interactive models such as touch and ink.

For example, with a small amount of XAML markup, you can give users with powerful map-related visualization features.

This image shows a Windows Forms application that opens a XAML-based modern UI that contains a map control.



Note

This example shows a XAML UI by adding a UWP project to the solution. That is the stable supported approach to showing XAML UIs in a desktop application. The alternative to this approach is to add UWP XAML controls directly to your desktop application by using a XAML Island. XAML Islands are currently available as a developer preview. Although we encourage you to try them out in your own prototype code now, we do not recommend that you use them in production code at this time. These APIs and controls will continue to mature and stabilize in future Windows releases. To learn more about XAML Islands, see UWP controls in desktop applications

The design pattern

To show a XAML-based UI, do these things:

1 Set up your solution

2 Create a XAML UI

3 Add a protocol extension to the UWP project

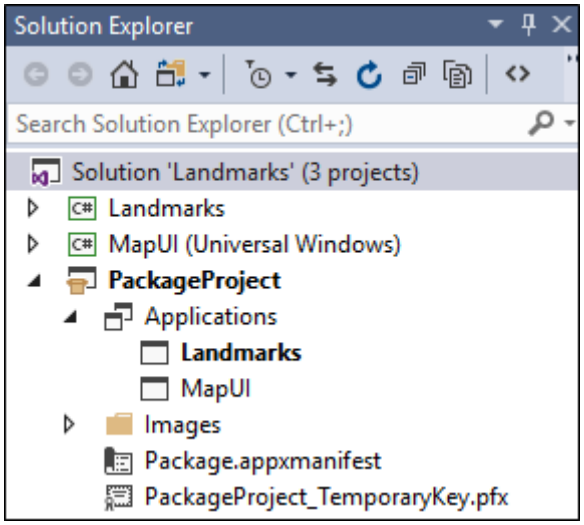
4 Start the UWP app from your desktop app

5 In the UWP project, show the page that you want

Set up your solution

For general guidance on how to set up your solution, see the First, set up your solution section at the beginning of this guide.

Your solution would look something like this:



In this example, the Windows Forms project is named **Landmarks** and the UWP project that contains the XAML UI is named **MapUI**.

Create a XAML UI

Add a XAML UI to your UWP project. Here's the XAML for a basic map.

XML

```
<Grid Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
Margin="12,20,12,14">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>
    <maps:MapControl x:Name="myMap" Grid.Column="0" Width="500" Height="500"
                     ZoomLevel="{Binding ElementName=zoomSlider,Path=Value,
Mode=TwoWay}"
                     Heading="{Binding ElementName=headingSlider,Path=Value,
Mode=TwoWay}"
                     DesiredPitch="{Binding
ElementName=desiredPitchSlider,Path=Value, Mode=TwoWay}"
                     HorizontalAlignment="Left"
                     MapServiceToken="<Your Key Goes Here" />
    <Grid Grid.Column="1" Margin="12">
        <StackPanel>
            <Slider Minimum="1" Maximum="20" Header="ZoomLevel"
Name="zoomSlider" Value="17.5"/>
            <Slider Minimum="0" Maximum="360" Header="Heading"
Name="headingSlider" Value="0"/>
            <Slider Minimum="0" Maximum="64" Header=" DesiredPitch"
Name="desiredPitchSlider" Value="32"/>
        </StackPanel>
    </Grid>
</Grid>
```

Add a protocol extension

In **Solution Explorer**, open the **package.appxmanifest** file of the Packaging project in your solution, and add this extension.

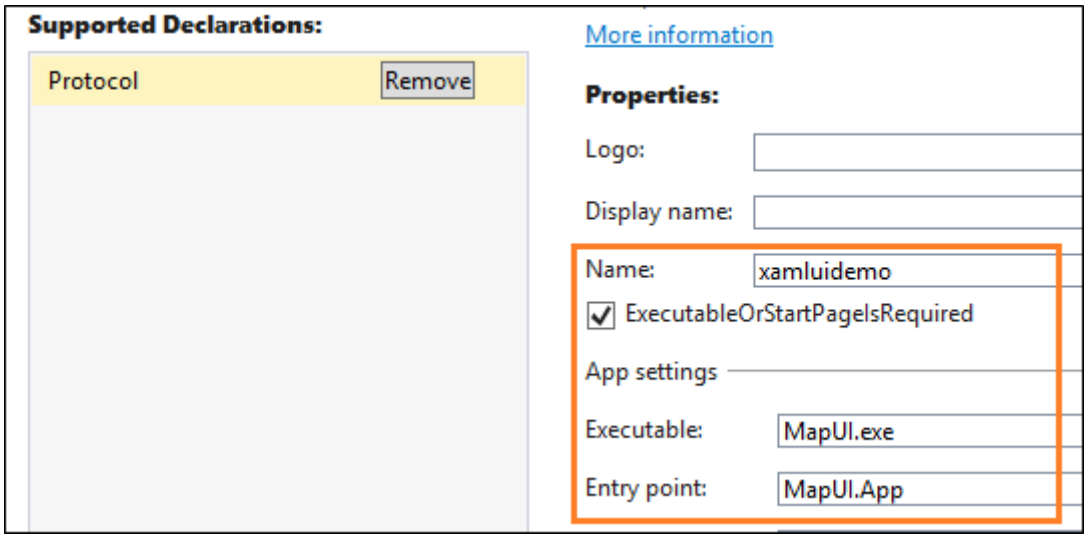
XML

```
<Extensions>
  <uap:Extension Category="windows.protocol" Executable="MapUI.exe"
EntryPoint="MapUI.App">
    <uap:Protocol Name="xamluidemo" />
```

```
  </uap:Extension>
</Extensions>
```

Give the protocol a name, provide the name of the executable produced by the UWP project, and the name of the entry point class.

You can also open the **package.appxmanifest** in the designer, choose the **Declarations** tab, and then add the extension there.



> ⓘ Note
>
> Map controls download data from the internet so if you use one, you'll have to add the "internet client" capability to your manifest as well.

Start the UWP app

First, from your desktop application, create a Uri that includes the protocol name and any parameters you want to pass into the UWP app. Then, call the LaunchUriAsync method.

C#

```
private void Statue_Of_Liberty_Click(object sender, EventArgs e)
{
    ShowMap(40.689247, -74.044502);
}

private async void ShowMap(double lat, double lon)
{
    string str = "xamluidemo://";

    Uri uri = new Uri(str + "location?lat=" +
```

```
        lat.ToString() + "&?lon=" + lon.ToString());

    var success = await Windows.System.Launcher.LaunchUriAsync(uri);

}
```

Parse parameters and show a page

In the **App** class of your UWP project, override the **OnActivated** event handler. If the app is activated by your protocol, parse the parameters and then open the page that you want.

C#

```
protected override void
OnActivated(Windows.ApplicationModel.Activation.IActivatedEventArgs e)
{
    if (e.Kind == ActivationKind.Protocol)
    {
        ProtocolActivatedEventArgs protocolArgs =
(ProtocolActivatedEventArgs)e;
        Uri uri = protocolArgs.Uri;
        if (uri.Scheme == "xamluidemo")
        {
            Frame rootFrame = new Frame();
            Window.Current.Content = rootFrame;
            rootFrame.Navigate(typeof(MainPage), uri.Query);
            Window.Current.Activate();
        }
    }
}
```

In the code behind your XAML page, override the `OnNavigatedTo` method to use the parameters passed into the page. In this case, we'll use the latitude and longitude that were passed into this page to show a location in a map.

C#

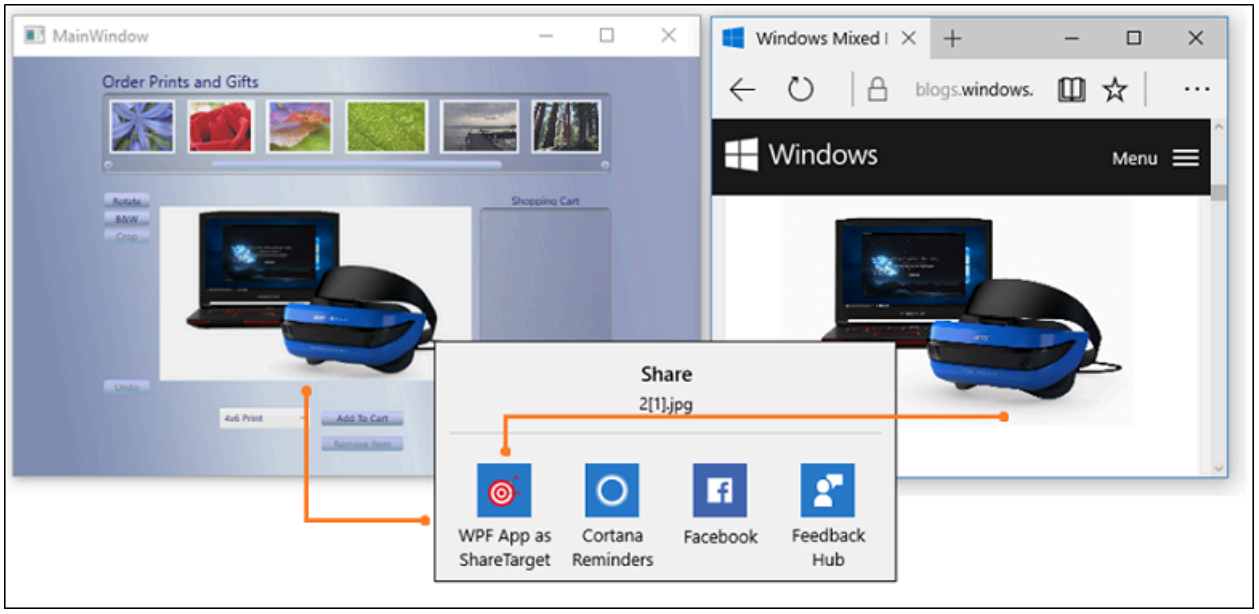
```
protected override void OnNavigatedTo(NavigationEventArgs e)
 {
     if (e.Parameter != null)
     {
         WwwFormUrlDecoder decoder = new
WwwFormUrlDecoder(e.Parameter.ToString());

         double lat = Convert.ToDouble(decoder[0].Value);
         double lon = Convert.ToDouble(decoder[1].Value);

         BasicGeoposition pos = new BasicGeoposition();
```

```
            pos.Latitude = lat;
            pos.Longitude = lon;

            myMap.Center = new Geopoint(pos);

            myMap.Style = MapStyle.Aerial3D;

        }

        base.OnNavigatedTo(e);
    }
```

Making your desktop application a share target

You can make your desktop application a share target so that users can easily share data such as pictures from other apps that support sharing.

For example, users could choose your application to share pictures from Microsoft Edge, the Photos app. Here's a WPF sample application that has that capability.



.

See the complete sample here

The design pattern

To make your application a share target, do these things:

[1] Add a share target extension

[2] Override the OnShareTargetActivated event handler

3 Add desktop extensions to the UWP project

4 Add the full trust process extension

5 Modify the desktop application to get the shared file

The following steps

Add a share target extension

In **Solution Explorer**, open the **package.appxmanifest** file of the Packaging project in your solution and add the share target extension.

XML

```
<Extensions>
      <uap:Extension
          Category="windows.shareTarget"
          Executable="ShareTarget.exe"
          EntryPoint="App">
        <uap:ShareTarget>
          <uap:SupportedFileTypes>
            <uap:SupportsAnyFileType />
          </uap:SupportedFileTypes>
          <uap:DataFormat>Bitmap</uap:DataFormat>
        </uap:ShareTarget>
      </uap:Extension>
</Extensions>
```

Provide the name of the executable produced by the UWP project, and the name of the entry point class. This markup assumes that the name of the executable for your UWP app is `ShareTarget.exe`.

You'll also have to specify what types of files can be shared with your app. In this example, we are making the WPF PhotoStoreDemo desktop application a share target for bitmap images so we specify `Bitmap` for the supported file type.

Override the OnShareTargetActivated event handler

Override the **OnShareTargetActivated** event handler in the **App** class of your UWP project.

This event handler is called when users choose your app to share their files.

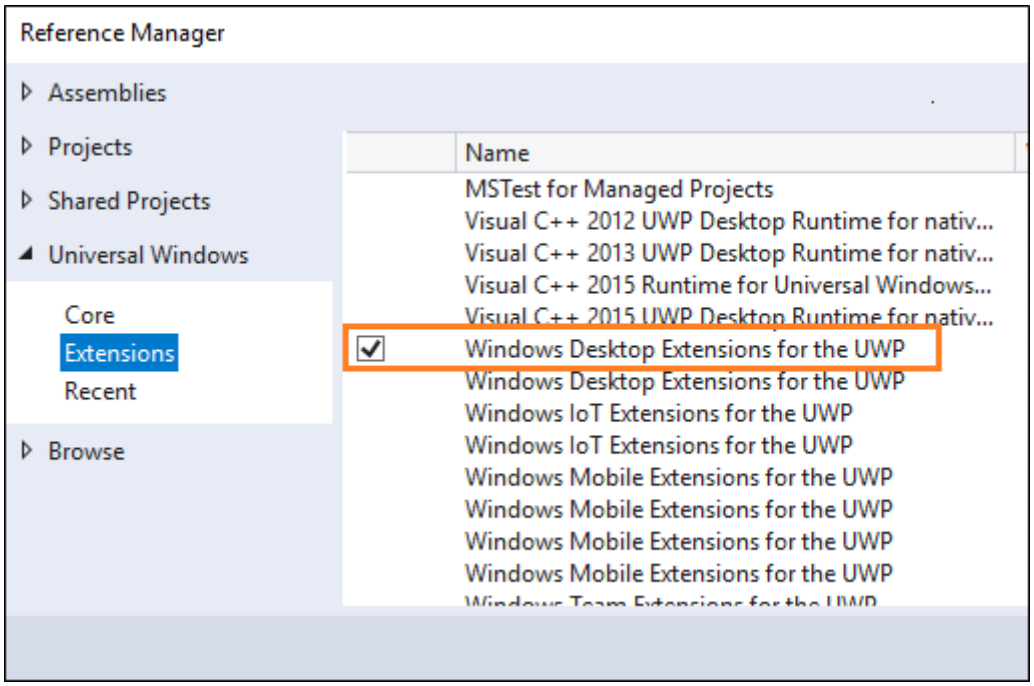
C#

```
protected override void OnShareTargetActivated(ShareTargetActivatedEventArgs
args)
{
    shareWithDesktopApplication(args.ShareOperation);
}

private async void shareWithDesktopApplication(ShareOperation
shareOperation)
{
    if (shareOperation.Data.Contains(StandardDataFormats.StorageItems))
    {
        var items = await shareOperation.Data.GetStorageItemsAsync();
        StorageFile file = items[0] as StorageFile;
        IRandomAccessStreamWithContentType stream = await
file.OpenReadAsync();

        await file.CopyAsync(ApplicationData.Current.LocalFolder);
            shareOperation.ReportCompleted();

        await
FullTrustProcessLauncher.LaunchFullTrustProcessForCurrentAppAsync();
    }
}
```

In this code, we save the image that is being shared by the user into a apps local storage folder. Later, we'll modify the desktop application to pull images from that same folder. The desktop application can do that because it is included in the same package as the UWP app.

Add desktop extensions to the UWP project

Add the **Windows Desktop Extensions for the UWP** extension to the UWP app project. You'll see more than one version of the extension (for example, 10.0.18362.0 and 10.0.19041.0). For info about how to choose a version, see Extension SDKs, and how to reference them.



Add the full trust process extension

In **Solution Explorer**, open the **package.appxmanifest** file of the Packaging project in your solution, and then add the full trust process extension next to the share target extension that you add this file earlier.

XML

```
<Extensions>
  ...
      <desktop:Extension Category="windows.fullTrustProcess"
Executable="PhotoStoreDemo\PhotoStoreDemo.exe" />
  ...
</Extensions>
```

This extension will enable the UWP app to start the desktop application to which you would like the share a file. In example, we refer to the executable of the WPF PhotoStoreDemo desktop application.

Modify the desktop application to get the shared file

Modify your desktop application to find and process the shared file. In this example, the UWP app stored the shared file in the local app data folder. Therefore, we would modify the WPF PhotoStoreDemo desktop application to pull photos from that folder.

C#

```
Photos.Path = Windows.Storage.ApplicationData.Current.LocalFolder.Path;
```

For instances of the desktop application that are already open by the user, we might also handle the FileSystemWatcher event and pass in the path to the file location. That way any open instances of the desktop application will show the shared photo.

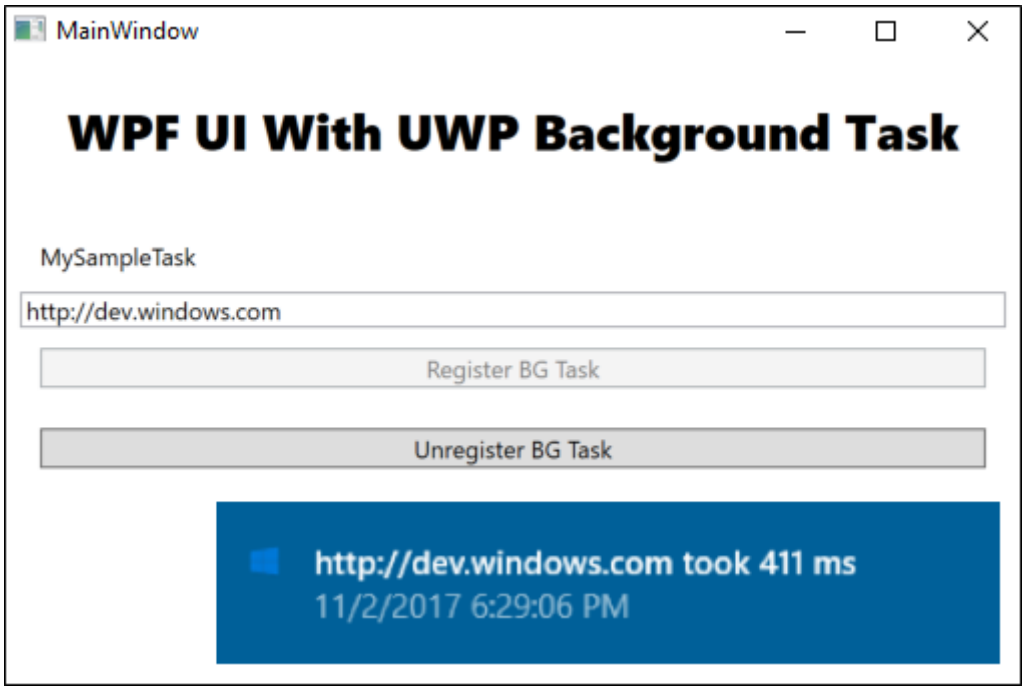
C#

```
...

   FileSystemWatcher watcher = new FileSystemWatcher(Photos.Path);

...

private void Watcher_Created(object sender, FileSystemEventArgs e)
{
    // new file got created, adding it to the list

Dispatcher.BeginInvoke(System.Windows.Threading.DispatcherPriority.Normal,
new Action(() =>
    {
        if (File.Exists(e.FullPath))
        {
            ImageFile item = new ImageFile(e.FullPath);
            Photos.Insert(0, item);
            PhotoListBox.SelectedIndex = 0;
            CurrentPhoto.Source = (BitmapSource)item.Image;
        }
    }));
}
```

Create a background task

You add a background task to run code even when the app is suspended. Background tasks are great for small tasks that don't require the user interaction. For example, your task can download mail, show a toast notification about an incoming chat message, or react to a change in a system condition.

Here's a WPF sample application that registers a background task.



The task makes an http request and measures the time that it takes for the request to return a response. Your tasks will likely be much more interesting, but this sample is great for learning the basic mechanics of a background task.

See the complete sample here.

The design pattern

To create a background service, do these things:

1. Implement the background task
2. Configure the background task
3. Register the background task

Implement the background task

Implement the background task by adding code to a Windows Runtime component project.

C#

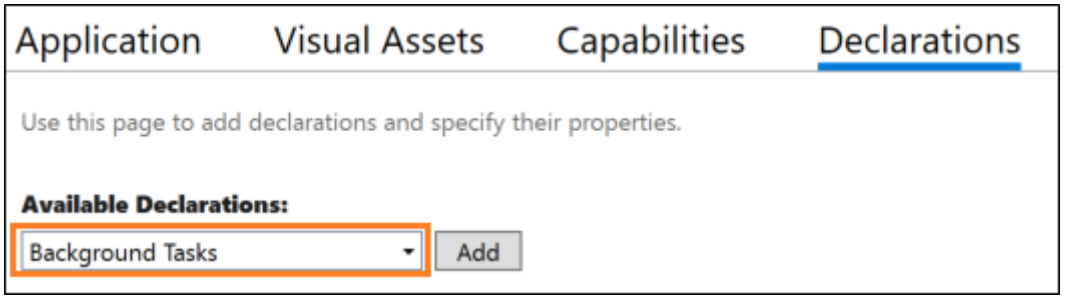
```
public sealed class SiteVerifier : IBackgroundTask
{
    public async void Run(IBackgroundTaskInstance taskInstance)
    {

        taskInstance.Canceled += TaskInstance_Canceled;
        BackgroundTaskDeferral deferral = taskInstance.GetDeferral();
```

```
            var msg = await MeasureRequestTime();
            ShowToast(msg);
            deferral.Complete();
        }

        private async Task<string> MeasureRequestTime()
        {
            string msg;
            try
            {
                var url =
ApplicationData.Current.LocalSettings.Values["UrlToVerify"] as string;
                var http = new HttpClient();
                Stopwatch clock = Stopwatch.StartNew();
                var response = await http.GetAsync(new Uri(url));
                response.EnsureSuccessStatusCode();
                var elapsed = clock.ElapsedMilliseconds;
                clock.Stop();
                msg = $"{url} took {elapsed.ToString()} ms";
            }
            catch (Exception ex)
            {
                msg = ex.Message;
            }
            return msg;
        }
```

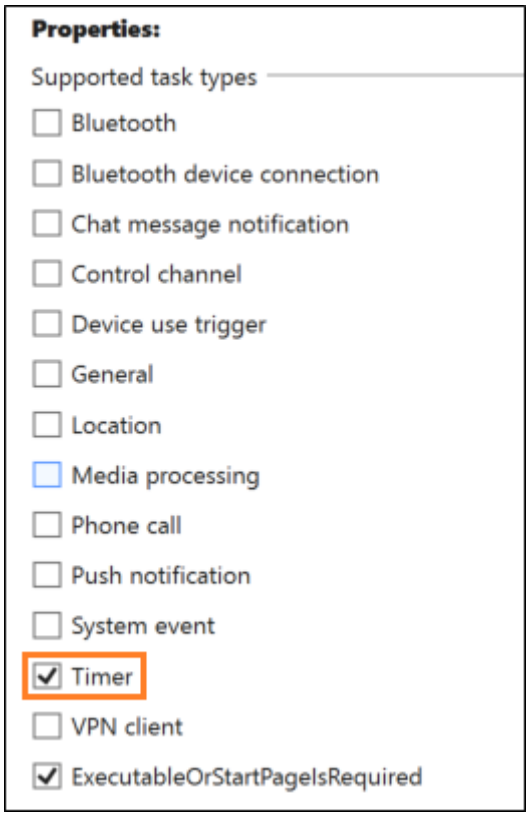
Configure the background task

In the manifest designer, open the **package.appxmanifest** file of the Packaging project in your solution.

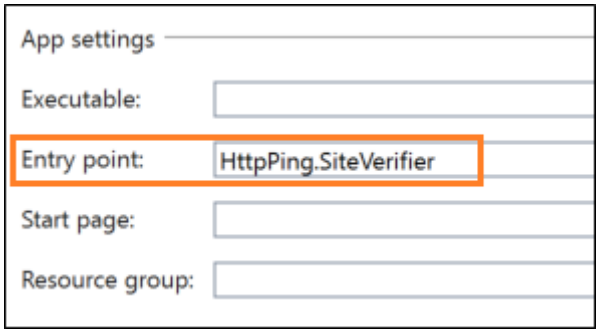
In the **Declarations** tab, add a **Background Tasks** declaration.



Then, choose the desired properties. Our sample uses the **Timer** property.



Provide the fully qualified name of the class in your Windows Runtime component that implements the background task.



Register the background task

Add code to your desktop application project that registers the background task.

C#

```
public void RegisterBackgroundTask(String triggerName)
{
    var current = BackgroundTaskRegistration.AllTasks
        .Where(b => b.Value.Name == triggerName).FirstOrDefault().Value;

    if (current is null)
    {
        BackgroundTaskBuilder builder = new BackgroundTaskBuilder();
        builder.Name = triggerName;
        builder.SetTrigger(new MaintenanceTrigger(15, false));
        builder.TaskEntryPoint = "HttpPing.SiteVerifier";
```

```
        builder.Register();
        System.Diagnostics.Debug.WriteLine("BGTask registered:" +
triggerName);
    }
    else
    {
        System.Diagnostics.Debug.WriteLine("Task already:" + triggerName);
    }
}
```

Find answers to your questions

Have questions? Ask us on Stack Overflow. Our team monitors these tags. You can also ask us here.

Distribute your packaged desktop app

Article • 10/05/2022

If you decide to package your desktop app in an MSIX package (see Building an MSIX package from your code), you can publish your packaged application to the Microsoft Store or sideload it on to one or more devices.

> **Note**
>
> Do you have a plan for how you might transition users to your packaged application? Before you distribute your app, see the **Transition users to your packaged app** section of this guide to get some ideas.

Distribute your application by publishing it to the Microsoft Store

The Microsoft Store is a convenient way for customers to get your app.

Publish your application to the Microsoft Store to reach the broadest audience. Also, organizational customers can acquire your application to distribute internally to their organizations through the Microsoft Store for Business.

If you plan to publish to the Microsoft Store, you'll be asked a few extra questions as part of the submission process. That's because your package manifest declares a restricted capability named **runFullTrust**, and we need to approve your application's use of that capability. You can read more about this requirement here: Restricted capabilities.

You don't have to sign your application before you submit it to the Store.

> **Important**
>
> If you plan to publish your application to the Microsoft Store, make sure that your application operates correctly on devices that run Windows 10 S or Windows 11 S. This is a Store requirement. See **Test your Windows app for Windows 10 S or Windows 11 S**.

Distribute your application without placing it onto the Microsoft Store

If you'd rather distribute your application without using the Store, you can manually distribute apps to one or more devices.

This might make sense if you want greater control over the distribution experience or you don't want to get involved with the Microsoft Store certification process.

To distribute your application to other devices without placing it in the Store, you have to obtain a certificate, sign your application by using that certificate, and then sideload your application onto those devices.

You can create a certificate or obtain one from a popular vendor such as Verisign.

If you plan to distribute your application onto devices that run Windows 10 S or Windows 11 S, your application has to be signed by the Microsoft Store so you'll have to go through the Store submission process before you can distribute your application onto those devices.

If you create a certificate, you have to install it into the **Trusted Root** or **Trusted People** certificate store on each device that runs your app. If you get a certificate from a popular vendor, you won't have to install anything onto other systems besides your app.

> **Important**
>
> Make sure that the publisher name on your certificate matches the publisher name of your app.

To sign your application by using a certificate, see Sign an application package using SignTool.

To sideload your application onto other devices, see Sideload LOB apps in Windows.

Transition users to your packaged app

Before you distribute your app, consider adding a few extensions to your package manifest to help users get into the habit of using your packaged app. Here's a few things you can do.

- Point existing Start tiles and taskbar buttons to your packaged app.
- Associate your packaged application with a set of file types.

- Make your packaged application open certain types of files by default.

For the complete list of extensions and the guidance for how to use them, see Transition users to your app.

Also, consider adding code to your packaged application that accomplishes these tasks:

- Migrates user data associated with your desktop application to the appropriate folder locations of your packaged app.
- Gives users the option to uninstall the desktop version of your app.

Let's talk about each one of these tasks. We'll start with user data migration.

Migrate user data

If you're going to add code that migrates user data, it's best to run that code only when the application is first started. Before you migrate the users data, display a dialog box to the user that explains what is happening, why it is recommended, and what's going to happen to their existing data.

Here's an example of how you could do this in a .NET-based packaged app.

C#

```
private void MigrateUserData()
{
    String sourceDir = Environment.GetFolderPath
        (Environment.SpecialFolder.ApplicationData) + "\\AppName";

    if (sourceDir != null)
    {
        DialogResult migrateResult = MessageBox.Show
            ("Would you like to migrate your data from the previous version
of this app?",
             "Data Migration", MessageBoxButtons.YesNo);

        if (migrateResult.Equals(DialogResult.Yes))
        {
            String destinationDir =
                Windows.Storage.ApplicationData.Current.LocalFolder.Path +
"\\AppName";

            Process process = new Process();
            process.StartInfo.FileName = "robocopy.exe";
            process.StartInfo.Arguments = "%LOCALAPPDATA%\\AppName " +
destinationDir + " /move";
            process.StartInfo.CreateNoWindow = true;
            process.Start();
            process.WaitForExit();
```

```
            if (process.ExitCode > 1)
            {
                //Migration was unsuccessful -- you can choose to
block/retry/other action
            }
        }
    }
}
```

Uninstall the desktop version of your app

It is better not to uninstall the users desktop application without first asking them for permission. Display a dialog box that asks the user for that permission. Users might decide not to uninstall the desktop version of your app. If that happens, you'll have to decide whether you want to block usage of the desktop application or support the side-by-side use of both apps.

Here's an example of how you could do this in a .NET-based packaged app.

To view the complete context of this snippet, see the **MainWindow.cs** file of this sample WPF picture viewer with transition/migration/uninstallation.

C#

```
private void RemoveDesktopApp()
{
    //Typically, you can find your uninstall string at this location.
    String uninstallString = (String)Microsoft.Win32.Registry.GetValue

(@"HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion"
+
         @"\Uninstall\{7AD02FB8-B85E-44BC-8998-F4803BA5A0E3}\",
"UninstallString", null);

    //Detect if the previous version of the Desktop application is
installed.
    if (uninstallString != null)
    {
        DialogResult uninstallResult = MessageBox.Show
            ("To have the best experience, consider uninstalling the "
              + " previous version of this app. Would you like to do that
now?",
              "Uninstall the previous version", MessageBoxButtons.YesNo);

        if (uninstallResult.Equals(DialogResult.Yes))
        {
                    string[] uninstallArgs = uninstallString.Split(' ');

            Process process = new Process();
```

```
            process.StartInfo.FileName = uninstallArgs[0];
            process.StartInfo.Arguments = uninstallArgs[1];
            process.StartInfo.CreateNoWindow = true;

            process.Start();
            process.WaitForExit();

            if (process.ExitCode != 0)
            {
                //Uninstallation was unsuccessful - You can choose to block
the application here.
            }
        }
    }

}
```

Next steps

Have questions? Ask us on our Tech Community.

If you encounter issues publishing your application to the Store, this blog post contains some useful tips.

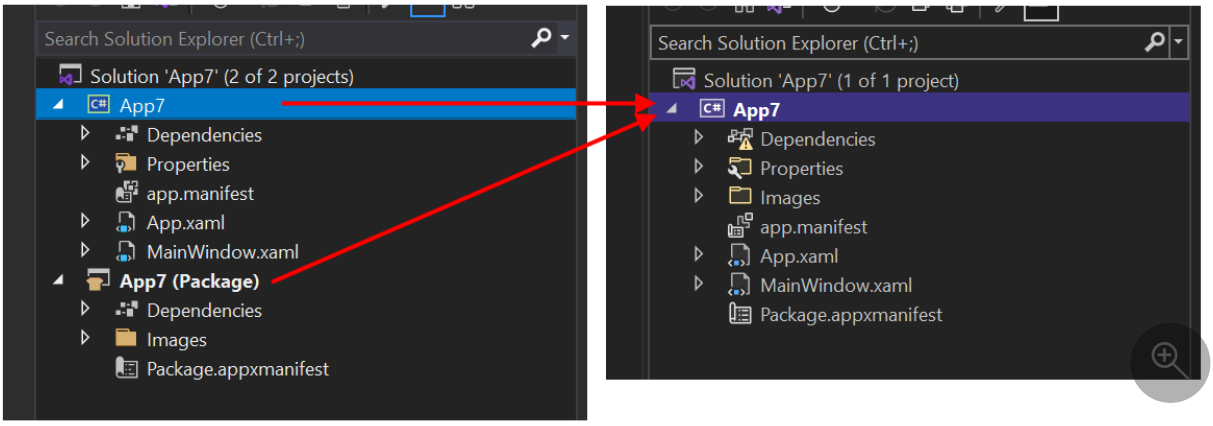
Package your app using single-project MSIX

Article • 02/03/2023

Single-project MSIX is a feature that lets you build a packaged WinUI 3 desktop app without the need for a separate packaging project. A WinUI 3 app is one that uses the WinUI 3 framework for its user interface (UI); and it uses the Windows App SDK. To package a desktop app that's *not* a WinUI 3 app, see Set up your desktop application for MSIX packaging in Visual Studio.

The single-project MSIX feature is available as a Visual Studio extension that you can use for these scenarios:

- Create a *new* WinUI 3 desktop app using the **Blank App, Packaged (WinUI 3 in Desktop)** Visual Studio project template that comes with the Windows App SDK. That project is configured to build your app into an MSIX package without needing a separate packaging project.
- Modify an *existing* WinUI 3 desktop app that uses a separate packaging project. The modification involves removing the separate packaging project, as shown in the illustration below.



Overview

This section introduces some important details about the single-project MSIX feature.

Benefits

Before the introduction of the single-project MSIX feature, if you wanted to build a packaged WinUI 3 desktop app, then you needed two projects in your solution—your app project, plus an additional **Windows Application Packaging Project** (see Set up

your desktop application for MSIX packaging in Visual Studio). The single-project MSIX feature enables you to develop and build your app using only a project for your app. That provides a cleaner project structure and a more straightforward development experience. For example, you no longer need to select the separate packaging project as your startup project.

Supported project types

The single-project MSIX feature supports WinUI 3 templates in Visual Studio (C# and C++) for the Windows App SDK.

Limitations

Single-project MSIX supports only a single executable in the generated MSIX package. If you need to combine multiple executables into a single MSIX package, then you'll need to continue using a Windows Application Packaging Project in your solution.

Install the single-project MSIX packaging tools

The single-project MSIX packaging tools include Visual Studio project templates that you can use to create new packaged WinUI 3 desktop apps. Those tools are included with the Windows App SDK extension for Visual Studio. For installation instructions for the Windows App SDK, see Install tools for the Windows App SDK.

Windows App SDK 0.8 and C# version of 1.0 Preview 3: The single-project MSIX packaging tools are *not* included with the Windows App SDK extension for Visual Studio for Windows App SDK version 0.8, or for C# projects with up to and including Preview 3 of the Windows App SDK 1.0. So if you're using those versions, then you *might* need to explicitly install the single-project MSIX packaging tools. See the info below:

- **Visual Studio 2022 version 17.1 and later:** The Single-project MSIX Packaging Tools for Visual Studio 2022 VSIX extension is built into Visual Studio 2022 version 17.1 and later.
- **Visual Studio 2022 prior to version 17.1:** Install the Single-project MSIX Packaging Tools for Visual Studio 2022 VSIX extension.
- **Visual Studio 2019:** Install the Single-project MSIX Packaging Tools for Visual Studio 2019 VSIX extension. The extension requires Visual Studio 2019 version 16.10.x or later.

You can then, in Visual Studio, confirm that you have the extension installed. Click **Extensions** > **Manage Extensions** > **Installed** > **All**, and confirm that *Single-project MSIX Packaging Tools* is listed.

Create a new project

If you're using Windows App SDK 1.0 Preview 2 or later, then you can create a new WinUI 3-based app that includes single-project MSIX support simply by using the **Blank App, Packaged (WinUI 3 in Desktop)** template. For more information, see Create your first WinUI 3 project.

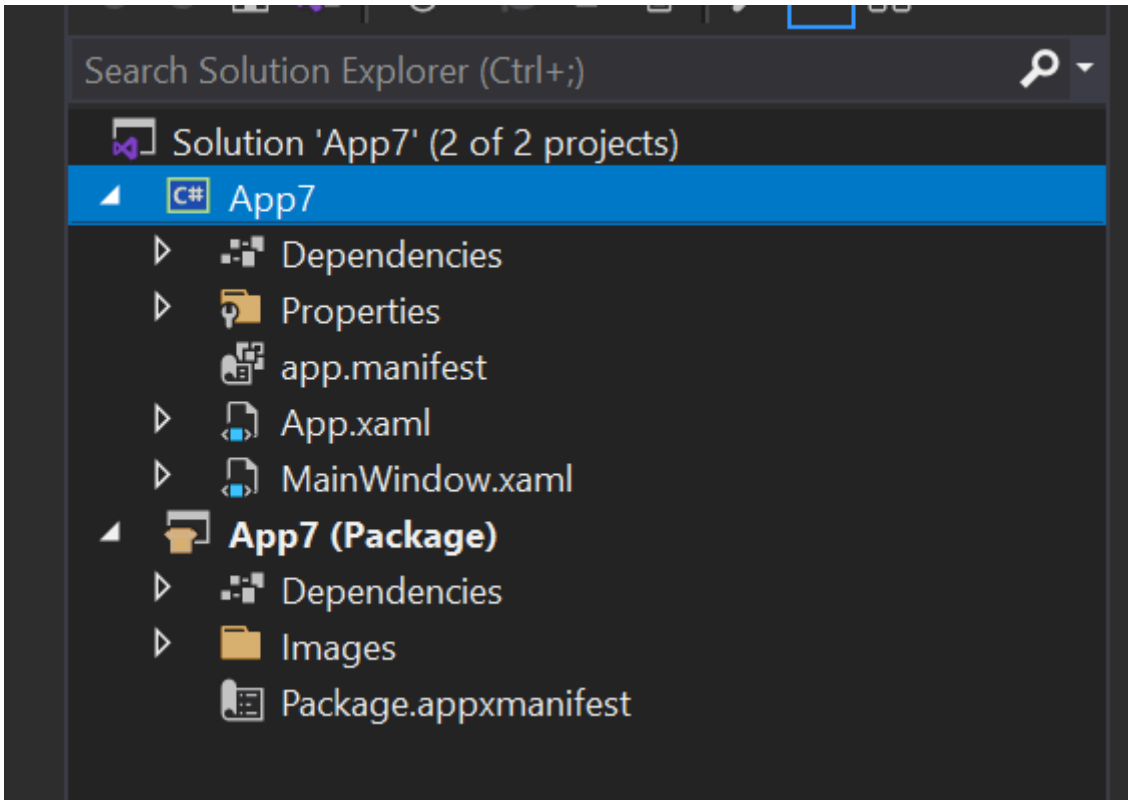
Modify an existing project

Follow the steps in this section to modify an existing WinUI 3-based app that uses a separate packaging project. The steps include moving the package manifest (and other support needed to build an MSIX package) into the application project, and then removing the separate packaging project.

Step 1: Create or open an existing packaging project

If you already have a solution for a WinUI 3 desktop app (see WinUI 3 templates in Visual Studio) that includes a **Windows Application Packaging Project** (see Set up your desktop application for MSIX packaging in Visual Studio), then open that solution in Visual Studio now.

If not, then create a new WinUI 3 desktop app in Visual Studio by using the **Blank App, Packaged with Windows Application Packaging Project (WinUI 3 in Desktop)** template. Your solution will look similar to the screenshot below.



Step 2: Edit the application project settings

Next, edit some configuration settings to use the single-project MSIX feature. There are different instructions depending on your project type and Visual Studio version.

C#

1. In **Solution Explorer**, double-click the project node for your application to open the **.csproj** file in the XML editor. Add the following XML to the main **<PropertyGroup>** element.

   XML

   ```
   <EnableMsixTooling>true</EnableMsixTooling>
   <PublishProfile>Properties\PublishProfiles\win10-$(Platform).pubxml
   </PublishProfile>
   ```

   When you're done, the **<PropertyGroup>** element should look similar to this.

   XML

   ```
   <Project Sdk="Microsoft.NET.Sdk">
     <PropertyGroup>
       <OutputType>WinExe</OutputType>
       <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
       ...
   ```

```
    <UseWinUI>true</UseWinUI>
    <EnableMsixTooling>true</EnableMsixTooling>

<PublishProfile>Properties\PublishProfiles\win10-$(Platform).pubxml
</PublishProfile>
  </PropertyGroup>
```

2. Save your changes, and close the **.csproj** file.

3. Perform one of the following procedures, depending on your version of Visual Studio.

   For Visual Studio 2022 or later:

   a. In **Solution Explorer**, right-click the **Properties** folder under the project node for your application, and select **Add** > **New Item...**.

   b. Select **Text File**, name the new file **launchSettings.json**, and click **Add**. Make sure the new file is in the **Properties** folder of your application project.

   c. Copy the following settings into the new file. You're free to change the values as needed for your scenario. The **MyApp** value can be any string; it doesn't need to match the name of your application.

   JSON

```
{
    "profiles": {
        "MyApp": {
            "commandName": "MsixPackage",
            "commandLineArgs": "", /* Command line arguments to
pass to the app. */
            "alwaysReinstallApp": false, /* Uninstall and then
reinstall the app. All information about the app state is
deleted. */
            "remoteDebugEnabled": false, /* Indicates that the
debugger should attach to a process on a remote machine. */
            "allowLocalNetworkLoopbackProperty": true, /* Allow
the app to make network calls to the device it is installed on.
*/
            "authenticationMode": "Windows", /* The
authentication scheme to use when connecting to the remote
machine. */
            "doNotLaunchApp": false, /* Do not launch the app,
but debug my code when it starts. */
            "remoteDebugMachine": "", /* The name of the remote
machine. */
            "nativeDebugging": false /* Enable debugging for
managed and native code together, also known as mixed-mode
debugging. */
        }
```

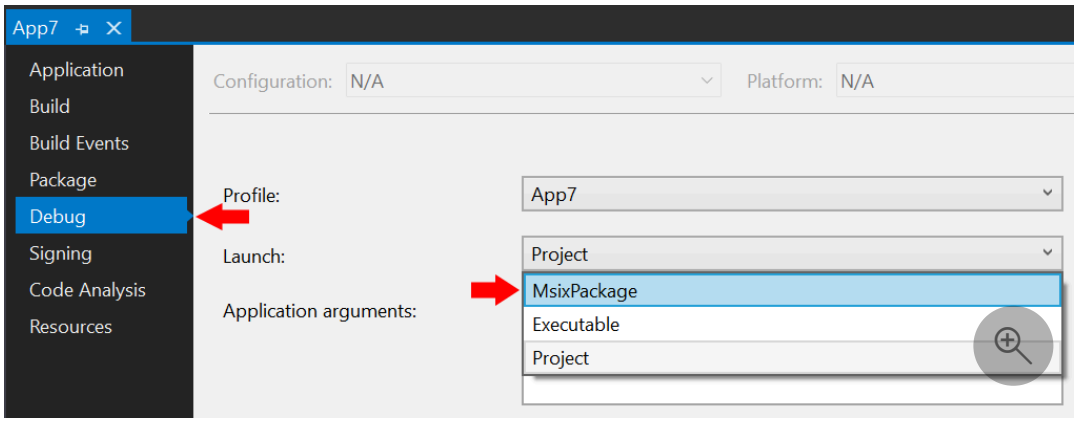
```
    }
}
```

d. Save and close the **launchSettings.json** file.

For Visual Studio 2019:

a. In **Solution Explorer**, right-click the project node for your application, and select **Properties**.

b. Select the **Debug** tab and set the **Launch** property to **MsixPackage**. You might have to select this twice if it reverts back on the first attempt.



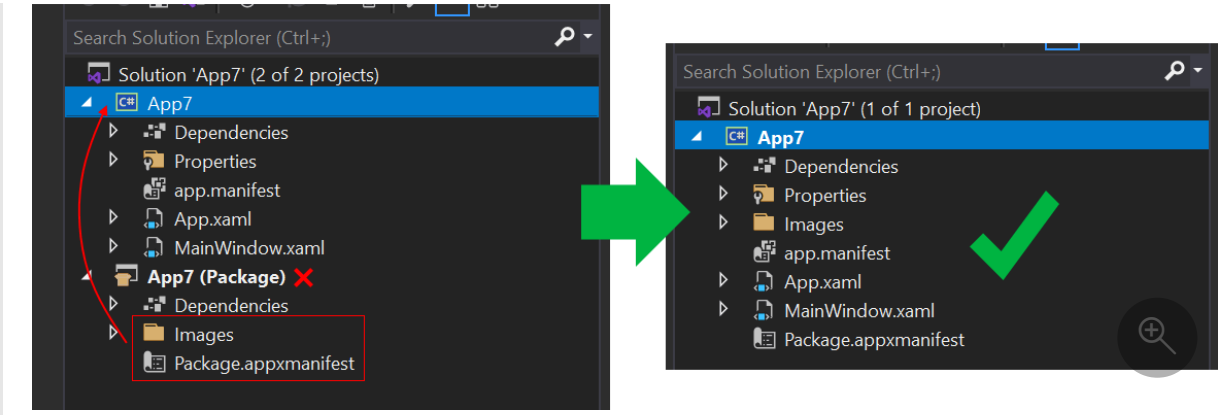
c. Save your changes.

Step 3: Move files to the application project

Next, move several important files to the application project. There are different instructions depending on your project type.

C#

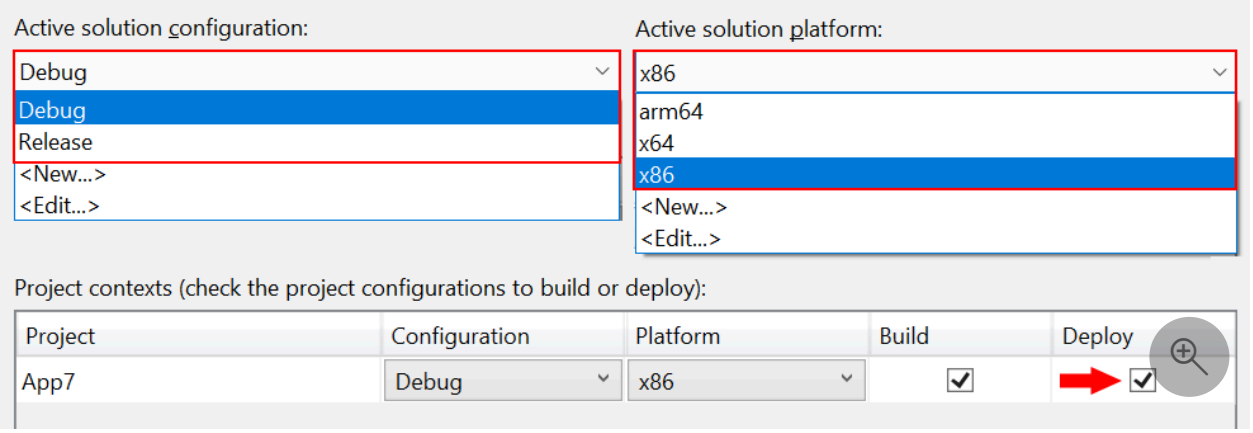
1. In **File Explorer**, move the **Package.appxmanifest** file and the **Images** folder from your packaging project to your application project. Place this file and folder in the top level of the application project's folder hierarchy.
2. In Visual Studio, in **Solution Explorer**, multi-select all of the images inside the **Images** folder. In the **Properties** window, set **Build Action** to *Content*.
3. Remove the packaging project from your solution.



Step 4: Enable deploying in Configuration Manager

1. Select **Build** > **Configuration Manager**.
2. In **Configuration Manager**, click the **Deploy** check box for every combination of configuration and platform (for example, **Debug** and **x86**, **Debug** and **arm64**, **Release** and **x64**, and more).

> ⓘ Note
>
> Be sure to use the **Active solution configuration** and **Active solution platform** drop-downs at the top instead of the **Configuration** and **Platform** drop-downs in the same row as the **Deploy** check box.



Step 5: Deploy your app

Build and deploy your application project. Visual Studio will build your application into an MSIX package, install the package, and then run your application.

Step 6: Package your app for publishing

Use the Package & Publish command in Visual Studio to package your application to publish it to the Microsoft Store.

Automate building and packaging your single-project MSIX app

You can use `msbuild` to build and package your single-project MSIX app, thereby allowing you to automate the workflow. The technique for a single-project solution, and the command-line, is only slightly different from what you might already be doing if you have a two-project solution—one that has a **Windows Application Packaging Project** (see Set up your desktop application for MSIX packaging in Visual Studio).

The important build command option for a single-project solution is `/p:GenerateAppxPackageOnBuild=true`. Without that option, the project will build, but you won't get an MSIX package. Include that option, and that will cause the MSIX package to be generated.

There's a full example in the form of a GitHub Action that builds a WinUI 3 single-project solution.

> **Note**
>
> Single-project MSIX doesn't currently support producing MSIX bundles (see Bundling MSIX packages). It produces only a single MSIX. But you can bundle `.msix` files into an MSIX bundle by using the MSIX Bundler GitHub Action.

How a packaged desktop app runs on Windows

For a deeper dive on what happens to files and registry entries when you create a Windows app package for your desktop application, see Understanding how packaged desktop apps run on Windows.

Provide feedback

To send us your feedback, report problems, or ask questions about the single-project MSIX feature, post a discussion or issue on the Windows App SDK GitHub repository.

Grant package identity by packaging with external location

Article • 05/22/2025

Many Windows features can be used by a desktop app only if that app has package identity at runtime. See Features that require package identity. If you have an existing desktop app, with its own installer, then there's very little you need to change in order to benefit from package identity.

Starting in Windows 10, version 2004, you can grant package identity to an app simply by building and registering a *package with external location* with your app. Packaging with external location allows you to register a simple identity package in your existing installer without changing how or where you install your application. You might be familiar with full MSIX packaging; this is a much lighter-weight option.

You can add an identity package to an existing Visual Studio project with the Windows Application Packaging Project and Package with External Location extension. This approach is recommended when there is a single application project that needs identity. The tooling provides a visual manifest editor, visual Resource Designer for localization, graphical wizard for creating and trusting self-signed certificates, automatic updating of application manifests, and PowerShell scripts to register and unregister the identity package for local testing.

If you don't build with Visual Studio or want to bundle multiple application executables under a shared identity, you can build an identity package manually.

Grant package identity by packaging with external location in Visual Studio

Article • 05/22/2025

For the motivations behind adding package identity, as well as the differences between building identity packages in Visual Studio and building them manually, see Overview.

This topic describes how to build and register an identity package by using Visual Studio. For info about building an identity package manually, see Grant package identity by packaging with external location manually.

These are the steps (which this topic describes in detail) to build and register an identity package by using Visual Studio:

1. Install Visual Studio components
2. Add a Packaging Project to your solution
3. Configure the Packaging Project for signing
4. Build and test the Packaging Project in Release Mode
5. Register the identity package in your installer
6. Optional steps

Install Visual Studio components

Creating an identity package in Visual Studio requires the **Windows Application Packaging Project** and the **Package with External Location** extension.

Install the **Windows Application Packaging Project** components as described in Required Visual Studio version and workload.

In Visual Studio, via the **Extensions** > **Manage Extensions** menu item, install the **Package with External Location** extension.

Add a Packaging Project to your solution

To add a Packaging Project to your solution with a Project Reference to your application project, see Set up the Windows Application Packaging Project in your solution.

Enable packaging with external location by right-clicking the Packaging Project in Solution Explorer, navigating to the **External Location** tab provided by the **Package with External Location** extension, enabling the **Package with External Location** option, and saving the changes.

Set the **Package name** and **Publisher display name** fields of the identity package by double-clicking `Package.appxmanifest` in the Packaging Project to open the visual manifest editor, navigating to the Packaging tab, and setting the **Package name** and **Publisher display name** fields to the desired values.

If you have a custom application manifest in your application project, then for info about synchronizing the values with the values from `Package.appxmanifest`, see Add identity metadata to your desktop application manifests. The **Package with External Location** extension uses **App** for the **applicationId**.

If you don't have a custom application manifest, then Visual Studio will produce the appropriate artifacts during the build process. .NET projects embed a manifest by default, which conflicts with the produced manifest artifacts. To resolve that, right-click the Project, open **Properties**, and in the **Application** tab under the **Manifest** section, change **Embed manifest with default settings** to **Create application without a manifest**.

Configure the Packaging Project for signing

Generate a certificate for signing by walking through the **Publish** > **Create App Packages** wizard shown in Create an app package using the packaging wizard.

On the first screen, ensure that **Sideloading** is selected, and **Enable automatic updates** is unchecked. On the second screen, create a self-signed certificate if necessary, then click the **Trust** button to trust it in the Local Machine Trusted People certificate store. On the final screen, set **Generate app bundle** to *Never*, and click **Create** to complete the signing configuration.

Build and test the Packaging Project in Release Mode

To avoid complications from Debug mode dependencies, set the Build configuration to Release mode, and build the Packaging Project.

Building the Packaging Project produces a **PackageWithExternalLocation** folder in the build output. That folder contains the `MSIX` file representing the identity package, as well as `Install` and `Remove` PowerShell scripts to register and unregister the generated identity package locally for testing.

The `Install` PowerShell script registers the generated identity package locally, and connects it with the `ExternalLocation` sibling folder for testing purposes. To test the application with identity, run the application executable from the `ExternalLocation` folder.

To associate identity with your application in production, you'll need to ship the generated identity package with your application, and register it in your installer.

Register the identity package in your installer

The last step to associate identity with your application is to register the identity package in your installer, and associate it with your application's installation directory.

The code listing below demonstrates registering the identity package by using the PackageManager.AddPackageByUriAsync method.

C#

```
using Windows.Management.Deployment;

...

var externalUri = new Uri(externalLocation);
var packageUri = new Uri(packagePath);

var packageManager = new PackageManager();

var options = new AddPackageOptions();
options.ExternalLocationUri = externalUri;

await packageManager.AddPackageByUriAsync(packageUri, options);
```

Be aware of the following important details about this code:

- Set `externalLocation` to the absolute path of your application's installation directory (without any executable names).
- Set `packagePath` to the absolute path of the identity package produced in the previous step (with the file name).

For production-ready code in C# and C++, see Sample apps below. The samples also demonstrate how to unregister the identity package on uninstall.

Sample apps

For fully functional C# and C++ apps that demonstrate how to register an identity package, see the PackageWithExternalLocation samples.

Optional steps

Localization

Some features that understand package identity might result in strings from your identity package manifest being displayed in the Windows OS. For example:

- An application that uses camera, microphone, or location APIs will have a dedicated control toggle in Windows Privacy Settings along with a brokered consent prompt that users can use to grant or deny access to those sensitive resources.
- An application that registers a share target will show up in the share dialog.

To localize the strings in the identity package manifest, see Localize the manifest.

Grant package identity by packaging with external location

Article • 04/07/2025

Many Windows features can be used by a desktop app only if that app has package identity at runtime. See Features that require package identity. If you have an existing desktop app, with its own installer, there's very little you need to change in order to benefit from package identity.

Starting in Windows 10, version 2004, you can grant package identity to an app simply by building and registering a *package with external location* with your app. Packaging with external location allows you to register a simple identity package in your existing installer without changing how or where you install your application. If you're familiar with full MSIX packaging, this is a much lighter-weight option as described below.

To build and register an identity package, follow these steps:

1. Create a package manifest for the identity package
2. Build and sign the identity package
3. Add identity metadata to your desktop application manifest
4. Register the identity package in your installer

Create a package manifest for the identity package

The first step to creating an identity package is to create a package manifest based on the below template. This is an MSIX manifest but is only used for identity and doesn't alter the app's runtime behavior.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<Package IgnorableNamespaces="uap uap10"
  xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"
  xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
  xmlns:uap10="http://schemas.microsoft.com/appx/manifest/uap/windows10/10"

xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation/windows1
0/restrictedcapabilities">
  <Identity Name="ContosoPhotoStore" Publisher="CN=Contoso"
Version="1.0.0.0" ProcessorArchitecture="neutral" />
  <Properties>
    <DisplayName>Contoso PhotoStore</DisplayName>
```

```
    <PublisherDisplayName>Contoso</PublisherDisplayName>
    <Logo>Assets\storelogo.png</Logo>
    <uap10:AllowExternalContent>true</uap10:AllowExternalContent>
  </Properties>
  <Resources>
    <Resource Language="en-us" />
  </Resources>
  <Dependencies>
    <TargetDeviceFamily Name="Windows.Desktop" MinVersion="10.0.19041.0"
MaxVersionTested="10.0.26100.0" />
  </Dependencies>
  <Capabilities>
    <rescap:Capability Name="runFullTrust" />
    <rescap:Capability Name="unvirtualizedResources"/>
  </Capabilities>
  <Applications>
    <Application Id="ContosoPhotoStore" Executable="ContosoPhotoStore.exe"
uap10:TrustLevel="mediumIL" uap10:RuntimeBehavior="win32App">
      <uap:VisualElements AppListEntry="none" DisplayName="Contoso
PhotoStore" Description="Contoso PhotoStore App"
BackgroundColor="transparent"
Square150x150Logo="Assets\Square150x150Logo.png"
Square44x44Logo="Assets\Square44x44Logo.png" />
    </Application>
  </Applications>
</Package>
```

Note the below important details about this manifest:

- Fill in the `Identity` element attributes with the details of your application
  - `Publisher` must match the `Subject` of the certificate used to sign the application
- Fill in the `DisplayName` and `PublisherDisplayName` elements with the details of your application
  - Unless you add additional features to the manifest beyond simple identity, these values are not displayed anywhere
- Update the `Logo` element to a relative path within your application's installation directory that will resolve to a .png, .jpg, or .jpeg image
- Ensure the `AllowExternalContent` element is set to `true` as shown which enables reusing your existing installer
- Set `TargetDeviceFamily` `MinVersion` and `MaxVersionTested` per below:
  - Set `MinVersion` to `10.0.19041.0` as shown for maximum reach and uniformity across Windows 10 and Windows 11 OS versions
  - Set `MinVersion` to `10.0.26100.0` to restrict the identity package to Windows 11, version 24H2 and above
  - Set `MaxVersionTested` to `10.0.26100.0` as shown

- Ensure the `runFullTrust` and `unvirtualizedResources` capabilities are declared as shown for Win32 compatibility
- Add an `Application` element as shown for each binary associated with your application
  - Ensure `TrustLevel` is `mediumIL` and `RuntimeBehavior` is `win32App` as shown for Win32 compatibility
- The `VisualElements` child element is required, but the `AppListEntry="none"` attribute ensures the identity package isn't shown among installed apps
  - Update the `DisplayName` and `Description` attributes with relevant details and leave the other attributes as shown (the referenced image paths do not need to resolve)

The identity package created from this manifest will be connected to your application's installation directory when you register the package in a later step.

Build and sign the identity package

After you create your identity package manifest, build the identity package using the MakeAppx.exe tool in the Windows SDK.

Console

```
MakeAppx.exe pack /o /d <path to directory that contains manifest> /nv /p
<output path>\MyPackage.msix
```

Note: The `/nv` flag is required to bypass validation of referenced file paths in the manifest.

In order to be installed on end user computers, the identity package must be signed with a certificate that is trusted on the target computer. You can create a new self-signed certificate for development purposes and sign your identity package using SignTool, which is available in the Windows SDK, but a production certificate from an IT Department or a service like Azure Trusted Signing will be required to register the package on end user computers.

Console

```
SignTool.exe sign /fd SHA256 /a /f <path to certificate>\MyCertificate.pfx
/p <certificate password> <path to package with external
location>\MyPackage.msix
```

Note: For how to build and sign the identity package within a CI/CD pipeline with production certificates, see the MSIX and CI/CD Pipeline Overview for examples.

Add identity metadata to your desktop application manifest

You connect the identity package with your application executables by including application manifests (a.k.a side-by-side or fusion manifests) with metadata that matches metadata from the identity package manifest.

Below is an example application manifest snippet demonstrating the `msix` element required to connect your binaries with metadata from your identity package.

XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-com:asm.v1">
  <assemblyIdentity version="0.0.0.0" name="ContosoPhotoStore"/>
  <msix xmlns="urn:schemas-microsoft-com:msix.v1"
          publisher="CN=Contoso"
          packageName="ContosoPhotoStore"
          applicationId="ContosoPhotoStore"
        />
</assembly>
```

The attributes of the `msix` element must match these values from the identity package manifest:

- The `packageName` and `publisher` attributes must match the `Name` and `Publisher` attributes in the Identity element in your identity package manifest, respectively
- The `applicationId` attribute must match the `Id` attribute of the corresponding Application element in your identity package manifest

Register the identity package in your installer

The last step to associate identity with your application is to register the identity package in your installer and associate it with your application's installation directory.

The below snippet demonstrates using the PackageManager.AddPackageByUriAsync method to register the identity package.

C#

```
using Windows.Management.Deployment;

...

var externalUri = new Uri(externalLocation);
var packageUri = new Uri(packagePath);

var packageManager = new PackageManager();

var options = new AddPackageOptions();
options.ExternalLocationUri = externalUri;

await packageManager.AddPackageByUriAsync(packageUri, options);
```

Note the below important details about this code:

- Set `externalLocation` to the absolute path of your application's installation directory (without any executable names)
- Set `packagePath` to the absolute path of the identity package produced in the previous step (with the file name)

For a complete example including unregistering the package on uninstall, see StartUp.cs.

Sample app

See the SparsePackages sample for a fully functional sample app that demonstrates how to grant package identity to a desktop app by registering an identity package.

Deployment overview

Article • 10/14/2023

The topics in this section introduce options and guidance around deploying different types of Windows apps.

Advantages and disadvantages of packaging your app

Your first decision will be whether or not to package your app.

- **Packaged app**. Packaged apps are the only kind that have *package identity* at runtime. Package identity is needed for many Windows extensibility features—including background tasks, notifications, live tiles, custom context menu extensions, and share targets. That's because the operating system (OS) needs to be able to identify the caller of the corresponding API. See Features that require package identity.
  - Commonly, a packaged app's process runs inside a lightweight app container, and is isolated using file system and registry virtualization (see AppContainer for legacy apps and MSIX AppContainer apps). But you can configure a packaged app to *not* run in an app container.
  - A packaged app is packaged by using MSIX technology (see What is MSIX?).
  - **Packaged app with external location**. But because some existing apps aren't yet ready for all of their content to be present inside an MSIX package, there's an option for your app to be *packaged with external location*. That enables your app to have package identity; thereby being able to use those features that require it. For more info, see Grant package identity by packaging with external location.
  - A packaged app is *installed* by using MSIX, also. But if you choose to *package with external location*, then you can think of that as a "bring-your-own-installer" model. So there *will* be some installer work for you to do with that option. It's essentially a hybrid option between a packaged and an unpackaged app.
- **Unpackaged app**. You can opt out of using MSIX altogether by creating an unpackaged app. But be aware that an unpackaged app *doesn't* have package identity at runtime; so it misses out on certain Windows features (see Features that require package identity).

Each type of app can be published to the Microsoft Store, and installed that way or via Windows App Installer.

> **Important**
>
> We recommend that you package your app, and configure it to run in an app container. It'll be a seamless, modern, and reliable installation and update experience for your customers; and it'll be secure at runtime.

Expand table

	Packaged (and optionally running in an app container)	**Packaged with external location or unpackaged**
Advantages	Gives your users an easy way to install, uninstall, and update your app. Uninstall is clean—when your app is uninstalled, the system is restored to the same state it was in before installation—no artifacts are left behind. This kind of app also supports incremental and automatic updates. And the Microsoft Store optimizes for apps of this kind (although they can be used in or out of the Store). You get the benefits of having package identity.	With these options, your app is unrestricted in terms of the kind of app it is, the APIs it can call, and its access to the Registry and file system. Packaging with external location means that you get the benefits of having package identity.
Disadvantages	Your app is limited in terms of the kind of app it can be, and the agency it can have within the system. For example, an NT Service isn't possible. Inter-process communication (IPC) options are limited; privileged/elevated access is restricted if you're publishing to the Microsoft Store; file/Registry access is virtualized (but also see Flexible virtualization). And in some situations enterprise policies can disable updates by disabling the Microsoft Store.	With these options, an app that is at risk of causing stale configuration data and software to accumulate after the app has been uninstalled. That can be an issue for the customer and for the system. Your app will typically be installed and updated using `.exe` or `.msi` files, or via other installation and update solutions; using a custom installer, ClickOnce, or xcopy deployment. An unpackaged app lacks the benefits of having package identity.

For more info about package install location, working directory, and file and registry virtualization, see Understanding how packaged desktop apps run on Windows.

Use the Windows App SDK

After deciding whether or not to package your app, you can next decide whether or not to use the Windows App SDK in your app. See Windows App SDK deployment overview.

Win32 and .NET desktop apps

If you build a Win32 desktop app (sometimes called a *classic desktop app*) or a .NET app —including Windows Presentation Foundation (WPF) and Windows Forms (WinForms)— then you can package and deploy your app using MSIX.

- Create an MSIX package from an existing installer
- Build an MSIX package from source code
- Manage your MSIX deployment

You can also package and deploy these types of apps using other installation technologies.

- Application installation and servicing
- Windows Installer
- .NET application publishing overview
- Deploying the .NET Framework and applications
- Deploying a WPF application
- ClickOnce Deployment for Windows Forms

UWP apps

UWP apps are packaged and deployed using MSIX.

- Overview of packaging UWP apps
- Package a UWP app in Visual Studio
- Manage your MSIX deployment

Related topics

- Get started with WinUI
- Get started developing apps

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Windows App SDK deployment overview

Article • 11/02/2023

There are two ways in which you can deploy the Windows App SDK:

- **Framework-dependent**. Your app depends on the Windows App SDK runtime and/or Framework package being present on the target machine. Framework-dependent deployment is the default deployment mode of the Windows App SDK for its efficient use of machine resources and serviceability.
- **Self-contained**. Your app carries the Windows App SDK dependencies with it. Self-contained deployment is a deployment option that was introduced in Windows App SDK 1.1 Preview 1.

This topic also uses the terms *packaged app*, *packaged app with external location*, and *unpackaged app*. For explanations of those terms, see the Deployment overview.

Expand table

	Deploy framework-dependent	Deploy self-contained
Advantages	*Small deployment*. Only your app and its other dependencies are distributed. The Windows App SDK runtime and Framework package are installed automatically by framework-dependent apps that are packaged; or as part of the Windows App SDK runtime installer by framework-dependent apps that are either packaged with external location or unpackaged. *Serviceable*. Servicing updates to the Windows App SDK are installed automatically via the Windows App SDK Framework package without any action required of the app.	*Control Windows App SDK version*. You control which version of the Windows App SDK is deployed with your app. Servicing updates of the Windows App SDK won't impact your app unless you rebuild and redistribute it. *Isolated from other apps*. Apps and users can't uninstall your Windows App SDK dependency without uninstalling your entire app. *Xcopy deployment*. Since the Windows App SDK dependencies are carried by your app, you can deploy your app by simply xcopy-ing your build output, without any additional installation requirements.

	Deploy framework-dependent	Deploy self-contained
Disadvantages	*Additional installation dependencies*. Requires installation of the Windows App SDK runtime and/or Framework package, which can add complexity to app installation. *Shared dependencies*. Risk that shared dependencies are uninstalled. Apps or users uninstalling the shared components can impact the user experience of other apps that share the dependency. *Compatibility risk*. Risk that servicing updates to the Windows App SDK introduce breaking changes. While servicing updates should provide backward compatibility, it's possible that regressions are introduced.	*Larger deployments (unpackaged apps only)*. Because your app includes the Windows App SDK, the download size and hard drive space required are greater than would be the case for a framework-dependent version. *Performance (unpackaged apps only)*. Slower to load, and uses more memory since code pages aren't shared with other apps. *Not serviceable*. The Windows App SDK version distributed with your app can be updated only by releasing a new version of your app. You're responsible for integrating servicing updates of the Windows App SDK into your app.

Also see Create your first WinUI 3 project, and Use the Windows App SDK in an existing project.

More info about framework-dependent deployment

Before configuring your framework-dependent app for deployment, to learn more about the dependencies your app takes when it uses the Windows App SDK, review Deployment architecture for the Windows App SDK.

Packaged apps

If you've chosen to go with a framework-dependent packaged app (see Deployment overview), then here are instructions on how to deploy the Windows App SDK runtime with the app:

- Windows App SDK deployment guide for framework-dependent packaged apps
- Manage your MSIX deployment

Packaged with external location or unpackaged apps

If you've chosen to go with a framework-dependent packaged app with external location, or a framework-dependent unpackaged app (see Deployment overview), then here are instructions on how to deploy the Windows App SDK runtime with the app:

- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK

More info about self-contained deployment

See Windows App SDK deployment guide for self-contained apps.

Initialize the Windows App SDK

The way that you should initialize the Windows App SDK depends on whether, and how, you package your app; and on the way in which you deploy relative to the Windows App SDK runtime. Use the section below that applies to your app.

Packaged apps

Expand table

How your app deploys	How to initialize
Framework-dependent	See Call the Deployment API.
Self-contained	No initialization necessary.

Unpackaged apps, and apps packaged with external location

Expand table

How your app deploys	How to initialize
Framework-dependent	See Use the bootstrapper API in an app packaged with external location or unpackaged.
Self-contained	See Opting out of (or into) automatic UndockedRegFreeWinRT support.

Related topics

- Deployment overview
- Deployment architecture for the Windows App SDK
- Windows App SDK deployment guide for framework-dependent packaged apps
- Manage your MSIX deployment
- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK
- Windows App SDK deployment guide for self-contained apps
- Create your first WinUI 3 project
- Use the Windows App SDK in an existing project

Project properties

Article • 05/31/2024

This topic describes the project properties that you can set in your Visual Studio project file to customize how your app is deployed.

Expand table

Property name and description	Values	For more info
AppxPackage. Specifies whether or not a WinUI 3 app is packaged.	*false* (for an unpackaged app), or absent (for a packaged app)	Create a new project for an unpackaged WinUI 3 desktop app
EnableMsixTooling. Enables the single-project MSIX feature for a project.	*true* (to enable), or absent (to disable)	Package your app using single-project MSIX
UseWinUI. Specifies whether you're using the WinUI 3 user interface framework in your app.	*true*, or absent (for *false*)	WinUI in the Windows App SDK (WinUI 3)
WindowsAppSdkBootstrapInitialize. Determines whether or not the Windows App SDK leverages module initializers to call the bootstrapper API automatically at app startup.	*true* (the default for executables), *false* (the default for non-executables)	Opting out of (or into) automatic module initialization
WindowsAppSdkSelfContained. Determines whether or not an app is deployed *self-contained*.	*true*, or absent (for *false*)	Windows App SDK deployment guide for self-contained apps
WindowsAppSdkUndockedRegFreeWinRTInitialize. Determines whether or not the Windows App SDK's implementation of undocked registration-free Windows Runtime (*UndockedRegFreeWinRT*) is enabled automatically at app startup.	*true* (the default for executables), *false* (the default for non-executables)	Opting out of (or into) automatic UndockedRegFreeWinRT support
WindowsPackageType. Setting `<WindowsPackageType>None</WindowsPackageType>` for	*None*, or absent (to	Create a new project for an unpackaged WinUI 3

Property name and description	Values	For more info
an unpackaged app causes the *auto-initializer* to locate and load a version of the Windows App SDK version that's most appropriate for your app.	disable the auto-initializer)	desktop app Behind the scenes, and opting out of automatic module initialization

Example

Here's an excerpt from a typical `.csproj` file for a C# WinUI 3 project showing some of the project properties from the table above in use.

XML

```
...
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
    <TargetPlatformMinVersion>10.0.17763.0</TargetPlatformMinVersion>
    ...
    <UseWinUI>true</UseWinUI>
    <EnableMsixTooling>true</EnableMsixTooling>
  </PropertyGroup>
...
```

Related topics

- Deployment overview

Deployment architecture and overview for framework-dependent apps

Article • 04/19/2023

This article explains a high-level architecture of Windows App SDK deployment. The concepts below apply primarily to Windows App SDK framework-dependent apps. A framework-dependent app depends on the Windows App SDK runtime being present on the target machine.

There are two main options to distribute a framework-dependent app:

Expand table

App deployment method	Requirements
Packaged	- Must declare dependency on Framework package in the package manifest. - Deployment API is required for Microsoft Store distributed apps and recommended for non-Store distributed apps to ensure runtime dependencies are installed.
Packaged with external location or unpackaged	- Must distribute runtime either using the Installer or by installing required MSIX packages directly. - Additional runtime requirements: Must initialize access to the Windows App SDK runtime via the Bootstrap API.

For more details on these requirements, see the following articles:

- Windows App SDK deployment guide for framework-dependent packaged apps
- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged

Key terms

The following sections define key terms for Windows App SDK deployment and additional details on some of these packages.

Expand table

Term	Definition
Windows App SDK runtime	The MSIX packages required by an app to use the Windows App SDK. These packages include: Framework, Main, Singleton, and DDLM.

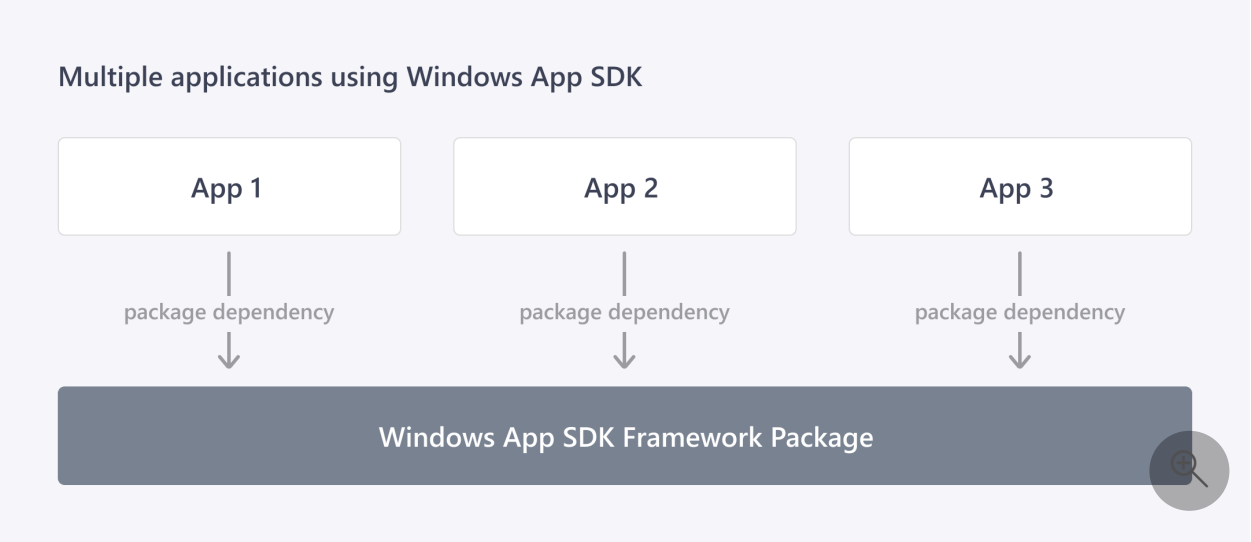
Term	Definition
	Depending on the features used and your app deployment method, you will need a certain set of these packages on the target machine.
Framework package	Contains binaries used at run time by apps (most Windows App SDK features). The framework includes a bootstrapper component that enables apps to automatically install the latest version of the Windows App SDK, which will be updated on a regular release cadence.
Main package	Package that contains background tasks to keep track of dynamic dependencies, and enables automatic updates to the Framework package from the Microsoft Store.
Singleton package	Contains background tasks, services, app extensions, and other components not included in the Framework package such as Push Notifications. This is generally a single long-running process that is brokered between apps.
Dynamic Dependency Lifetime Manager (DDLM) package	Prevents the OS from performing servicing updates to the MSIX packages while a packaged with external location or unpackaged app is in use.
Bootstrapper	An app-local binary used by packaged with external location and unpackaged apps to locate and load the best Windows App SDK version match as needed by the app.
Provisioning	The process of installing and registering packages (including files and registry keys) system-wide to eliminate the need for repeated installation by the other users. It can be done either as part of the OS or done during installation of an app.
Installer	Refers to the .exe installer which deploys the Framework, Main, Singleton, and DDLM packages.
MSIX	Modern installer technology that enables users to safely install an app per user, directly from the Microsoft Store or a web site. On Enterprise or shared PCs, apps can be installed for all users via PowerShell and MDM.

Framework package

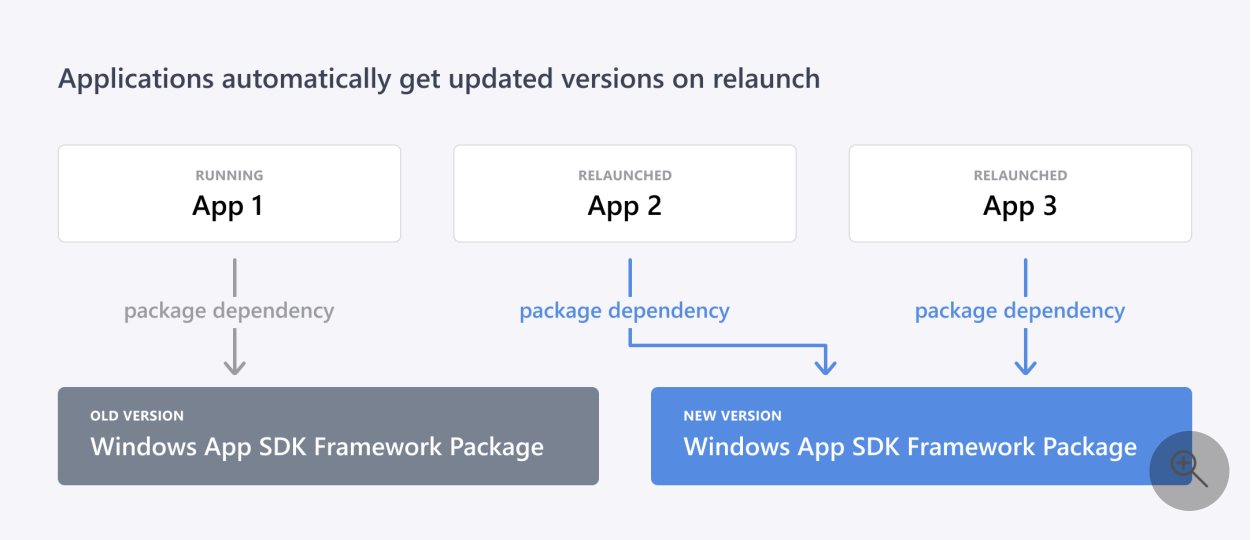
When you build an app that uses the Windows App SDK, your app references a set of Windows App SDK runtime components that are distributed to end users via a *framework package*. The framework package allows apps to access Windows App SDK components through a single shared source on the user's device, instead of bundling them into the app package. The framework package also carries its own resources, such as DLLs and API definitions (COM and Windows Runtime registrations). These resources

run in the context of your app, so they inherit the capabilities and privileges of your app, and don't assert any capabilities or privileges of their own. For more information about framework package dependencies, see MSIX framework packages and dynamic dependencies.

The Windows App SDK framework package is an MSIX package that is deployed to end users through the Microsoft Store. It can be easily and quickly updated with servicing releases, which may include security and reliability fixes. All framework-dependent apps that use the Windows App SDK have a dependency on a shared instance of the framework package, as illustrated in the following diagram.



When a new version of the Windows App SDK framework package is serviced, all framework-dependent apps are updated to the new version without themselves having to redistribute a copy. Windows updates to the newest version of frameworks as they are released, and apps will automatically reference the latest framework package version during relaunch. Older framework package versions will not be removed from the system until they are no longer running or being actively used by apps on the system.



Because app compatibility is important to Microsoft and to apps that depend on the Windows App SDK, the Windows App SDK framework package follows Semantic Versioning 2.0.0 rules. This means that after we release version 1.0 of the Windows App SDK, the Windows App SDK framework package will guarantee compatibility between minor and patch version changes, and breaking changes will occur only between major version updates.

Singleton package

The **singleton package** ensures that a single long-running process can handle services that are used across multiple apps, which may be running on different versions of the Windows App SDK.

The Windows App SDK singleton is needed to enable push notifications for unpackaged apps and packaged Win32 applications using Windows versions below 20H1, which cannot be supported by the existing UWP PushNotificationTrigger and ToastNotificationActionTrigger class. Future Windows App SDK features that cannot be supported by the Framework package will be added to the Singleton package.

Additional requirements for unpackaged apps

Bootstrapper

The bootstrapper is a library that must be included with your packaged with external location or unpackaged app. It provides the bootstrapper API (see Use the Windows App SDK runtime for apps packaged with external location or unpackaged), which enables unpackaged apps to perform these important tasks:

- Initialize the Dynamic Dependency Lifetime Manager (DDLM) for the Windows App SDK framework package.
- Find and load the Windows App SDK framework package to the app's package graph.

To accomplish these tasks, the nuget package leverages module initializers to wire up the bootstrapper for you. Simply set `<WindowsPackageType>None</WindowsPackageType>` in your project file. In advanced scenarios, if you want control over the initialization, you can call the bootstrapper API directly in your app's startup code (see Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK) so that it can properly initialize the system for the unpackaged app. Your app must use the bootstrapper API before it can use Windows App SDK features such as WinUI, App lifecycle, MRT Core, and DWriteCore.

The bootstrapper library in the Windows App SDK 1.0 release includes:

- **Microsoft.WindowsAppRuntime.Bootstrap.dll** (C++ and C#)
- **Microsoft.WindowsAppRuntime.Bootstrap.Net.dll** (C# wrapper)

Dynamic Dependency Lifetime Manager (DDLM)

The purpose of the DDLM is to prevent servicing of the Windows App SDK framework package while it is in use by an unpackaged app. It contains a server that must be initialized by the bootstrapper early in an app's startup to provide that functionality.

There is one DDLM for each version and architecture of the Windows App SDK framework package. This means on an `x64` computer, you may have both an `x86` and an `x64` version of the DDLM to support apps of both architectures.

Additional requirements

- For packaged apps, the VCLibs framework package dependency is a requirement. For more info, see C++ Runtime framework packages for Desktop Bridge.
- For unpackaged apps, the Visual C++ Redistributable is a requirement. For more info, see Microsoft Visual C++ Redistributable latest supported downloads.
- **C#**. For the .NET runtime, see Download .NET.

Related topics

- Windows App SDK deployment guide for framework-dependent packaged apps
- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged

Windows App SDK deployment guide for framework-dependent packaged apps

Article • 08/19/2024

This article provides guidance about deploying framework-dependent packaged apps (see What is MSIX?) that use the Windows App SDK. The equivalent topic for other framework-dependent packaging options is Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged.

Overview

By default, when you create a project using one of the WinUI 3 templates in Visual Studio, your project is configured to build the app into an MSIX package using either single-project MSIX (see Package your app using single-project MSIX) or a Windows Application Packaging project (see Set up your desktop application for MSIX packaging in Visual Studio). You can then build an MSIX package for your app by using the instructions in Package a desktop or UWP app in Visual Studio. After you build an MSIX package for your app, you have several options to Manage your MSIX deployment.

To learn more about the packages that your packaged app might need when it uses the Windows App SDK, see Deployment architecture for the Windows App SDK. Those include the *Framework*, *Main*, and *Singleton* packages; which are all signed and published by Microsoft. There are two main requirements for deploying a packaged app:

1. Deploy the Windows App SDK framework package.
2. Call the Deployment API.

Prerequisites

- For packaged apps, the VCLibs framework package dependency is a requirement. For more info, see C++ Runtime framework packages for Desktop Bridge.
- **C#**. .NET 6 or later is required. For more info, see .NET Downloads.

Deploy the Windows App SDK framework package

The Windows App SDK framework package contains the Windows App SDK binaries used at run time, and it is installed with your application. The framework has different deployment requirements for different channels of the Windows App SDK.

Stable version

When you install a stable release version (see Stable channel release notes) of the Windows App SDK NuGet package on your development computer, and you create a project using one of the provided WinUI 3 project templates, the generated package manifest contains a PackageDependency element that specifies a dependency on the framework package.

However, if you build your app package manually using a separate Windows Application Packaging Project, then you must declare a **PackageReference** in your `Application (package).wapproj` file, like the following:

```xml
<ItemGroup>
   <PackageReference Include="Microsoft.WindowsAppSDK" Version="1.0.1">
       <IncludeAssets>build</IncludeAssets>
   </PackageReference>
</ItemGroup>
```

That package dependency ensures that the Framework package is installed when your app is deployed to another computer.

Preview version

When you install a preview release version (see Preview channel release notes) of the Windows App SDK NuGet package on your development computer, a preview version of the Windows App SDK framework package is deployed during build time as a NuGet package dependency.

Call the Deployment API

Also see Initialize the Windows App SDK.

The Deployment API is provided by the Windows App SDK framework package, and is available in the Microsoft.Windows.ApplicationModel.WindowsAppRuntime namespace. The Windows application model doesn't support declaring a dependency on the Main and Singleton packages. The Deployment API is therefore required for these reasons:

1. To deploy the Singleton package for features not in the Framework package (for example, push notifications).
2. To deploy the Main package, which enables automatic updates to the Framework package from the Microsoft Store.

For packaged apps that are *not* distributed through the Store, you as the developer are responsible for distributing the Framework package. We recommended that you call the Deployment API so that any critical servicing updates are delivered. Note that for using features outside the Framework package (for example, push notifications), the Singleton package must be deployed (this can be done with the Deployment API, or by redistributing the MSIX packages using your own install method).

> **ⓘ Important**
>
> In Windows App SDK version 1.0, only packaged apps that are full trust or that have the **packageManagement** restricted capability have the permission to use the Deployment API to install the Main and Singleton package dependencies. Support for partial trust packaged apps will be coming in later releases.

You should call the Deployment API after your app's process is initialized, but before your app uses Windows App SDK runtime features that use the Singleton package (for example, push notifications). The main methods of the Deployment API are the static GetStatus and Initialize methods of the DeploymentManager class.

- The **GetStatus** method returns the current deployment status of the Windows App SDK runtime that's currently loaded. Use this method to identify whether there's work required to install Windows App SDK runtime packages before the current app can use Windows App SDK features.
- The **Initialize** method verifies whether all required packages are present to a minimum version needed by the Windows App SDK runtime that's currently loaded. If any package dependencies are missing, then the method attempts to register those missing packages. Beginning in Windows App SDK 1.1, the **Initialize** method also supports the option to force-deploy the Windows App SDK runtime packages. That shuts down any processes for the *Main* and *Singleton* runtime packages, and thus interrupts their services (for example, push notifications won't deliver notifications during this time).

Deployment API sample app

For additional guidance on how to use the **GetStatus** and **Initialize** methods of the **DeploymentManager** class, explore the available sample app.

Explore Deployment API sample app

Address installation errors

If the Deployment API encounters an error during installation of the Windows App SDK runtime packages, it returns an error code that describes the problem.

For example, if your app is not full trust, or doesn't have the packageManagement restricted capability, then you'll get an **ACCESS_DENIED** error code. To review other error codes that you may encounter and their possible causes, see Troubleshooting packaging, deployment, and query of Windows apps.

If the error code doesn't provide enough information, then you can find more diagnostic information in the detailed event logs (see Get diagnostic information).

If you encounter errors that you can't diagnose, then file an issue in the WindowsAppSDK GitHub repo with the error code and event logs so that we can investigate the issue.

Related topics

- Deployment architecture for the Windows App SDK
- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Release channels
- Package your app using single-project MSIX

Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged

Article • 08/07/2024

This topic provides guidance about deploying apps that are packaged with external location, or are unpackaged, and that use the Windows App SDK.

- Such apps are desktop apps (not UWP apps).
- They can be written in a .NET language such as C#, or in C++.
- For their user-interface, they can use WinUI 3, or WPF, or WinForms, or another UI framework.

Overview

Developers of packaged with external location and unpackaged apps are responsible for deploying required Windows App SDK runtime packages to their end users. This can be done either by running the installer or by installing the MSIX packages directly. These options are described in more detail in the Deploy Windows App SDK runtime section below.

Packaged with external location and unpackaged apps also have extra runtime requirements. You must initialize access to the Windows App SDK runtime using the Bootstrapper API. In addition, the Dynamic Dependencies API can be used if your app makes use of other framework packages aside from the Windows App SDK. These requirements are described in more detail in the Runtime requirements for apps packaged with external location or unpackaged section below.

Prerequisites

- Download the latest installer & MSIX packages.
- For apps that are packaged with external location or unpackaged, the Visual C++ Redistributable is a requirement. For more info, see Microsoft Visual C++ Redistributable latest supported downloads.
- **C#**. .NET 6 or later is required. For more info, see .NET Downloads.

Additional prerequisites

- Experimental and preview versions of the Windows App SDK require that sideloading is enabled to install the runtime.
    - Sideloading is automatically enabled on Windows 10 version 2004 and later.
    - If your development computer or the deployment computer is running **Windows 11**, confirm whether sideloading is enabled:
        - **Settings** > **Privacy & security** > **For developers**. Make sure the **Developer mode** setting is turned on.
    - If your development computer or the deployment computer is running **Windows 10 version 1909 or an earlier version**, confirm whether sideloading is enabled:
        - **Settings** > **Update & Security** > **For developers** > **Use developer features**. Confirm that **Sideload apps** or **Developer mode** is selected.
    - The **Developer mode** setting includes sideloading as well as other features.

      > **Note**
      >
      > If the computer is managed in an enterprise environment, there might be a policy preventing these settings from being changed. In that case if you get an error when you or your app tries to install the Windows App SDK runtime, contact your IT Professional to enable sideloading or **Developer mode**.

Deploy Windows App SDK runtime

Packaged with external location and unpackaged apps have two options to deploy the Windows App SDK runtime:

- **Option 1: Use the Installer**: The silent installer distributes all Windows App SDK MSIX packages. A separate installer is available for each of the `X64`, `X86` and `Arm64` architectures.
- **Option 2: Install the packages directly**: You can have your existing setup or MSI tool carry and install the MSIX packages for the Windows App SDK.

Option 1: Use the Installer

You can deploy all Windows App SDK runtime packages by running the installer. The installer is available at Downloads for the Windows App SDK. When running the installer (.exe), you should see an output similar to the following:

Console

```
Deploying package:
Microsoft.WindowsAppRuntime.1.0_0.318.928.0_x64__8wekyb3d8bbwe
Package deployment result : 0x0

Deploying package:
Microsoft.WindowsAppRuntime.1.0_0.318.928.0_x86__8wekyb3d8bbwe
Package deployment result : 0x0

Deploying package:
MicrosoftCorporationII.WindowsAppRuntime.Main.1.0_0.318.928.0_x64__8wekyb3d8
bbwe
Package deployment result : 0x0
Provisioning result : 0x0

Deploying package:
Microsoft.WindowsAppRuntime.Singleton_0.318.928.0_x64__8wekyb3d8bbwe
Package deployment result : 0x0
Provisioning result : 0x0

Deploying package: Microsoft.WinAppRuntime.DDLM.0.318.928.0-
x6_0.318.928.0_x64__8wekyb3d8bbwe
Package deployment result : 0x0
Provisioning result : 0x0

Deploying package: Microsoft.WinAppRuntime.DDLM.0.318.928.0-
x8_0.318.928.0_x86__8wekyb3d8bbwe
Package deployment result : 0x0
Provisioning result : 0x0

All install operations successful.
```

You can run the installer with no user interaction and suppress all text output with the `--quiet` option:

Console

```
WindowsAppRuntimeInstall.exe --quiet
```

You can also choose to force update the MSIX packages and shutdown any currently running Windows App SDK processes using the `--force` option. This feature is introduced in 1.1.

Console

```
WindowsAppRuntimeInstall.exe --force
```

To see all installer command line options, run `WindowsAppRuntimeInstall --h`.

After the installation is complete, you can run your packaged with external location or unpackaged app. For an example of how to build and run a packaged with external location or unpackaged app that uses the Windows App SDK, see Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK.

Chain the Windows App SDK installer to your app's setup

If you have a custom setup program for your app, you can chain the Windows App SDK setup process in your app's setup process. The Windows App SDK installer currently does not provide a default UI so you will need to chain by using your setup's custom UI.

You can silently launch and track the Windows App SDK setup while showing your own view of the setup progress by using ShellExecute. The Windows App SDK installer silently unpacks the Windows App MSIX bundle and calls the PackageManager.AddPackageAsync method to complete the installation. This is very similar to other runtime installers you may have used, like .NET, Visual C++, or DirectX.

For a code example that demonstrates how to run the Windows App SDK installer from your setup program, see the **RunInstaller** function in the installer functional tests.

Installer sample

See the sample below to see how to launch the installer from a Win32 setup program without popping up a console window during setup:

Explore Installer sample

Troubleshooting

Return codes

The following table lists the most common return codes for the Windows App SDK .exe installer. The return codes are the same for all versions of the installer.

Expand table

Return code	Description
0x0	Package installation or provisioning was completed successfully.
0x80073d06	One or more packages failed to install.
0x80070005	System-wide install or provisioning was not possible because the app is not running elevated or the user doing the installation doesn't have admin privileges.

Installation errors

If the Windows App SDK installer returns an error during installation, it will return an error code that describes the problem.

- See the list of common error codes.
- If the error code doesn't provide enough information, you can find more diagnostic information in the detailed event logs.
- Please file an issue with the error code and event logs so the issue can be investigated.

Option 2: Deploy Windows App SDK runtime packages directly

As an alternative to using the Windows App SDK installer for deployment to end users, you can manually deploy the MSIX packages through your app's program or MSI. This option can be best for developers who want more control.

For an example that demonstrates how your setup program can install the MSIX packages, see install.cpp in the Windows App SDK installer code.

To check whether the Windows App SDK is installed already (and, if so, what version), you can check for specific package families by calling PackageManager.FindPackagesForUserWithPackageTypes.

From a *mediumIL* (full trust) unpackaged process (see Application element), you can use the following code to check for a package registered to the current user:

C#

```
using Windows.Management.Deployment;

public class WindowsAppSDKRuntime
{
    public static IsPackageRegisteredForCurrentUser(
        string packageFamilyName,
```

```
        PackageVersion minVersion,
        Windows.System.ProcessorArchitecture architecture,
        PackageTypes packageType)
    {
        ulong minPackageVersion = ToVersion(minVersion);

        foreach (var p : PackageManager.FindPackagesForUserWithPackageTypes(
            string.Empty, packageFamilyName, packageType)
        {
            // Is the package architecture compatible?
            if (p.Id.Architecture != architecture)
            {
                continue;
            }

            // Is the package version sufficient for our needs?
            ulong packageVersion = ToVersion(p.Id.Version);
            if (packageVersion < minPackageVersion)
            {
                continue;
            }

            // Success.
            return true;
        }

        // No qualifying package found.
        return false;
    }

    private static ulong ToVersion(PackageVersion packageVersion)
    {
        return ((ulong)packageVersion.Major << 48) |
               ((ulong)packageVersion.Minor << 32) |
               ((ulong)packageVersion.Build << 16) |
               ((ulong)packageVersion.Revision);
    }
}
```

For the scenario above, calling **FindPackagesForUserWithPackageTypes** is preferable to calling **FindPackagesForUser**. That's because you can narrow the search to (for this example), just *framework* or main *packages*. And that avoids matching other types of packages (such as *resource*, *optional*, or *bundle*) which aren't of interest for this example.

To use the current/calling user context, set the *userSecurityId* parameter is to an empty string.

And now some info to help you decide *how* to call the function in the code example above. A properly installed runtime is composed of multiple packages that depend on the system's CPU architecture:

- On an x86 machine: Fwk=[x86], Main=[x86], Singleton=[x86], DDLM=[x86].
- On an x64 machine: Fwk=[x86, x64], Main=[x64], Singleton=[x64], DDLM=[x86, x64].
- On an arm64 machine: Fwk=[x86, x64, arm64], Main=[arm64], Singleton=[arm64], DDLM=[x86, x64, arm64].

For the *Main* and *Singleton* packages, their architecture should match the system's CPU architecture; for example, x64 packages on an x64 system. For the *Framework* package, an x64 system can run both x64 and x86 apps; similarly an arm64 system can run arm64, x64, and x86 apps. A *DDLM* package check is similar to a *Framework* check, except that `PackageType=main`, and the *packagefamilyname* differs, and more than one (different) *packagefamilyname* could be applicable, due to *DDLM*'s unique naming scheme. For more info, see the MSIX packages spec. So the checks are more like this:

C#

```
public static bool IsRuntimeRegisteredForCurrentUser(PackageVersion
minVersion)
{
    ProcessorArchitecture systemArchitecture = DetectSystemArchitecture();

    return IsFrameworkRegistered(systemArchitecture, minVersion) &&
           IsMainRegistered(systemArchitecture, minVersion) &&
           IsSingletonRegistered(systemArchitecture, minVersion) &&
           IsDDLMRegistered(systemArchitecture, minVersion);
}

private static ProcecssorArchitecture DetectSystemArchitecture()
{
    // ...see the call to IsWow64Process2(), and how the result is used...
    // ...as per `IsPackageApplicable()` in
    // [install.cpp]
(https://github.com/microsoft/WindowsAppSDK/blob/main/installer/dev/install.
cpp)
    // line 99-116...
    // ...WARNING: Use IsWow64Process2 to detect the system architecture....
    // ...         Other similar APIs exist, but don't give reliably
accurate results...
}

private static bool IsFrameworkRegistered(ProcessorArchitecture
systemArchitecture,
    PackageVersion minVersion)
{
    // Check x86.
    if (!IsPackageRegisteredForCurrentUser(

global::Microsoft.WindowsAppSDK.Runtime.Packages.Framework.PackageFamilyName
,
        minVersion, ProcessorArchitecture.X86,
```

```
        PackageTypes.Framework))
    {
        return false;
    }

    // Check x64 (if necessary).
    if ((systemArchitecture == ProcessorArchitecture.X64) ||
        (systemArchitecture == ProcessorArchitcture.Arm64))
    {
        if (!IsPackageRegisteredForCurrentUser(

global::Microsoft.WindowsAppSDK.Runtime.Packages.Framework.PackageFamilyName
,
            minVersion, ProcessorArchitecture.X64,
            PackageTypes.Framework))
        {
            return false;
        }
    }

    // Check arm64 (if necessary).
    if (systemArchitecture == ProcessorArchitcture.Arm64)
    {
        if (!IsPackageRegisteredForCurrentUser(

global::Microsoft.WindowsAppSDK.Runtime.Packages.Framework.PackageFamilyName
,
            minVersion, ProcessorArchitecture.Arm64,
            PackageTypes.Framework))
        {
            return false;
        }
    }

    return true;
}

private static bool IsMainRegistered(ProcessorArchitecture
systemArchitecture,
    PackageVersion minVersion)
{
    return IsPackageRegisteredForCurrentUser(

global::Microsoft.WindowsAppSDK.Runtime.Packages.Main.PackageFamilyName,
        minVersion,
        systemArchitecture,
        PackageTypes.Main);
}

private static bool IsSingletonRegistered(ProcessorArchitecture
systemArchitecture,
    PackageVersion minVersion)
{
    return IsPackageRegisteredForCurrentUser(
```

```
global::Microsoft.WindowsAppSDK.Runtime.Packages.Singleton.PackageFamilyName
,
        minVersion,
        systemArchitecture,
        PackageTypes.Main);
}

private static bool IsDDLMRegistered(ProcessorArchitecture
systemArchitecture,
    PackageVersion minVersion)
{
    // ...similar to IsFrameworkRegistered, but the packageFamilyName is
more complicated...
    // ...and no predefined constant is currently available...
    // ...for more details, see
    //
https://github.com/microsoft/WindowsAppSDK/blob/main/specs/Deployment/MSIXPa
ckages.md.
}
```

The info and code above covers the basic detection scenario. To detect whether the runtime is provisioned for all users, or to do the above from an App Container, and/or do it from a packaged `mediumIL` process, additional logic is needed.

Deployment scenarios

- **Installing the Windows App SDK Runtime system-wide**: System-wide install alters the machine for all users, including new users that are added in the future. If the app is running elevated and the user doing the installation has admin privileges, then the installer will register the MSIX packages system-wide by calling the ProvisionPackageForAllUsersAsync. If system-wide registration is not successful, the installation will be performed for the current user doing the installation only. In a managed Enterprise environment, the IT admin should be able to provision for everyone as usual.

- **Architectures redistributed by the Windows App SDK installer**: The Windows App SDK installer is available in the `x86`, `x64`, and `Arm64` architectures. Each version of the installer includes the MSIX packages for just the specific architecture it's named for. For example, if you run the `x86` `WindowsAppRuntimeInstall.exe` on an x64 or and Arm64 device, then that `x86` installer will deploy onto that device only the packages for the x86 architecture.

- **All Windows App SDK MSIX packages are already installed on the computer**: MSIX packages are installed to a system-wide location with only one copy on disk. If an app attempts installation of the Windows App SDK when all the MSIX

package dependencies are already installed on the machine, then the installation is not performed.

- **One or more of the Windows App SDK MSIX packages are not installed on the computer**: When deploying the Windows App SDK, always attempt to install all the MSIX packages (framework, main, singleton, DDLM) to ensure that all dependencies are installed and you avoid disruption to the end-user experience.

Runtime requirements for apps packaged with external location or unpackaged

Apps that are packaged with external location or unpackaged have extra runtime requirements to use the Windows App SDK runtime. This involves referencing and initializing the Windows App SDK Framework package at runtime. In addition, the Dynamic Dependencies API can be used to reference other framework packages outside of the Windows App SDK.

Use the Windows App SDK runtime

Packaged with external location and unpackaged apps must call the Bootstrapper API to use the Windows App SDK at run time. This is required before the app can use Windows App SDK features such as WinUI, App Lifecycle, MRT Core, and DWriteCore. A bootstrapper component enables packaged with external location and unpackaged apps to perform these important tasks:

- Find and load the Windows App SDK framework package to the app's package graph.
- Initialize the Dynamic Dependency Lifetime Manager (DDLM) for the Windows App SDK framework package. The purpose of the DDLM is to prevent servicing of the Windows App SDK framework package while it is in use by a packaged with external location or unpackaged app.

The simplest way to load the Windows App SDK runtime for packaged with external location and unpackaged apps is by setting the `<WindowsPackageType>None</WindowsPackageType>` property in your project file (.csproj or .vcxproj). You may also call the bootstrapper API directly in your app's startup code for more control over the initialization. For more details, see Use the Windows App SDK runtime for apps packaged with external location or unpackaged and Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK.

Dynamic Dependencies support allows packaged with external location and unpackaged apps to keep their existing deployment mechanism, such as MSI or any installer, and be able to leverage the Windows App SDK in their application. Dynamic dependencies can be used by packaged, packaged with external location, and unpackaged apps; although it is primarily intended to be used by packaged with external location and unpackaged apps.

There is one DDLM for each version and architecture of the Windows App SDK framework package. This means on an `x64` computer, you may have both an `x86` and an `x64` version of the DDLM to support apps of both architectures.

Reference other framework packages using Dynamic Dependencies API

If you want to use features in other framework packages outside of the Windows App SDK (e.g., DirectX), packaged with external location and unpackaged apps can call the Dynamic Dependencies API. In addition to the bootstrapper component, the Windows App SDK also provides a broader set of C/C++ functions and WinRT classes that implement the *dynamic dependency API*. This API is designed to be used to reference any framework package dynamically at run time.

For more information, see Use MSIX framework packages dynamically from your desktop app and the Dynamic Dependencies sample

Deploy .winmd files to the target machine

Along with your app, we recommend that you go ahead and deploy Windows Metadata (`.winmd`) files. Metadata can be used by various APIs and behaviors at runtime, and its absence can limit or break functionality. For example, metadata can be used to marshal objects across apartments boundaries; and the need to marshal can be a function of machine performance. Since there's no deterministic way to know whether you need metadata, you should deploy `.winmd`s unless you're extremely concerned about size.

Related topics

- Deployment architecture for the Windows App SDK

- Windows App SDK deployment guide for framework-dependent packaged apps
- Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK

- Check for installed versions of the Windows App SDK runtime
- Remove outdated Windows App SDK runtime versions from your development computer

Use the Windows App SDK runtime for apps packaged with external location or unpackaged

Article • 04/19/2023

> **Note**
>
> If your app is installed by using MSIX technology, then see Windows App SDK deployment guide for framework-dependent packaged apps.

If your app isn't installed by using MSIX (that is, it's packaged with external location or unpackaged), then you must initialize the Windows App SDK for use before you can call Windows App SDK features such as WinUI 3, App Lifecycle, MRT Core, and DWriteCore. Your app must initialize the Windows App SDK runtime before using any other feature of the Windows App SDK.

- Beginning in Windows App SDK 1.0, that can be done automatically when your app starts via auto-initialization (set the project property `<WindowsPackageType>None</WindowsPackageType>`). For a demonstration, see Create your first WinUI 3 project.
- But if you have advanced needs (such as handling errors by showing your own custom UI or logging, or if you need to load a version of the Windows App SDK that's different from the version you built with), then continue reading this topic. In those scenarios, instead of auto-initialization, you can call the bootstrapper API explicitly.

Either of the two techniques above allows an app that doesn't use MSIX to take a dynamic dependency on the Windows App SDK at run time.

For background information about dynamic dependencies, see MSIX framework packages and dynamic dependencies.

Behind the scenes, and opting out of automatic module initialization

The code generated by the `WindowsPackageType` property mentioned above leverages module initializers to call the bootstrapper API. The bootstrapper does the heavy lifting to find the Windows App SDK and enable the current process to use it. The generated

code handles both initialization and shutdown. You can control initialization's behavior with the following project properties:

- `<WindowsAppSDKBootstrapAutoInitializeOptions_Default>true</WindowsAppSDKBootstrapAutoInitializeOptions_Default>`
  - Use default options.
- `<WindowsAppSDKBootstrapAutoInitializeOptions_None>true</WindowsAppSDKBootstrapAutoInitializeOptions_None>`
  - Use no options.
- `<WindowsAppSDKBootstrapAutoInitializeOptions_OnError_DebugBreak>true</WindowsAppSDKBootstrapAutoInitializeOptions_OnError_DebugBreak>`
  - Call DebugBreak() if an error occurs.
- `<WindowsAppSDKBootstrapAutoInitializeOptions_OnError_DebugBreak_IfDebuggerAttached>true</WindowsAppSDKBootstrapAutoInitializeOptions_OnError_DebugBreak_IfDebuggerAttached>`
  - Call DebugBreak() if an error occurs only if a debugger is attached to the process.
- `<WindowsAppSDKBootstrapAutoInitializeOptions_OnError_FailFast>true</WindowsAppSDKBootstrapAutoInitializeOptions_OnError_FailFast>`
  - Perform a fail-fast if an error occurs.
- `<WindowsAppSDKBootstrapAutoInitializeOptions_OnNoMatch_ShowUI>true</WindowsAppSDKBootstrapAutoInitializeOptions_OnNoMatch_ShowUI>`
  - Prompt the user to acquire the Windows App SDK runtime if a matching one cannot be found.
- `<WindowsAppSDKBootstrapAutoInitializeOptions_OnPackageIdentity_NoOp>true</WindowsAppSDKBootstrapAutoInitializeOptions_OnPackageIdentity_NoOp>`
  - Succceed if called in a process with package identity (otherwise it fails and an error is returned).

If you want your app to have explicit control, then you can directly call the boostrapper API early in your application's startup. In that case you don't need `WindowsPackageType` in your project file.

> **Note**
>
> In addition to the automatic initialization and the bootstrapper API, the Windows App SDK also provides an implementation of the *dynamic dependency API*. This API enables your unpackaged apps to take a dependency on *any* framework package (not just the Windows App SDK framework package), and it is used internally by the

bootstrapper API. For more information about the dynamic dependency API, see Use the dynamic dependency API to reference MSIX packages at run time.

Opting out of (or into) automatic module initialization

The project property `<WindowsAppSdkBootstrapInitialize>false</WindowsAppSdkBootstrapInitialize>` disables the automatic module initialization described above (the bootstrapper API isn't called). That allows your app to take responsibility and directly call the bootstrapper API.

As of version 1.2 of the Windows App SDK (from the stable channel), automatic module initialization applies only to projects that produce an executable (that is, the **OutputType** project property is set to *Exe* or *WinExe*). This is to prevent adding auto-initializers into class library DLLs and other non-executables by default. If you *do* need an auto-initializer in a non-executable (for example, a test DLL loaded by a host process executable that doesn't initialize the bootstrapper), then you can explicitly enable an auto-initializer in your project with `<WindowsAppSdkBootstrapInitialize>true</WindowsAppSdkBootstrapInitialize>`.

Using the bootstrapper API

> **Important**
>
> The **MddBootstrapInitialize2** function mentioned below is available starting in version 1.1.

The bootstrapper API consists of three C/C++ functions that are declared in the mddbootstrap.h header file in the Windows App SDK: **MddBootstrapInitialize**, **MddBootstrapInitialize2**, and **MddBootstrapShutdown**. Those functions are provided by the bootstrapper library in the Windows App SDK. That library is a small DLL that you must distribute with your app; it's not part of the framework package itself.

MddBootstrapInitialize2

This function initializes the calling process to use the version of the Windows App SDK framework package that best matches the criteria that you pass to the function parameters. Typically, that results in referencing the version of the framework package that matches the Windows App SDK NuGet package that's installed. If multiple packages meet the criteria, then the best candidate is selected. This function must be one of the

first calls in the app's startup to ensure the bootstrapper component can properly initialize the Windows App SDK and add the run-time reference to the framework package.

The bootstrapper API uses the Dynamic Dependencies API to add the Windows App SDK runtime's framework package to the current process's package graph and otherwise enable access to the package.

This function also initializes the Dynamic Dependency Lifetime Manager (DDLM). That component provides infrastructure to prevent the OS from servicing the Windows App SDK framework package while it's being used by an unpackaged app.

MddBootstrapShutdown

This function removes the changes to the current process that were made by a call to **MddBootstrapInitialize**. After this function is called, your app can no longer call Windows App SDK APIs, including the dynamic dependencies API.

This function also shuts down the Dynamic Dependency Lifetime Manager (DDLM) so that Windows can service the framework package as necessary.

.NET wrapper for the bootstrapper API

Although you can call the C/C++ bootstrapper API directly from .NET apps, that requires the use of platform invoke to call the functions. In Windows App SDK 1.0 and later releases, a .NET wrapper for the bootstrapper API is available in the `Microsoft.WindowsAppRuntime.Bootstrap.Net.dll` assembly. That assembly provides an easier and more natural API for .NET developers to access the bootstrapper's functionality. The **Bootstrap** class provides static **Initialize**, **TryInitialize**, and **Shutdown** functions that wrap calls to the **MddBootstrapInitialize** and **MddBootstrapShutdown** functions for most common scenarios. For an example that demonstrates how to use the .NET wrapper for the bootstrapper API, see the C# instructions in Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK.

For more information about the .NET wrapper for the bootstrapper API, see these resources:

- Bootstrapper C# APIs.
- Section 6.1.4 of the dynamic dependencies specification.
- Bootstrap.cs: The open source implementation of the .NET wrapper for the bootstrapper API.

C++ wrapper for the bootstrapper API

A C++ wrapper for the bootstrapper API is available starting in Windows App SDK 1.1.

See Bootstrapper C++ API.

Declare OS compatibility in your application manifest

To declare operating system (OS) compatibility, and to avoid the Windows App SDK defaulting to Windows 8 behavior (and potential crashes), you can include a side-by-side application manifest with your packaged with external location or unpackaged app. See Application manifests (it's the file that declares things like DPI awareness, and is embedded into your app's `.exe` during build). This might be an issue if you're adding Windows App SDK support to an existing app, rather than creating a new one via a Visual Studio project template.

If you don't already have a side-by-side application manifest in your project, then add a new XML file to your project, and name it as recommended in Application manifests. Add to the file the **compatibility** element and the child elements shown in the following example. These values control the quirks level for the components running in your app's process.

Replace the **Id** attribute of the **maxversiontested** element with the version number of Windows that you're targeting (must be 10.0.17763.0 or a later release). Note that setting a higher value means that older versions of Windows won't run your app properly because every Windows release knows only of versions before it. So if you want your app to run on Windows 10, version 1809 (10.0; Build 17763), then you should either leave the 10.0.17763.0 value as is, or add multiple **maxversiontested** elements for the different values that your app supports.

XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
    <compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
        <application>
            <!-- Windows 10, version 1809 (10.0; Build 17763) -->
            <maxversiontested Id="10.0.17763.0"/>
            <supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a15a9a}" />
        </application>
    </compatibility>
</assembly>
```

See Also

- Windows App SDK and supported Windows releases
- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Dynamic dependencies specification
- MSIX framework packages and dynamic dependencies
- Runtime architecture for the Windows App SDK
- Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK

Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK

Article • 03/17/2023

This article shows how to configure an app that's not installed by using MSIX (that is, it's packaged with external location or unpackaged) to use the bootstrapper API so that it explicitly loads the Windows App SDK runtime, and calls Windows App SDK APIs. Apps that are not installed via MSIX include apps packaged with external location, and unpackaged apps.

> **ⓘ Important**
>
> Beginning in the Windows App SDK 1.0, the default approach to loading the Windows App SDK from a packaged with external location or unpackaged app is to use *auto-initialization* via the `<WindowsPackageType>` project property (as well as making additional configuration changes). For the steps involved in auto-initialization in the context of WinUI 3 project, see Create your first WinUI 3 project. Or, if have an existing project that's not WinUI 3, then see Use the Windows App SDK in an existing project.
>
> If you have advanced needs (such as custom error handling, or to load a specific version of the Windows App SDK), then you can instead call the bootstrapper API explicitly. And that's the approach that this topic demonstrates. Also, for more info, see Use the Windows App SDK runtime for apps packaged with external location or unpackaged.

This topic demonstrates explicitly calling the bootstrapper API from a basic Console app project; but the steps apply to any unpackaged desktop app that uses the Windows App SDK.

Before completing this tutorial, we recommend that you review Runtime architecture to learn more about the *Framework* package dependency that your app takes when it uses the Windows App SDK, and the additional components required to work in a packaged with external location or unpackaged app.

Prerequisites

1. Install tools for the Windows App SDK.
2. Ensure that all dependencies for packaged with external location and unpackaged apps are installed (see Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged). An easy way to do that is to run the Windows App SDK runtime installer.

Instructions

You can follow this tutorial using a C# or a C++ project.

> **Note**
>
> The dynamic dependencies and bootstrapper APIs fail when called by an elevated process. As a result, Visual Studio shouldn't be launched elevated. See Dynamic Dependencies doesn't support elevation #567 for more details.

C#

Follow these instructions to configure a C# WinUI 3 project that's either packaged with external location, or unpackaged.

1. In Visual Studio, create a new C# **Console App** project. Name the project **DynamicDependenciesTest**. After you create the project, you should have a "Hello, World!" C# console app.

2. Next, configure your project.
   a. In **Solution Explorer**, right-click your project and choose **Edit Project File**.
   b. Replace the value of the **TargetFramework** element with a Target Framework Moniker. For example, use the following if your app targets Windows 10, version 2004.

   XML

   ```
   <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
   ```

   a. Save and close the project file.

3. Change the platform for your solution to **x64**. The default value in a .NET project is **AnyCPU**, but WinUI 3 doesn't support that platform.

a. Select **Build** > **Configuration Manager**.
b. Select the drop-down under **Active solution platform** and click **New** to open the **New Solution Platform** dialog box.
c. In the drop-down under **Type or select the new platform**, select **x64**.
d. Click **OK** to close the **New Solution Platform** dialog box.
e. In **Configuration Manager**, click **Close**.

4. Install the Windows App SDK NuGet package in your project.
   a. In **Solution Explorer**, right-click the **Dependencies** node and choose **Manage Nuget Packages**.
   b. In the **NuGet Package Manager** window, select the **Browse** tab, and install the **Microsoft.WindowsAppSDK** package.

5. You're now ready to use the bootstrapper API (see Use the Windows App SDK runtime for apps packaged with external location or unpackaged) to dynamically take a dependency on the Windows App SDK framework package. This enables you to use the Windows App SDK APIs in your app.

   Open the **Program.cs** code file, and replace the default code with the following code to call the **Bootstrap.Initialize** method to initialize the bootstrapper. This code defines what version of the Windows App SDK the app is dependent upon when initializing the bootstrapper.

   > **ⓘ Important**
   >
   > You'll need to edit the code below to suit your specific configuration. See the descriptions of the parameters of the Bootstrap.Initialize method so that you can specify one of the versions of the Windows App SDK that you have installed.

   C#

```
using System;
using Microsoft.Windows.ApplicationModel.DynamicDependency;

namespace DynamicDependenciesTest
{
    class Program
    {
        static void Main(string[] args)
        {
            Bootstrap.Initialize(0x00010002);
            Console.WriteLine("Hello, World!");

            // Release the DDLM and clean up.
```

```
            Bootstrap.Shutdown();
        }
    }
}
```

At its root, the bootstrapper API is a native C/C++ API that enables you to use the Windows App SDK APIs in your app. But in a .NET app that uses the Windows App SDK 1.0 or later, you can use the .NET wrapper for the bootstrapper API. That wrapper provides an easier way of calling the bootstrapper API in a .NET app than calling the native C/C++ functions directly. The previous code example calls the static **Initialize** and **Shutdown** methods of the **Bootstrap** class in the .NET wrapper for the bootstrapper API.

6. To demonstrate that the Windows App SDK runtime components were loaded properly, add some code that uses the ResourceManager class in the Windows App SDK to load a string resource.

   a. Add a new **Resources File (.resw)** to your project (leave the default name).

   b. With the resources file open in the editor, create a new string resource with the following properties.

      - Name: **Message**
      - Value: **Hello, resources!**

   c. Save the resources file.

   d. Open the **Program.cs** code file, and replace the `Console.WriteLine("Hello, World!");` line with the following code.

```
C#

// Create a resource manager using the resource index generated
during build.
   var manager = new
Microsoft.Windows.ApplicationModel.Resources.ResourceManager("Dynam
icDependenciesTest.pri");

// Look up a string in the .resw file using its name.
Console.WriteLine(manager.MainResourceMap.GetValue("Resources/Messa
ge").ValueAsString);
```

   a. Click **Start Without Debugging** (or **Start Debugging**) to build and run your app. You should see the string `Hello, resources!` successfully displayed.

If your project is WPF

For a Windows Presentation Foundation (WPF) app, see Use the Windows App SDK in a WPF app.

Related topics

- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Runtime architecture

Check for installed versions of the Windows App SDK runtime

Article • 08/19/2024

To check which versions of the Windows App SDK runtime are installed on your development computer, open a **PowerShell** window and run one of the following commands.

Powershell

```
# For 1.0 and 1.0 Preview releases and later
get-appxpackage *appruntime*
# For shorter output displaying only PackageFullName
(get-appxpackage micro*win*appruntime*).packagefullname

# For 1.0 Experimental
get-appxpackage *WindowsAppSDK*

# For version 0.8
get-appxpackage *reunion*
```

You should see output similar to the following, which will include the `x64`, `x86`, or `Arm64` versions of the Framework package, Dynamic Dependency Lifetime Manager (DDLM) package, Main package, and Singleton package, depending on your computer and Windows App SDK version.

Console

```
Name              : Microsoft.WindowsAppRuntime.1.0
Publisher         : CN=Microsoft Corporation, O=Microsoft Corporation,
L=Redmond, S=Washington, C=US
Architecture      : X64
ResourceId        :
Version           : 0.318.928.0
PackageFullName   :
Microsoft.WindowsAppRuntime.1.0_0.318.928.0_x64__8wekyb3d8bbwe
InstallLocation   : C:\Program
Files\WindowsApps\Microsoft.WindowsAppRuntime.1.0_0.318.928.0_x64__8wekyb3d8
bbwe
IsFramework       : True
PackageFamilyName : Microsoft.WindowsAppRuntime.1.0_8wekyb3d8bbwe
PublisherId       : 8wekyb3d8bbwe
IsResourcePackage : False
IsBundle          : False
IsDevelopmentMode : False
NonRemovable      : False
```

```
IsPartiallyStaged : False
SignatureKind     : Store
Status            : Ok

Name              : Microsoft.WindowsAppRuntime.1.0
Publisher         : CN=Microsoft Corporation, O=Microsoft Corporation,
L=Redmond, S=Washington, C=US
Architecture      : X86
ResourceId        :
Version           : 0.318.928.0
PackageFullName   :
Microsoft.WindowsAppRuntime.1.0_0.318.928.0_x86__8wekyb3d8bbwe
InstallLocation   : C:\Program
Files\WindowsApps\Microsoft.WindowsAppRuntime.1.0_0.318.928.0_x86__8wekyb3d8
bbwe
IsFramework       : True
PackageFamilyName : Microsoft.WindowsAppRuntime.1.0_8wekyb3d8bbwe
PublisherId       : 8wekyb3d8bbwe
IsResourcePackage : False
IsBundle          : False
IsDevelopmentMode : False
NonRemovable      : False
IsPartiallyStaged : False
SignatureKind     : Store
Status            : Ok

Name              : MicrosoftCorporationII.WindowsAppRuntime.Main.1.0
Publisher         : CN=Microsoft Corporation, O=Microsoft Corporation,
L=Redmond, S=Washington, C=US
Architecture      : X64
ResourceId        :
Version           : 0.318.928.0
PackageFullName   :
MicrosoftCorporationII.WindowsAppRuntime.Main.1.0_0.318.928.0_x64__8wekyb3d8
bbwe
InstallLocation   : C:\Program

Files\WindowsApps\MicrosoftCorporationII.WindowsAppRuntime.Main.1.0_0.318.92
8.0_x64__8wekyb3d8bbwe
IsFramework       : False
PackageFamilyName :
MicrosoftCorporationII.WindowsAppRuntime.Main.1.0_8wekyb3d8bbwe
PublisherId       : 8wekyb3d8bbwe
IsResourcePackage : False
IsBundle          : False
IsDevelopmentMode : False
NonRemovable      : False
Dependencies      :
{Microsoft.WindowsAppRuntime.1.0_0.318.928.0_x64__8wekyb3d8bbwe}
IsPartiallyStaged : False
SignatureKind     : Store
Status            : Ok

Name              : Microsoft.WindowsAppRuntime.Singleton
Publisher         : CN=Microsoft Corporation, O=Microsoft Corporation,
```

```
L=Redmond, S=Washington, C=US
Architecture      : X64
ResourceId        :
Version           : 0.318.928.0
PackageFullName   :
Microsoft.WindowsAppRuntime.Singleton_0.318.928.0_x64__8wekyb3d8bbwe
InstallLocation   : C:\Program
Files\WindowsApps\Microsoft.WindowsAppRuntime.Singleton_0.318.928.0_x64__8we
kyb3d8bbwe
IsFramework       : False
PackageFamilyName : Microsoft.WindowsAppRuntime.Singleton_8wekyb3d8bbwe
PublisherId       : 8wekyb3d8bbwe
IsResourcePackage : False
IsBundle          : False
IsDevelopmentMode : False
NonRemovable      : False
Dependencies      :
{Microsoft.WindowsAppRuntime.1.0_0.318.928.0_x64__8wekyb3d8bbwe}
IsPartiallyStaged : False
SignatureKind     : Store
Status            : Ok

Name              : Microsoft.WinAppRuntime.DDLM.0.318.928.0-x6
Publisher         : CN=Microsoft Corporation, O=Microsoft Corporation,
L=Redmond, S=Washington, C=US
Architecture      : X64
ResourceId        :
Version           : 0.318.928.0
PackageFullName   : Microsoft.WinAppRuntime.DDLM.0.318.928.0-
x6_0.318.928.0_x64__8wekyb3d8bbwe
InstallLocation   : C:\Program

Files\WindowsApps\Microsoft.WinAppRuntime.DDLM.0.318.928.0-
x6_0.318.928.0_x64__8wekyb3d8bbwe
IsFramework       : False
PackageFamilyName : Microsoft.WinAppRuntime.DDLM.0.318.928.0-
x6_8wekyb3d8bbwe
PublisherId       : 8wekyb3d8bbwe
IsResourcePackage : False
IsBundle          : False
IsDevelopmentMode : False
NonRemovable      : False
Dependencies      :
{Microsoft.WindowsAppRuntime.1.0_0.318.928.0_x64__8wekyb3d8bbwe}
IsPartiallyStaged : False
SignatureKind     : Developer
Status            : Ok

Name              : Microsoft.WinAppRuntime.DDLM.0.318.928.0-x8
Publisher         : CN=Microsoft Corporation, O=Microsoft Corporation,
L=Redmond, S=Washington, C=US
Architecture      : X86
ResourceId        :
Version           : 0.318.928.0
PackageFullName   : Microsoft.WinAppRuntime.DDLM.0.318.928.0-
```

```
x8_0.318.928.0_x86__8wekyb3d8bbwe
InstallLocation   : C:\Program

Files\WindowsApps\Microsoft.WinAppRuntime.DDLM.0.318.928.0-
x8_0.318.928.0_x86__8wekyb3d8bbwe
IsFramework       : False
PackageFamilyName : Microsoft.WinAppRuntime.DDLM.0.318.928.0-
x8_8wekyb3d8bbwe
PublisherId       : 8wekyb3d8bbwe
IsResourcePackage : False
IsBundle          : False
IsDevelopmentMode : False
NonRemovable      : False
Dependencies      :
{Microsoft.WindowsAppRuntime.1.0_0.318.928.0_x86__8wekyb3d8bbwe}
IsPartiallyStaged : False
SignatureKind     : Developer
Status            : Ok
```

Related topics

- Windows App SDK and supported Windows releases
- Runtime architecture
- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Windows App SDK deployment guide for framework-dependent packaged apps
- Remove outdated Windows App SDK runtime versions from your development computer

Remove outdated Windows App SDK runtime versions from your development computer

Article • 04/19/2023

As you install updated versions of the Windows App SDK runtime over time, you may want to remove outdated versions from your development computer. In general, you only need the Windows App SDK runtime and extension that you have chosen to reference and require in your application. Unless your application has specific reasons for earlier extensions or runtimes, you may safely remove older versions.

The Windows App SDK runtime includes the Framework, Main, Singleton, and Dynamic Dependency Lifetime Manager (DDLM) packages. All of these packages can be uninstalled by using **PowerShell** commands. The Main, Singleton, and DDLM packages can alternatively be uninstalled in **Settings**.

> **ⓘ Important**
>
> We recommend that you do not remove Windows App SDK runtime packages from end-user's machines after deployment. Doing so will break other applications that are taking a dependency on those packages.

Remove Windows App SDK runtime components using PowerShell

First, run the `get-appxpackage` command to list all versions of the Windows App SDK runtime packages that are installed on your computer (including the framework, main, singleton, and DDLM packages). The list includes details about each runtime package, including the **PackageFullName** value. For an example of the output of this command, see Check for installed versions of the Windows App SDK runtime.

Powershell

```
# For 1.0 and 1.0 Preview releases
get-appxpackage *appruntime*

# For 1.0 Experimental
get-appxpackage *WindowsAppSDK*
```

```
# For version 0.8
get-appxpackage *reunion*
```

Next, run the `remove-appxpackage` command to remove specific versions of the Windows App SDK runtime packages from your development computer. For each package, specify the **PackageFullName** value of the runtime packages you want to remove, as provided by the preceding step. The following example removes the framework, main, and DDLM package, respectively. Make sure you specify the correct **PackageFullName** values for the packages on your computer.

Powershell

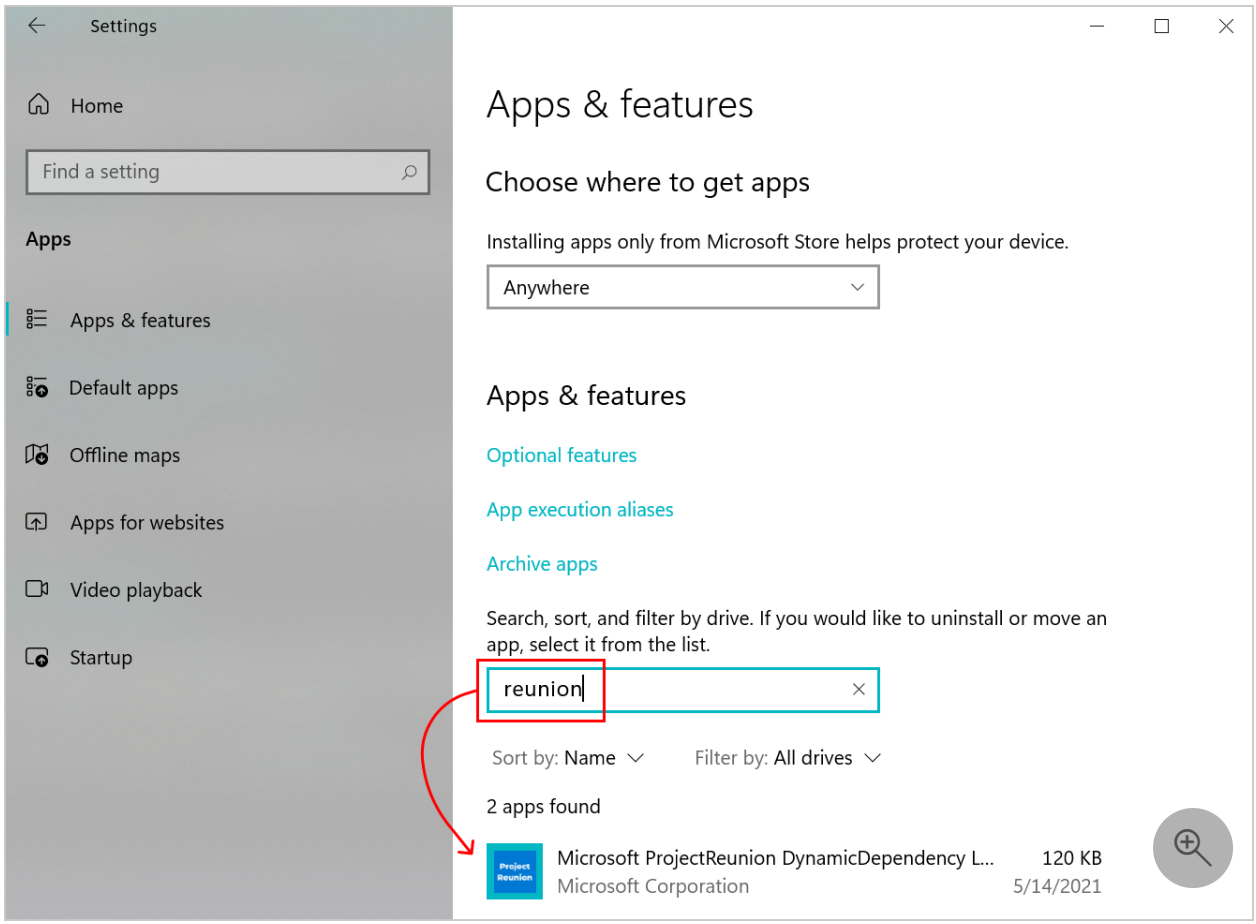
```
remove-appxpackage -Package Microsoft.ProjectReunion.0.8-
preview_8000.144.525.0_x86__8wekyb3d8bbwe
remove-appxpackage -Package Microsoft.ProjectReunion.Main.0.8-
preview_8000.144.525.0_x64__8wekyb3d8bbwe
remove-appxpackage -Package Microsoft.ProjectReunion.DDLM.8000.144.525.0-x8-
p_8000.144.525.0_x86__8wekyb3d8bbwe
```

Remove main and DDLM packages in Settings

To remove the Main and Dynamic Dependency Lifetime Manager (DDLM) packages, you can alternatively use the **Apps & features** page in **Settings**.

> ⓘ **Note**
>
> The Windows App SDK framework package will not show up in **Apps & features**, and will need to be uninstalled using the **PowerShell** commands described earlier in this article.

On the **Apps & features** page, search for **reunion** to filter and show installed versions of Windows App SDK packages. Select any versions you want to remove from your computer and click **Uninstall**.



Related topics

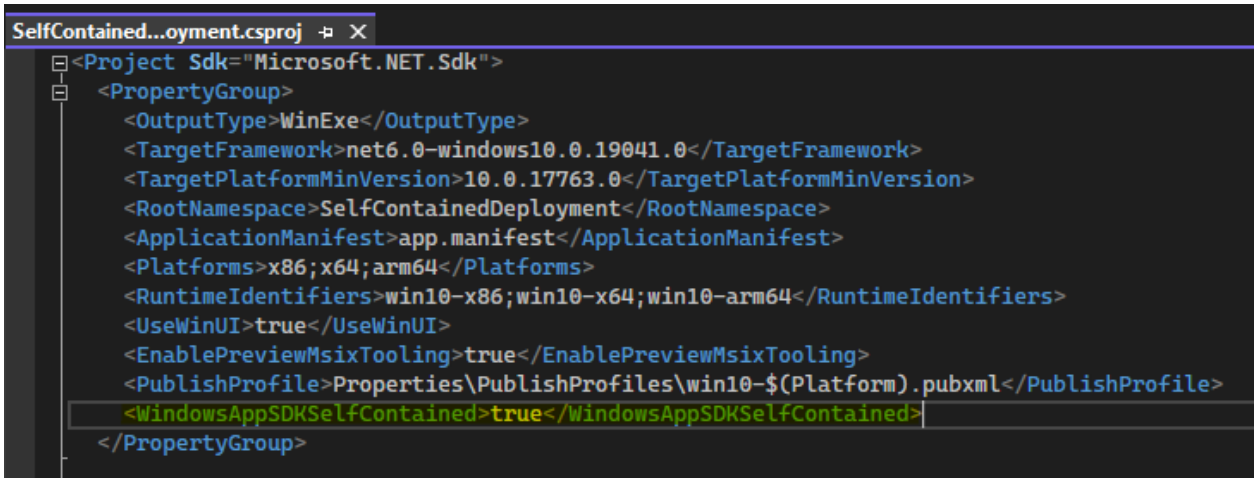
- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Runtime architecture
- Check for installed versions of the Windows App SDK runtime
- Install tools for the Windows App SDK
- Create your first WinUI 3 project
- Use the Windows App SDK in an existing project

Windows App SDK deployment guide for self-contained apps

Article • 08/11/2023

A Windows App SDK project is framework-dependent by default. To switch to self-contained deployment, follow the steps below (the terms *framework-dependent* and *self-contained* are described in Windows App SDK deployment overview).

- In Visual Studio, right-click the app project node, and click **Edit Project File** to open the app project file for editing. For a C++ project, first click **Unload Project**.
- In the app project file, inside the main `PropertyGroup`, add `<WindowsAppSDKSelfContained>true</WindowsAppSDKSelfContained>` as shown in the screenshot below.



- For packaged projects, in the app project file, at the end of the file before the closing `</Project>`, add the `Target` shown below.

```
  <Target Name="_RemoveFrameworkReferences"
BeforeTargets="_ConvertItems;_CalculateInputsForGenerateCurrentProjectAppxMa
nifest">
    <ItemGroup>
      <FrameworkSdkReference Remove="@(FrameworkSdkReference)"
Condition="$([System.String]::Copy('%
(FrameworkSdkReference.SDKName)').StartsWith('Microsoft.WindowsAppRuntime.')
)" />
    </ItemGroup>
  </Target>
```

> **Note**

This is a workaround for a bug in Windows App SDK 1.1 and will not be necessary with Windows App SDK 1.2. It is only required for packaged projects.

- Save and close the project file.
- Click **Reload Project**.
- If you're using a **Windows Application Packaging Project** (rather than the single-project MSIX that you get with **Blank App, Packaged (WinUI 3 in Desktop)**), then make all of the above changes in the project file for the packaging project as well.

> **Note**
>
> Library projects should not be changed. Self-contained deployment should only be configured in app projects (and, where applicable, in a **Windows Application Packaging Project**).

For sample apps, see Windows App SDK self-contained deployment samples.

Having set the `WindowsAppSDKSelfContained` property to `true` in your project file, the contents of the Windows App SDK Framework package will be extracted to your build output, and deployed as part of your application.

> **Note**
>
> .NET apps need to be **published as self-contained** as well to be fully self-contained. See **this sample** for how to configure .NET self-contained with publish profiles. `dotnet publish` is not yet supported with Windows App SDK 1.1.

> **Note**
>
> C++ apps need to use the **hybrid CRT** as well to be fully self-contained. Importing **HybridCRT.props** from **Directory.Build.props** is the recommended way to configure it for all projects in a solution (see an example in **Directory.Build.props**). A packaged app must also set `<UseCrtSDKReferenceStaticWarning>false</UseCrtSDKReferenceStaticWarning>` in their project file. See the **Self-contained deployment** sample app for how to use the hybrid CRT.

If your app is packaged (for more info, see Deployment overview), then the Windows App SDK dependencies will be included as content inside the MSIX package. Deploying

the app still requires registering the MSIX package like any other packaged app.

If your app is packaged with external location or unpackaged, then the Windows App SDK dependencies are copied next to the `.exe` in your build output. You can xcopy-deploy the resulting files, or include them in a custom installer.

Dependencies on additional MSIX packages

A small number of APIs in the Windows App SDK rely on additional MSIX packages that represent critical operating system (OS) functionality.

- For example (as of the Windows App SDK 1.1), push notifications APIs (PushNotificationManager) and app notifications APIs (AppNotificationManager) have a dependency on the *Singleton* package (see Deployment architecture for the Windows App SDK).

That means that if you want to use those APIs in a self-contained app, then you have the following options:

1. You could make your functionality optional, and light it up *only if and when possible*. Calling the APIs' **IsSupported** method (PushNotificationManager.IsSupported and AppNotificationManager.IsSupported) will let you check dynamically at runtime whether or not the APIs are available to the calling app on the system it's running on.

    - This enables safe, conditional, optional use of the APIs without compromising the simplicity of your self-contained deployment.
    - Only if the OS services are installed outside of your app deployment will your app light up the appropriate functionality. But in fact there are *some* cases where the APIs will work even without the Singleton package being present; so calling **IsSupported** to check is often a good idea.

2. Deploy the required MSIX packages as part of your app installation.

    - This allows you to depend on the API in all scenarios. But requiring MSIX package deployment of dependencies as part of your app deployment can compromise the simplicity of self-contained deployment.

3. Don't use the API.

    - Consider alternative APIs that provide similar functionality without additional deployment requirements.

Opting out of (or into) automatic UndockedRegFreeWinRT support

The project property **WindowsAppSdkUndockedRegFreeWinRTInitialize** was introduced in version 1.2 of the Windows App SDK (from the stable channel). If that property is set to *true* then it ensures that the Windows App SDK's implementation of undocked registration-free Windows Runtime (*UndockedRegFreeWinRT*) is enabled automatically at app startup. That support is needed by unpackaged self-contained apps.

WindowsAppSdkUndockedRegFreeWinRTInitialize defaults to *true* if **WindowsAppSDKSelfContained** is *true* and **WindowsPackageType** is *None* and (as of version 1.2 of the Windows App SDK) **OutputType** project property is set to *Exe* or *WinExe* (that is, the project produces an executable). That last condition is to prevent adding automatic UndockedRegFreeWinRT support into class library DLLs and other non-executables by default. If you *do* need automatic UndockedRegFreeWinRT support in a non-executable (for example, a test DLL loaded by a host process executable that doesn't initialize UndockedRegFreeWinRT), then you can explicitly enable it in your project with `<WindowsAppSdkUndockedRegFreeWinRTInitialize>true</WindowsAppSdkUndockedRegFreeWinRTInitialize>`.

Related topics

- Windows App SDK deployment overview
- Windows App SDK self-contained deployment samples
- Deployment overview
- Deployment architecture for the Windows App SDK

Set up continuous integration for your WinUI 3 app

Article • 03/20/2023

You can use GitHub Actions to set up continuous integration builds for WinUI 3 projects. In this article, we'll look at different ways to do this. We'll also show you how to perform these tasks by using the command line so that you can integrate with any other build system.

Prerequisites

- Start with a single-project MSIX WinUI 3 app or migrate your project to use single-project MSIX.
- Sign up for GitHub and create a repository if you haven't done so already.

Step 1: Set up your certificate

MSIX apps must be signed in order to be installed. If you already have a certificate, you can skip this step. You can easily create a test certificate by opening your app in Visual Studio, right clicking your WinUI 3 project, and selecting **Package and Publish** -> **Create App Packages**.

Then select **Next** to move to the **Select signing method** page, and click the **Create...** button to create a new certificate. Choose the publisher name and **leave the password field blank**, and create the certificate.

Then, close/cancel out of the dialogs and notice that a new **.pfx** file has been created in your project. This is the certificate you can sign your MSIX with!

Step 2: Add your certificate to the Actions secrets

You should avoid submitting certificates to your repo if at all possible, and git ignores them by default. To manage the safe handling of sensitive files like certificates, GitHub supports secrets.

To upload a certificate for your automated build:

1. **Encode your certificate as a Base 64 string**: Open PowerShell to the directory that contains your certificate, and execute the following command, replacing the pfx file name with your certificate's file name.

PowerShell

```
$pfx_cert = Get-Content 'App1_TemporaryKey.pfx' -AsByteStream

[System.Convert]::ToBase64String($pfx_cert) | Out-File
'App1_TemporaryKey_Base64.txt'
```

2. In your GitHub repository, go to the **Settings** page and click **Secrets** on the left.
3. Click **New repository secret**, name it `BASE64_ENCODED_PFX`, and copy/paste the text from the text file in the PowerShell output into the secret value.

Step 3: Set up your workflow

Next, in your repository, go to the Actions tab and create a new workflow. Chose the **set up a workflow yourself** option instead of one of the workflow templates.

Copy/paste the following into your workflow file, and then update...

1. **Solution_Name** to the name of your solution
2. **dotnet-version** to either 5.0 or 6.0 depending on your project

> ⓘ Note
>
> For the step uploading the artifact (the last step below), if the build output doesn't land in a folder that contains your solution, then replace `env.Solution_Name` with `github.workspace` (the GitHub actions Workspace folder).

yml

```
# This workflow will build, sign, and package a WinUI 3 MSIX desktop
application
# built on .NET.

name: WinUI 3 MSIX app

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]
```

```yaml
jobs:

  build:

    strategy:
      matrix:
        configuration: [Release]
        platform: [x64, x86]

    runs-on: windows-latest  # For a list of available runner types, refer
to
                             #
https://help.github.com/en/actions/reference/workflow-syntax-for-github-
actions#jobsjob_idruns-on

    env:
      Solution_Name: your-solution-name                         # Replace
with your solution name, i.e. App1.sln.

    steps:
    - name: Checkout
      uses: actions/checkout@v2
      with:
        fetch-depth: 0

    # Install the .NET Core workload
    - name: Install .NET Core
      uses: actions/setup-dotnet@v1
      with:
        dotnet-version: 6.0.x

    # Add  MSBuild to the PATH: https://github.com/microsoft/setup-msbuild
    - name: Setup MSBuild.exe
      uses: microsoft/setup-msbuild@v1.0.2

    # Restore the application to populate the obj folder with
RuntimeIdentifiers
    - name: Restore the application
      run: msbuild $env:Solution_Name /t:Restore
/p:Configuration=$env:Configuration
      env:
        Configuration: ${{ matrix.configuration }}

    # Decode the base 64 encoded pfx and save the Signing_Certificate
    - name: Decode the pfx
      run: |
        $pfx_cert_byte = [System.Convert]::FromBase64String("${{
secrets.BASE64_ENCODED_PFX }}")
        $certificatePath = "GitHubActionsWorkflow.pfx"
        [IO.File]::WriteAllBytes("$certificatePath", $pfx_cert_byte)

    # Create the app package by building and packaging the project
    - name: Create the app package
      run: msbuild $env:Solution_Name /p:Configuration=$env:Configuration
/p:Platform=$env:Platform
```

```
/p:UapAppxPackageBuildMode=$env:Appx_Package_Build_Mode
/p:AppxBundle=$env:Appx_Bundle
/p:PackageCertificateKeyFile=GitHubActionsWorkflow.pfx
/p:AppxPackageDir="$env:Appx_Package_Dir" /p:GenerateAppxPackageOnBuild=true
      env:
        Appx_Bundle: Never
        Appx_Package_Build_Mode: SideloadOnly
        Appx_Package_Dir: Packages\
        Configuration: ${{ matrix.configuration }}
        Platform: ${{ matrix.platform }}

    # Remove the pfx
    - name: Remove the pfx
      run: Remove-Item -path GitHubActionsWorkflow.pfx

    # Upload the MSIX package:
https://github.com/marketplace/actions/upload-a-build-artifact
    - name: Upload MSIX package
      uses: actions/upload-artifact@v2
      with:
        name: MSIX Package
        path: ${{ env.Solution_Name }}\\Packages
```

Step 4: Commit the workflow and watch it run!

Commit the workflow file to your main branch, and then go to the Actions tab on your GitHub repository and watch your workflow run! It should successfully run and produce artifacts that contain your built MSIX app.

Building from command line

If you want to build your solution by using the command line, or by using any other CI system, run MSBuild with these arguments. The `GenerateAppxPackageOnBuild` property causes the MSIX package to be generated.

PowerShell

```
/p:AppxPackageDir="Packages"
/p:UapAppxPackageBuildMode=SideloadOnly
/p:AppxBundle=Never
/p:GenerateAppxPackageOnBuild=true
```

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Windows app restore: maximize the value of your app

Article • 09/29/2023

To maximize retention of your users as they move to a new device, your app (in conjunction with Windows app restore) should offer the best possible restore experience. This topic defines the tenets of Windows app restore that will enable your app to deliver that experience and keep as many of your customers engaged as possible.

Why app restore is critical

When the users of your app move to their next Windows PC, they need the peace of mind that their apps will transfer over to their new device.

Windows will back up the user's app list to the cloud; during restore, it will create pinned app placeholders on the new Windows PC so that users find their apps right where they expect them to be. This makes installation of the app very straightforward. However, in addition to the application install, users also want the rest of their app configuration and settings to transfer over to their new device.

If you ensure that your apps are following best practices to maximize the effectiveness of Windows app restore capabilities, then your users will be able to get back to productivity as quickly as possible on their new PC. This will, in turn, allow your app to retain your users on their new PC. The rest of this topic discusses those best practices.

App restore tenets

These tenets are guidelines and best practices for you to enable an optimal backup and restore experience for the users of your apps. These are a collection of existing best practices that we've gathered.

- **Publish your app to the Microsoft Store**. Your app should enable trusted distribution through the Microsoft Store (see Publish Windows apps and games). This is the easiest way for you to provide a trusted and seamless (think fewer clicks) experience for your users.
- **Package your app**. Your app should be *packaged* (for definitions, see Deployment overview). A packaged app enables the system to better understand the files, data,

and settings that matter to an app; and enables the system to more easily restore apps on demand.

- **Store critical app state in the cloud**. Your app should store its critical app state in the cloud. Having an app installed on a new device is only the first step. Getting users seamless back to their prior app state—their recents, their favorites, their preferences—is the goal; and the best way to do this is to store that critical user state information in the cloud. Local state should be thought of as only a temporary optimization.
- **Write user-generated content to Known Folders**. Your app should write user-generated content to the Windows known folders (see the **KnownFolders** class). Keep it separate from app state—if your app produces user-generated content (files, sound clips, videos, etc.), then you should write that content to the Windows Known Folders (Documents, Pictures, Music, Videos, etc.). This enables Windows, via OneDrive, to back up those files to the cloud and fluidly keep in sync across devices using its files-on-demand technology.

Publish your app to the Microsoft Store

The Microsoft Store is the most reliable distribution cloud for your Windows apps (see Publish Windows apps and games). Users can easily search for and find your app for installation.

When a user installs a new operating system (OS), and chooses to restore from their previous PC, those apps that are in the Microsoft Store will automatically be listed in the **All Apps** list, and have pins available in the same locations on the **Start** menu and taskbar as before. Those shortcuts allow the user to immediately access the app and install it.

Package your app

Another key to creating a great experience for your users is ensuring that the app gets installed, and behaves correctly. The best way to do that is with a *packaged* app (see Deployment overview).

A packaged app (either a packaged desktop app or a Universal Windows Platform app) is packaged using MSIX, and it's run inside of a lightweight app container. The packaged app process and its child processes run inside the container; and they're isolated using file system and registry virtualization. It's these aspects of packaging that make installation extremely reliable, and ensure that the app doesn't misbehave or leave registry configuration or app files on the PC when the user uninstalls.

For more info about the benefits of using MSIX for packaging, see What is MSIX?.

The benefits of the MSIX format, and packaged apps, don't end with app reliability. Packaging your app also means that it will be able to be quickly installed when users migrate to a new PC. Following an install, Windows will begin rehydrating the packaged apps that it restored. Since rehydrating takes time, if the user clicks the link before the app is rehydrated, then Windows will immediately download and install the app, allowing the user to run it as early as possible.

Store critical app state in the cloud

As you can see, Windows does a great job in helping your customers find and install your app on their new PC. But what about app data, such as app settings? To deliver the best user experience, we recommend that you use the cloud to store your app's state. By storing app data in the cloud, your users can have a consistent experience across devices. And when users don't need to reconfigure their app settings, your user satisfaction rises dramatically.

Storing app settings to the cloud requires a service. To provide as rich an experience as possible, Microsoft provides a variety of services that eliminate the need to spin up servers, or pick your database, or worry about scale or security. Those services provide a great developer experience that lets you store application data in the cloud by using SQL or NoSQL APIs. To help build scalable and robust applications, you can also sync data on all devices, and enable the application to work with or without a network connection. For more info about Microsoft services, see Store, sync, and query mobile application data from the cloud.

For more info about the best practices of storing app data, see Store and retrieve settings and other app data.

Write user-generated content to Known Folders

Windows introduced known folders with Windows Vista. Since that time, users have come to expect that they can find the content they create with their apps in those locations. Writing user-generated content to those locations has the added benefit that OneDrive will back up those folders, if enabled, to ensure they're available to the user on their new PC (see Back up your folders with OneDrive). By using standard Windows APIs to write your user-generated content to the known folders, you're improving the user experience, and decreasing friction in adopting your app.

User-visible files

You should store files that you wish a user to see and to interact with in the appropriate folder in the user's profile. You should store general files in the `FOLDERID_Documents` location; typically in a sub-folder. And you should store pictures, music, and video in their appropriate `FOLDERID_Pictures`, `FOLDERID_Music`, and `FOLDERID_Videos` locations.

Machine-specific app data

You should store data that's specific to the machine on which the app is currently running in the `FOLDERID_LocalAppData` folder; normally in a sub-folder. That includes data such as:

- System performance metrics. Information gathered and persisted about the current machine, and used to optimize the behavior of the app on that specific machine. For example, if you've gathered info about the machine's graphics capabilities and performance (in order to determine the optimal rendering quality), then you shouldn't roam that data.
- User customizations connected with machine-specific capabilities. An app that optimizes its rendering performance based on the machine's graphics capabilities and performance should also store any changes that it allows the user to make to those preferences as machine-specific data. That ensures that the user enjoys what they determine to be the best experience for the machine they happen to be running on the app on.

> **Tip**
>
> The reason we advise not to store machine-specific data in known folders is that those user-specific folders travel with the user between machines (they *roam*). So storing machine-specific data can result in conflicts and problems when users use your app on multiple machines, or after an upgrade.

App data that's not machine-specific

You should store data that's not machine-specific in the `FOLDERID_Documents` location; typically in a sub-folder. Those files often contain user-provided app customization such as: default action to perform on launch; custom backgrounds; or other data that shouldn't change from one machine to another.

Best practices for unpackaged apps

If you can't package your app, then be sure that your installer implements the recommendations below. That will ensure that it's possible to backup and restore the **Start** menu shortcuts that enable installing on a new machine that's restored from backup.

- Make sure that your installer specifies an `InstallLocation` value in its uninstall registry key. When using Windows Installer specify this using ARPINSTALLLOCATION. That's needed in order to enable the mapping of the **Start** menu shortcuts to the product.
- Make sure that that location is specific to the product; usually the sub-directory under `C:\Program Files\<Publisher>\<Application>`.
- Make sure that your **Start** menu shortcuts have machine-independent `System.AppUserModel.ID` (AMUID) values. That's best done by specifying them explicitly in the shortcut metadata. For more info, see Where to Assign an AppUserModelID.

Get started: Publish your first app in the Microsoft Store

Article • 12/04/2024

Introduction

Distributing apps via Microsoft Store is a good choice for developers of any app type and size. The Microsoft Store is a centralized hub for Windows users to discover and install a wide range of apps, providing developers access to a vast audience of over a billion users across Windows 10 and Windows 11. Microsoft Store also offers you various ways to make money from your apps, and lets you choose your own commerce platform and revenue sharing model. Microsoft Store also supports a wide range of app types and technologies, and allows you to bring your traditional desktop apps to the Store without changing your code or installer.

For the full benefits of publishing your app into the Microsoft Store, please visit benefits of distributing your apps via Microsoft Store.

Prerequisite

A Windows app developer account in Partner Center is needed before you start the app submission process. There are 2 types of developer accounts available in Partner Center: **Individual and Company**.

To understand how to open developer account, you can watch the following video:

Open individual developer account

Who should select an individual account:

- **Independent developers** whose distribution of apps through the Store is **not in relation to their business, trade, or profession**
- **Small scale creators** producing content for non-commercial purposes
- Individuals creating digital content as a **hobbyist, amateur, school, or personal project**

The individual account costs approximately **$19 USD**. It is a one-time registration fee. (The fees varies depending on your country or region)

https://learn-video.azurefd.net/vod/player?id=663fa063-4205-4b88-a473-4a2782d49197&locale=en-us&embedUrl=%2Fwindows%2Fapps%2Fpublish%2F

For more details, refer steps to create individual developer account.

Get started with app submission

MSIX or PWA

We recommend packaging your app (which is built with any app framework - UWP, Win32, PWA, WinApp SDK etc.), as MSIX. By packaging your app as MSIX, you can take advantages of many features like a complimentary binary hosting (provided by Microsoft), complementary code signing (provided by Microsoft), Microsoft Store commerce platform, package flighting, advanced integration with Windows (to use features like share dialog, launch from Store etc), Windows 11 backup and restore etc.

For packaging your Win32 app as MSIX, follow these steps.

Note: If you distribute your application as a web download (EXE /MSI) and you are planning to distribute it as a packaged application (MSIX) in the Store, you might want to prevent users from installing both versions or migrate users from the unpackaged web version to the Store version. Learn more about transitioning users from web unpackaged to Store packaged app.

Choice of commerce platform

Microsoft Store offers developers a flexible and transparent revenue sharing model (including in-app purchases, subscriptions, ads, and tips) that lets you choose your own commerce platform and keep 100% of the revenue for non-gaming apps, or use Microsoft's commerce platform and pay a competitive fee of 12% for games and 15% for apps. This means that you can maximize your profit and control your business model, while benefiting from the convenience and security of Microsoft's commerce platform.

App submission [MSIX/PWA]

To understand the submission process of MSIX apps, you can watch the following video.

https://learn-video.azurefd.net/vod/player?id=4f68f734-a116-4cae-a0a3-89a415087e61&locale=en-us&embedUrl=%2Fwindows%2Fapps%2Fpublish%2F

For more details, refer to the steps below.

Reserve your app's name [MSIX/PWA]

All apps on the Microsoft Store must have a unique name. To secure a name for your app, the first step is to reserve it, which you can do up to three months before publishing, even if development has not started.

Steps to Reserve Your App's Name:

1. Navigate to the Partner Center apps and games page.
2. Click **New product**.
3. Click on **MSIX** or **PWA** app. If you want to submit an MSIX or PWA game, click on **Game**.
4. Enter the name you would like to use and click **Check availability**. If the name is available, you will see a green check mark. If the name is already in use, you will see a message indicating so.
5. Once you have selected an available name that you would like to reserve, click **Reserve product name.**

For more information, please visit Reserve your app's name - Windows apps | Microsoft Learn.

Create your app submission [MSIX/PWA]

1. After reserving an app name, you will be redirected to Application overview page. Click on **Start Submission**. A product submission in draft status will appear. This draft includes all the submission steps that need to be completed.
2. For submitting a PWA, learn how to convert your website into a PWA.
3. Complete each tab of the submission draft:
    - **Pricing and Availability**: Provide the price of your app, where it will be available to customers and decide whether you'll offer a free trial by setting pricing and availability for your app. Set app pricing and availability - Windows apps | Microsoft Learn

- **Properties**: Define your app's category, secondary category, sub-category and provide system requirements for your app, by entering your app properties. Enter app properties - Windows apps | Microsoft Learn
- **Age Ratings**: For your app to receive the appropriate age and content ratings administered by the IARC rating system, answer age rating questionnaire and generate the age rating. Age ratings for MSIX apps - Windows apps | Microsoft Learn
- **Packages**: Upload the app packages for your app which will be downloaded by the Store user. Upload app packages - Windows apps | Microsoft Learn
- **Store Listings**: Enter the app description, screenshots, and logos that will be displayed on your app's Product Details Page (PDP) in the Store, by creating a Store listing. Create app store listings - Windows apps | Microsoft Learn
- **Submission options (optional)**: Enter notes for certification or decide when your app should publish to the Store using submission options. Manage submission options - Windows apps | Microsoft Learn

4. After you complete a section, click on **Save**. Once all sections are complete and you are ready to submit, click on **Submit for certification**.

If your draft is missing any necessary information, it will be marked as **Incomplete.**

For more information, please visit Create an app submission - Windows apps | Microsoft Learn.

App certification process [MSIX/PWA]

When you finish creating your app's submission and click Submit to the Store, the submission enters the certification step. This process can take up to three business days. During certification, we will perform security tests, technical compliance tests and also a content compliance check on you app submission. You will be notified if your submission fails any of these tests.

After your submission passes certification, on an average, customers will be able to see the app's listing within 15 minutes depending on their location. You will be notified when your submission is published to the Store, and the app's status in the dashboard change to 'In Microsoft Store'.

To understand the certification process of MSIX apps, you can watch the following video.

https://learn-video.azurefd.net/vod/player?id=4f55d620-642f-4653-8fb4-1d2e49e3dff4&locale=en-us&embedUrl=%2Fwindows%2Fapps%2Fpublish%2F

For more information, please visit The app certification process - Windows apps | Microsoft Learn.

Post-certification [MSIX/PWA]

Analyze performance for MSIX apps and games

View detailed analytics for your apps and games in Partner Center. Statistics and charts let you know how your apps are performing in the Store; from how many customers you have reached to how they are using your app and what they have to say about it. You can also find metrics on app install, app health, app usage, and more. You can view analytic reports right in Partner Center or download the reports you need to analyze your data offline. We also provide several ways for you to access your analytics data outside of Partner Center.

To understand how to analyze your MSIX app's performance, you can watch the following video.

https://learn-video.azurefd.net/vod/player?id=fa8f1e24-e923-4ec1-b4b4-83956ef3ed51&locale=en-us&embedUrl=%2Fwindows%2Fapps%2Fpublish%2F

For more details, you can refer to the following reports.

Reports available for your apps

There are different reports available for your apps, check the descriptions to find the metrics you are looking for.

Expand table

Report	Description
Acquisitions report	See how many people have seen and installed your app in Store. You can also review data for different acquisition channels, markets, and platform details in this report.

Report	Description
Add-on acquisition report	See how many add-ons you have sold, along with demographic and platform details.
Usage report	See how customers on Windows 10 or Windows 11 (including Xbox) are using your app, including data about custom events that you have defined.
Health report	Get data related to the performance and quality of your app, including crashes and unresponsive events.
Reviews report	See the rating and reviews your customers have left for your app and provide responses to let customers know you are listening to their feedback.
Insights report	See meaningful insights about your app like significant (changes increases or decreases that we detected over the last 30 days in your acquisitions and health data).

Appendix [MSIX/PWA]

Useful links

- Microsoft Store Developer CLI documentation
- Store Policies and Code of Conduct

Troubleshooting

- Resolve submission errors
- Avoid common certification failures

Full app submission documentation

- If you are seeking more detailed documentation, click here.

Contact information for support

- If you need further assistance, a support ticket can be raised from here.

Windows App SDK

Article • 07/15/2024

The Windows App SDK is a set of new developer components and tools that represent the next evolution in the Windows app development platform. The Windows App SDK provides a unified set of APIs and tools that can be used in a consistent way by any desktop app on Windows 11 and downlevel to Windows 10, version 1809.

The Windows App SDK doesn't replace the Windows SDK or existing desktop Windows app types such as .NET (including Windows Forms and WPF) and desktop Win32 with C++. Instead, the Windows App SDK complements those existing tools and app types with a common set of APIs that developers can rely on across these platforms. For more details, see Benefits of the Windows App SDK.

> **Important**
>
> If you're working on a UWP app, then refer to **Migrate from UWP to the Windows App SDK**.

Get started with the Windows App SDK

The Windows App SDK provides extensions for Visual Studio 2022 and Visual Studio 2019. Those extensions include project templates configured to use the Windows App SDK components in new projects. The Windows App SDK libraries are also available via a NuGet package that you can install in existing projects.

To learn about configuring Visual Studio for creating new Windows App SDK projects, including the required workloads and components, see Getting Started with WinUI.

If you've already installed the required developer tools, then you're ready to Create your first WinUI 3 project. You can also use the Windows App SDK in an existing project.

For guidance on specific Windows App SDK versions, see Release channels and Downloads.

Windows App SDK features

The following table highlights the development features that are provided by the current releases of the Windows App SDK. For more details about the release channels

of the Windows App SDK that include each of these features, see Features available by release channel.

Expand table

Feature	Description
WinUI 3	The premiere native user interface (UI) framework for Windows desktop apps, including managed apps that use C# and .NET and native apps that use C++ with the Win32 API. WinUI 3 provides consistent, intuitive, and accessible experiences using the latest user interface (UI) patterns.
Render text with DWriteCore	Render text using a device-independent text layout system, high quality sub-pixel Microsoft ClearType text rendering, hardware-accelerated text, multi-format text, wide language support, and much more.
Manage resources with MRT Core	Manage app resources such as strings and images in multiple languages, scales, and contrast variants independently of your app's logic.
App lifecycle: App instancing	Control whether multiple instances of your app's process can run at the same time.
App lifecycle: Rich activation	Process information about different kinds activations for your app.
App lifecycle: Power management	Gain visibility into how your app affects the device's power state, and enable the app to make intelligent decisions about resource usage.
Manage app windows	Create and manage the windows associated with your app.
Push notifications	Send raw notifications and app notifications to your app from the cloud using Azure App Registration identities.
App notifications	Send app notifications to your app locally.
Deployment	Deploy Windows App SDK runtime with your unpackaged and packaged app

Windows App SDK release channels

The following table provides an overview of the different release channels.

Expand table

Release channel	Description
Stable	This channel is supported for use by apps in production environments. It only includes stable APIs. By default, the Windows App SDK docs focus on the Stable channel.
Preview	This channel provides a preview of the next stable release. There may be breaking API changes between a given preview channel release and the next stable release. For documentation on using the Preview release, see Install tools for preview and experimental channels of the Windows App SDK.
Experimental	This channel includes experimental features that are in early stages of development. Experimental features may be removed from the next release, or may never be released. For documentation on using the Experimental release, see Install tools for preview and experimental channels of the Windows App SDK.

For more details about the release channels of the Windows App SDK, see Windows App SDK release channels.

Benefits of the Windows App SDK for Windows developers

The Windows App SDK provides a broad set of Windows APIs with implementations that are decoupled from the OS and released to developers via NuGet packages. The Windows App SDK is not meant to replace the Windows SDK. The Windows SDK will continue to work as is, and there are many core components of Windows that will continue to evolve via APIs that are delivered via OS and Windows SDK releases. Developers are encouraged to adopt the Windows App SDK at their own pace.

Unified API surface across desktop app platforms

Developers who want to create desktop Windows apps must choose between several app platforms and frameworks. Although each platform provides many features and APIs that can be used by apps that are built using other platforms, some features and APIs can only be used by specific platforms. The Windows App SDK unifies access to Windows APIs for desktop Windows 11 and Windows 10 apps. No matter which app model you choose, you will have access to the same set of Windows APIs that are available in the Windows App SDK.

Over time, we plan to make further investments in the Windows App SDK that remove more distinctions between the different app models. For example, your app might use the WinUI Desktop model, it might use the XAML Islands model, it might use WPF, or it

could be a non-XAML app that uses Composition Islands or other aspects of the Windows App SDK. In any of these cases, we aim to make access to the Windows APIs in the Windows App SDK the same easy experience. The Windows App SDK will include both WinRT APIs and native C APIs.

Consistent experience across Windows versions

As the Windows APIs continue to evolve with new OS versions, developers must use techniques such as version adaptive code to account for all the differences in versions to reach their application audience. This can add complexity to the code and the development experience.

Windows App SDK APIs will work on Windows 11 and downlevel to Windows 10, version 1809. This means that as long as your customers are on Windows 10, version 1809, or any later version of Windows, you can use new Windows App SDK APIs and features as soon as they are released, and without having to write version adaptive code.

Faster release cadence

New Windows APIs and features are usually shipped within major Windows OS updates that release about once a year (often less frequently). Windows App SDK ships new releases about once every six months. This release cadence ensures that you continuously have access to the latest innovations in the Windows development platform.

Give feedback and contribute

We are building the Windows App SDK as an open source project. We have a lot more information on our Github page about how we're building the Windows App SDK, and how you can be a part of the development process. Check out our contributor guide to ask questions, start discussions, or make feature proposals. We want to make sure that the Windows App SDK brings the biggest benefits to developers like you.

Related topics

- Release channels and release notes
- Install tools for the Windows App SDK
- Create your first WinUI 3 project
- Use the Windows App SDK in an existing project
- Deploy apps that use the Windows App SDK

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Windows App SDK and supported Windows releases

Article • 10/23/2024

This topic identifies the versions of the Windows client and Windows Server operating systems (OS) supported by the Windows App SDK, along with the various servicing categories, channels, and SKUs for each OS.

This table summarizes the Windows client and server releases supported by the Windows App SDK (the following sections provide details on the various servicing categories, channels, and SKUs for each OS).

Expand table

Windows App SDK	Release date	Windows client	Windows Server
1.5	2024-02-29	Win10 1809 Win10 21H2 Win10 22H2 Win10 23H2 Win11 21H2 Win11 22H2 Win11 23H2 Win11 24H2	Server 2019 Server 2022
1.6	2024-09-04	Win10 1809 Win10 21H2 Win10 22H2 Win10 23H2 Win11 21H2 Win11 22H2 Win11 23H2 Win11 24H2	Server 2019 Server 2022

Windows client SKUs and channels

The Windows client OS runs on personal devices such as workstations, desktop PCs, laptops, and tablets.

There are four servicing categories that apply to various combinations of Windows client SKUs and servicing channels.

Expand table

Servicing channel		SKU	SKU
		Home, Pro	Enterprise
GA*		Shortest servicing	Short servicing
LTSC**	Mainstream	N/A	Long servicing
	Extended	N/A	Longest servicing

* The General Availability (GA) servicing channel includes the majority of Windows devices.

** The Long Term Servicing Channel (LTSC) includes devices that need to continue running older versions of Windows.

Five older versions of Windows client are still in service for at least one servicing category. Each version will continue to be supported by the Windows App SDK until the latest applicable Windows servicing end-date for that version. The following table shows the end-date for each servicing category.

Expand table

Version	Build	Servicing end-dates				Max end-dates
		GA Home, Pro	GA Enterprise	LTSC Mainstream	LTSC Extended	
Win10 1809	17763	Past	Past	Past	2029-01-09	2029-01-09
Win10 21H2	19044	Past	2024-06-11	2027-01-12	2032-01-13	2032-01-13
Win10 22H2	19045	2025-10-14	2025-10-14	N/A	N/A	2025-10-14
Win11 21H2	22000	Past	2024-10-08	N/A	N/A	2024-10-08
Win11 22H2	22621	2024-10-08	2025-10-14	N/A	N/A	2025-10-14
Win11 23H2	22631	2025-11-11	2026-11-10	N/A	N/A	2026-11-10
Win11	26100	2026-10-13	2027-10-12	2029-10-09	2034-10-10	2034-10-10

24H2

Windows Server SKUs and channels

Windows Server is the platform for building an infrastructure of connected applications, networks, and web services, from the workgroup to the data center. Windows Server offers services to other devices, has more features and capacities than a client OS, and can support more network connections.

The following table shows the end-date for each servicing category.

Expand table

Version	Build	Servicing end-dates		Max end-dates
		LTSC Mainstream	LTSC Extended	
Server 2019	17763	Past	2029-01-09	2029-01-09
Server 2022	20348	2026-10-13	2031-10-14	2032-01-13

Related articles

- Check for installed versions of the Windows App SDK runtime

Windows App SDK release channels

Article • 01/15/2025

The latest version of the Windows App SDK ships via three release channels: Experimental, Preview, and Stable. The following table provides an overview of these release channels.

> **Note**
>
> When you Install tools for the Windows App SDK, install the release channel that best serves your development scenario.

Expand table

	Channel	Description	Release cadence	Supported?	Latest release
	Preview (release notes)	This channel provides a preview of the next stable release. There may be breaking API changes between a given preview channel release and the next stable release.	At least one preview per stable version	No	1.7.0-preview1 (02/11/2025)
	Stable (release notes)	This channel is supported for use by apps in production environments. It includes only stable APIs.	No more frequent than every six months (+ servicing)	Yes	1.7.0 (1.7.250310001) (03/18/2025)
	Experimental (release notes)	This channel includes experimental features that are in early stages of development. Experimental features may be removed from the next release, or may never be released.	As needed when requiring feedback for features in early design or prototype stages	No	1.7.0-experimental3 (02/06/2025)

For a comprehensive list of all current and previous releases of the Windows App SDK, including download locations, see Downloads for the Windows App SDK.

Features available by release channel

The following table shows which features are currently available in each release channel.

Expand table

Feature	Stable	Preview	Experimental
Deployment guide for framework-dependent packaged apps	Available	Available	Available
Deployment guide for framework-dependent apps packaged with external location or unpackaged	Available	Available	Available
WinUI 3	Available	Available	Available
Text rendering	Available	Available	Available
Manage resources	Available	Available	Available
App lifecycle: App instancing	Available	Available	Available
App lifecycle: Rich activation	Available	Available	Available
App lifecycle: Power management	Available	Available	Available
Manage app windows	Available	Available	Available
Push notifications	Available	Available	Available
App notifications	Available	Available	Available
Windows Widgets	Available	Available	Available

Windows App SDK release servicing policy

The following policies define the servicing you can expect when you use a given Windows App SDK release.

Release lifecycle

The Windows App SDK has a lifecycle. A lifecycle begins when a version or service is released and ends when it's no longer supported. Knowing key dates in this release lifecycle helps you make informed decisions about when to upgrade or make other changes to your software.

Expand table

Windows App SDK version	Original release date	Latest patch version	Patch release date	Support level	End of servicing
1.7	03/18/2025	1.7.250310001	03/18/2025	Current	03/18/2026
1.6	09/04/2024	1.6.250205002	02/11/2025	Maintenance	09/04/2025
1.5	02/29/2024	1.5.250108004	01/15/2025	Out of Support	02/28/2025
1.4	08/29/2023	1.4.240802001	08/13/2024	Out of Support	08/29/2024
1.3	04/12/2023	1.3.230724000	07/25/2023	Out of Support	04/12/2024
1.2	11/10/2022	1.2.230313.1	03/15/2023	Out of Support	11/10/2023
1.1	05/24/2022	1.1.5	09/14/2022	Out of Support	05/24/2023
1.0	11/16/2021	1.0.4	06/14/2022	Out of Support	11/16/2022
0.8	6/24/2021	0.8.12	08/03/2022	Out of Support	6/24/2022
0.5	3/29/2021	0.5.9	8/10/2021	Out of Support	11/1/2021

Servicing

Customers can choose *Current* releases or *Maintenance* releases. Both types of releases receive critical fixes throughout their lifecycle, for security, compatibility, and reliability. You must stay up to date with the latest patches to qualify for support.

The quality of all releases is the same. The only difference is the servicing time frame.

Servicing for Current releases

Current releases are the latest stable release and they receive fixes more frequently. During the servicing period, Windows App SDK current release is updated to improve functional capabilities and mitigate security vulnerabilities.

Functional improvements are typically very targeted, and may address the following:

- Resolve reported crashes.
- Resolve severe performance issues.
- Resolve functional bugs in key scenarios.
- Add support for a new operating system versions

Servicing for Maintenance releases

Maintenance releases are no longer the latest stable release, but they will still receive critical fixes. The bar for fixes in Maintenance releases will be higher than the bar for fixes to the Current release.

After the maintenance period ends, the release is out of support.

Microsoft support for the Windows App SDK

The Windows App SDK is governed by the Microsoft Modern Lifecycle.

Although the Windows App SDK is currently backward compatible to Windows 10 version 1809, the Windows App SDK is supported by Microsoft only on the Windows releases still in support.

Support has two key benefits:

- Patches are provided for free, as required for functional or security issues.
- You can contact Microsoft support to request help, potentially at a cost.

Support is conditional on using the latest Windows App SDK patch update and a supported operating system.

End of support

End of support refers to the date when Microsoft no longer provides fixes, updates, or online technical assistance. End of support may also be referred to as 'end of life' or abbreviated 'EOL'. This is the time to make sure you have the latest available update installed.

Updates are cumulative, with each update built upon all of the updates that preceded it. A device needs to install the latest update to remain supported. Updates may include new features, fixes (security and/or non-security), or a combination of both. Not all features in an update will work on all devices. Update availability may vary, for example by country, region, network connectivity, or hardware capabilities (including, for example, free disk space).

Your use of out-of-support Windows App SDK versions may put your applications at risk. You are strongly recommended to not use out-of-support software.

How to get help

- The Windows App SDK uses GitHub Issues to track bugs and feature requests. Search the existing issues before filing new issues to avoid duplicates. For new issues, file your bug or feature request as a new issue.
- For help and questions about using the Windows App SDK, search for existing questions or post a new question on our GitHub Discussions page.
- Technical support for the use of the Windows App SDK may be available from Microsoft Customer Support Services (CSS). If you're a Premier or Unified Support customer, reach out to your account manager for further assistance. Otherwise, visit the Support For Business site to open a new support case for the Windows App SDK.

Related topics

- Stable channel
- Preview channel
- Experimental channel
- Install tools for preview and experimental channels of the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 project

Latest stable channel release notes for the Windows App SDK

Article • 03/19/2025

The stable channel provides releases of the Windows App SDK that are supported for use by apps in production environments. Apps that use the stable release of the Windows App SDK can also be published to the Microsoft Store.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.

Stable channel release note archive:

- Stable channel release notes for the Windows App SDK 1.6
- Stable channel release notes for the Windows App SDK 1.5
- Stable channel release notes for the Windows App SDK 1.4
- Stable channel release notes for the Windows App SDK 1.3
- Stable channel release notes for the Windows App SDK 1.2
- Stable channel release notes for the Windows App SDK 1.1
- Stable channel release notes for the Windows App SDK 1.0
- Stable channel release notes for the Windows App SDK 0.8
- Stable channel release notes for the Windows App SDK 0.5

Downloads for the Windows App SDK

> **Note**
>
> The Windows App SDK Visual Studio Extensions (VSIX) are no longer distributed as a separate download. They are available in the Visual Studio Marketplace inside Visual Studio.

Version 1.7

The following sections describe new and updated features and known issues for version 1.7.

In an existing Windows App SDK app, you can update your Nuget package to 1.7.250310001 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Downloads for the Windows App SDK.

Version 1.7.0 (1.7.250310001)

New Badge Notifications Feature

The notification badge conveys a summary or status information specific to an app. This can be numeric (1-99) or a glyph from one of the system-provided glyphs. This new functionality provides an easy way for apps to show status, such as number of unread mails in a mail app or number of new posts in a social media app.

For more info, see GitHub #4926.

New CameraCaptureUI API

Developers have encountered challenges in the desktop environments due to WinRT CameraCaptureUI being dependent on CoreWindows, and lack of InitializeWithWindow support. The team has released this new `Microsoft.Windows.Media.Capture.CameraCaptureUI` API to WinAppSDK to provide a streamlined solution with feature parity, now supporting WindowID in the constructor for enhanced desktop compatibility.

For more info, see GitHub issue #4721.

New Authentication API

A new `OAuth2Manager` API provides a streamlined solution for web authentication, offering OAuth 2.0 capabilities with full feature parity across all Windows platforms supported by Windows App SDK. This new Authentication Manager is different from the public WebAuthentication Broker API, as it better aligns with OAuth best practices.

For more info, see GitHub issue #4772.

New Background Task support

Background tasks are app components that run in the background without a user interface, performing actions like download files, syncing data, sending notifications or

updating files. The new `BackgroundTaskBuilder` API provides WinAppSDK dependent apps the ability to directly register the full trust COM components with background tasks, removing the need to implement a workaround.

For more info, see GitHub #4831.

New TitleBar control

A new `TitleBar` control makes it much easier to create a great, customizable titlebar for your app. Configure properties such as the titlebar icon, Title, and Subtitle, include an integrated back button, or even add a custom control like a search box! The control includes robust titlebar capabilities like empty-space draggable regions, theme responsiveness, caption buttons, and built-in accessibility support so you can focus on your personalized design and still get the same reliable titlebar as the default experience.

For more info, see GitHub #10056.

Support for MathML

`RichEditBox` now supports MathML, via `RichEditTextDocument.SetMathMode` and `RichEditTextDocument.SetMathML`.

For more info, see GitHub #4196.

Enhanced Runtime

- Windows App SDK's Dynamic Dependencies APIs delegate all calls to Windows 11's implementation when running on >= Windows 11 24H2 (10.0.26100.0) providing improved performance and robustness. This holds true for all C/C++ (Mdd*()) and WinRT (namespace Microsoft.Windows.ApplicationModel.DynamicDependency) APIs.
  - Packaged processes calling Windows App SDK's Dynamic Dependencies APIs is now supported on >= Windows 11 24H2 (10.0.26100.0). This is still unsupported on older systems (WinAppSDK's implementation doesn't support packaged apps).
  - This has no impact to the developer experience. Callers can continue using the Bootstrapper API to add the WinAppSDK framework package to the calling process' package graph.
  - For more info, see GitHub PR #4949.

- Undocked Registration-free WinRT (URFW) is not enabled on >= Windows 11 24H2 (10.0.26100.0). The OS' implementation handles all Registration-free WinRT activity on these systems providing improved performance and robustness. For more info, see GitHub PR #4949.
- Detours is not used on >= Windows 11 24H2 (10.0.26100.0). Detours was only used by Windows App SDK's implementations of Dynamic Dependencies and Registration-free WinRT, but as those features are now handled by the OS' implementations there's no need for them to initialize or otherwise wire up Detours. This provides a small performance gain when loading Microsoft.WindowsAppRuntime.dll. For more info, see GitHub PR #4949.

New AppWindow APIs

New `AppWindow` APIs make it easier to control your app windows to create a great experience. New capabilities include using `SetTaskBarIcon` and `SetTitleBarIcon` to independently set the taskbar and titlebar icons, using `AppWindowTitleBar.PreferredTheme` to set the light/dark theme of the titlebar, and using new properties like `OverlappedPresenter.PreferredMinimumWidth` and `OverlappedPresenter.PreferredMaximumHeight` to set a minimum or maximum width or height for the window.

New Island APIs

The updates in the Microsoft.UI.Content namespace introduce several significant enhancements and new features aimed at improving the functionality and interoperability of the ContentIsland APIs. These changes are designed to support new hosting scenarios, enhance rendering capabilities, and ensure better synchronization of input and accessibility states. Key updates include:

1. New primitives for hosting ContentIslands:
    - `DesktopPopupSiteBridge`: Enables hosting a `ContentIsland` in the environment of a Win32 window with WS_POPUP style, facilitating scenarios where applications use popup windows for dialog boxes and message boxes.
    - `ChildSiteLink`: Allows a parent `ContentIsland` to host a nested child `ContentIsland`, providing a seamless partitioning of the rendering surface without user experience seams.
    - `DesktopAttachedSiteBridge`: Attaches to an existing Win32 window instead of creating a new one, designed to host a `ContentIsland` with

Windows.UI.Composition.Visuals at the root of the Win32 window hierarchy, ensuring full control over Win32-based input processing and accessibility.

2. Enhanced rendering and input synchronization:

    - The `LocalToParentTransformMatrix` and `ActualSize` properties of a `ChildSiteLink` are updated relative to the parent `ContentIsland` before rendering, avoiding latency and ensuring synchronized input and accessibility states.

3. ContentIslands with Windows.UI.Composition.Visuals:

    - `ContentIsland` can use Windows.UI.Composition.Visuals for rendering and Win32 window APIs for input processing, enabling interoperability with applications that use legacy UX frameworks. This allows for a gradual adoption of newer UX frameworks layered on top of the Windows App SDK Scene Graph, such as WinUI and React Native for Windows on Fabric. These updates collectively enhance the flexibility, performance, and interoperability of the ContentIsland APIs, enabling developers to create more sophisticated and responsive applications.

Additionally, the updates in the Microsoft.UI.Xaml namespace introduce a new `XamlIsland` API, which allows for the hosting of Xaml content within a SiteBridge or a `ChildSiteLink`. The `XamlIsland` offers greater flexibility compared to the `DesktopWindowXamlSource` API. While `DesktopWindowXamlSource` requires hosting within an existing Win32 window, the `XamlIsland` exposes a `ContentIsland`, enabling more options for hosting Xaml content.

Other notable changes

- New `RuntimeCompatibilityOptions` support will allow more control over how servicing changes affect apps. For more info, see GitHub #4966.
- A new `ReleaseInfo` API provides easy access to the version of the Windows App SDK Runtime in use. For more info, see GitHub #2893.
- Note: Windows Copilot Runtime APIs are not included this release. To experiment with these APIs, please continue to use the 1.7-experimental3 release and share your feedback!

New APIs for 1.7.0

This release includes the following new APIs compared to the stable 1.6 release:

```
```

```
Microsoft.Security.Authentication.OAuth

    AuthFailure
    AuthRequestParams
    AuthRequestResult
    AuthResponse
    ClientAuthentication
    CodeChallengeMethodKind
    OAuth2Manager
    TokenFailure
    TokenFailureKind
    TokenRequestParams
    TokenRequestResult
    TokenResponse
```

```
Microsoft.UI.Content

    ChildSiteLink
    ContentAutomationOptions
    ContentEnvironmentStateChangedEventArgs
        DidDisplayScaleChange

    ContentIsland
        AutomationOption
        Children
        CreateForSystemVisual
        FindAllForSystemCompositor
        FragmentRootAutomationProvider
        GetBySystemVisual
        LocalToClientTransformMatrix
        LocalToParentTransformMatrix
        NextSiblingAutomationProvider
        ParentAutomationProvider
        Popups
        PreviousSiblingAutomationProvider
        ProcessesKeyboardInput
        ProcessesPointerInput

    ContentIslandEnvironment
        DisplayScale

    ContentIslandStateChangedEventArgs
        DidLocalToClientTransformMatrixChange
        DidLocalToParentTransformMatrixChange

    ContentSite
        LocalToClientTransformMatrix
        LocalToParentTransformMatrix
        ProcessesKeyboardInput
        ProcessesPointerInput
```

```
    ContentSiteAutomationProviderRequestedEventArgs
    ContentSiteEnvironment
        DisplayScale

    ContentSiteEnvironmentView
        DisplayScale

    ContentSiteView
        AutomationOption
        LocalToClientTransformMatrix
        LocalToParentTransformMatrix
        ProcessesKeyboardInput
        ProcessesPointerInput

    DesktopAttachedSiteBridge
    DesktopChildSiteBridge
        CreateWithDispatcherQueue

    DesktopPopupSiteBridge
    IContentSiteAutomation
    IContentSiteInput
    IContentSiteLink
```

```
Microsoft.UI.Input

    InputFocusNavigationHost
        GetForSiteLink
```

```
Microsoft.UI.Text

    RichEditTextDocument
        GetMathML
        GetMathMode
        SetMathML
        SetMathMode
```

```
Microsoft.UI.Windowing

    AppWindow
        SetTaskbarIcon
        SetTaskbarIcon
        SetTitleBarIcon
        SetTitleBarIcon
```

```
    AppWindowTitleBar
        PreferredTheme

    OverlappedPresenter
        PreferredMaximumHeight
        PreferredMaximumWidth
        PreferredMinimumHeight
        PreferredMinimumWidth

    TitleBarTheme
```

```
Microsoft.UI.Xaml

    XamlIsland
    XamlRoot
        ContentIsland
```

```
Microsoft.UI.Xaml.Controls

    TitleBar
    TitleBarAutomationPeer
    TitleBarTemplateSettings
```

```
Microsoft.Windows.ApplicationModel.Background

    BackgroundTaskBuilder
```

```
Microsoft.Windows.ApplicationModel.Background.UniversalBGTask

    Task
```

```
Microsoft.Windows.ApplicationModel.WindowsAppRuntime

    ReleaseInfo
    RuntimeCompatibilityChange
    RuntimeCompatibilityOptions
```

```
    RuntimeInfo
    WindowsAppRuntimeVersion
```

```
Microsoft.Windows.BadgeNotifications

    BadgeNotificationGlyph
    BadgeNotificationManager
```

```
Microsoft.Windows.Media.Capture

    CameraCaptureUI
    CameraCaptureUIMaxPhotoResolution
    CameraCaptureUIMaxVideoResolution
    CameraCaptureUIMode
    CameraCaptureUIPhotoCaptureSettings
    CameraCaptureUIPhotoFormat
    CameraCaptureUIVideoCaptureSettings
    CameraCaptureUIVideoFormat
```

Bug fixes

This release includes the following bug fixes:

- Changed `SplitButton` so touch input now matches the behavior of mouse input. For more info, see GitHub issue #178.
- Changed cascading menus so sub menus now open immediately if clicked. For more info, see GitHub issue #939.
- Fixed an issue where opening a `ComboBox` which is in a flyout closes all flyouts. For more info, see GitHub issue #1467.
- Fixed an issue where `SwipeControl` would randomly crash in a `ListView`. For more info, see GitHub issue #2527.
- Fixed an issue where drag-and-drop only a `ListViewItem` would leave it in the wrong visual state. For more info, see GitHub issue #3458.
- Fixed an issue in `StackLayout` so that it respects the ItemsRepeater.HorizontalAlignment and ItemsRepeater.VerticalAlignment properties (when StackLayout.Orientation is Vertical and Horizontal respectively). The old layout behaved as if the ItemsRepeater alignment was Stretch. With the fix, the layout results in items aligned to the right when the Right alignment is used, for example. For more info, see GitHub issue #3842.

- Fixed a potential crash when using a resource which contains an `x:Bind`. For more info, see GitHub issue #5786.
- Fixed an issue where deleting items in the `ItemsRepeater`'s source would not generate items which moved up into view. For more info, see GitHub issue #6661.
- Fixed an issue where the right Alt key would not show keytips for Access Keys. For more info, see GitHub issue #8447. **Note:** This may result in key events for the right Alt key no longer being delivered to handles in the app or controls.
- Fixed an issue where using a ResourceDictionary containing only a single resource would fail to find that resource and likely cause a crash. For more info, see GitHub issue #8832.
- Fixed a crash where `UniformGridLayout` would sometimes pick a wrong layout anchor and cause infinite layout passes when scrolling backwards. For more info, see GitHub issue #9199.
- Fixed an issue where setting `NavigationFailedEventArgs.Handled` to True would still throw an exception. For more info, see GitHub issue #9632.
- Fixed an issue where `TabView` would not apply any specified `CornerRadius`. For more info, see GitHub issue #9846.
- Fixed a potential layout cycle crash in `StackLayout`. For more info, see GitHub issue #9852.
- Fixed a potential crash in `ItemsView` when removing items. For more info, see GitHub issue #9868.
- Fixed an issue in 1.7-preview1 where popups no longer correctly moved with their parent window. For more info, see GitHub issue #10386.
- Based on feedback from 1.7-preview1, renamed some properties on the new `TitleBar` control.

Related topics

- Preview channel
- Experimental channel
- Install tools for the Windows App SDK
- Create your first WinUI 3 project
- Use the Windows App SDK in an existing project
- Deploy apps that use the Windows App SDK

Latest preview channel release notes for the Windows App SDK

Article • 02/12/2025

> **ⓘ Important**
>
> The preview channel is **not supported** for use in production environments, and apps that use the preview releases cannot be published to the Microsoft Store.

The preview channel includes releases of the Windows App SDK with preview channel features in late stages of development. Preview releases do not include experimental features and APIs but may still be subject to breaking changes before the next stable release.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on preview releases, see Install tools for preview and experimental channels of the Windows App SDK.

Preview channel release note archive:

- Preview channel release notes for the Windows App SDK 1.6
- Preview channel release notes for the Windows App SDK 1.5
- Preview channel release notes for the Windows App SDK 1.4
- Preview channel release notes for the Windows App SDK 1.3
- Preview channel release notes for the Windows App SDK 1.2
- Preview channel release notes for the Windows App SDK 1.1
- Preview channel release notes for the Windows App SDK 1.0

Version 1.7 Preview 1 (1.7-preview1)

This is the latest release of the preview channel for version 1.7.

In an existing Windows App SDK 1.6 (from the stable channel) app, you can update your Nuget package to 1.7.0-preview1 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Downloads for the Windows App SDK.

New features

New Badge Notifications Feature

New badge notification support allows showing a number or glyph badge on your app in the taskbar. For more info, see GitHub #4926.

New CameraCaptureUI API

A new CameraCaptureUI API makes it easier to capture photos and videos in your Windows App SDK app. For more info, see GitHub issue #4721.

New Authentication API

A new `OAuth2Manager` API provides a streamlined solution for web authentication, offering OAuth 2.0 capabilities with full feature parity across all Windows platforms supported by Windows App SDK. For more info, see GitHub issue #4772.

New Background Task support

A new `BackgroundTaskBuilder` API enables registering background tasks for Windows App SDK apps. For more info, see GitHub #4831.

New TitleBar control

A new `TitleBar` control makes it much easier to create a great, customizable titlebar for your app. Configure properties such as the titlebar icon, Title, and Subtitle, include an integrated back button, or even add a custom control like a search box! The control includes robust titlebar capabilities like empty-space draggable regions, theme responsiveness, caption buttons, and built-in accessibility support so you can focus on your personalized design and still get the same reliable titlebar as the default experience. For more info, see GitHub #10056.

Support for MathML

`RichEditBox` now supports MathML, via `RichEditTextDocument.SetMathMode` and `RichEditTextDocument.SetMathML`. For more info, see GitHub #4196.

Other notable changes

- New `RuntimeCompatibilityOptions` support will allow more control over how servicing changes affect apps. For more info, see GitHub #4966.
- A new `ReleaseInfo` API provides easy access to the version of the Windows App SDK Runtime in use. For more info, see GitHub #2893.
- Note: Windows Copilot Runtime APIs are not included this release. To experiment with these APIs, please continue to use the 1.7-experimental3 release and share your feedback!
- Note: New APIs for windowing on `AppWindow` are not included in this release. To experiment with these APIs, please continue to use the 1.7-experimental3 release and share your feedback!

New APIs

This release includes the following new APIs compared to the stable 1.6 release:

```
Microsoft.Security.Authentication.OAuth

    AuthFailure
    AuthRequestParams
    AuthRequestResult
    AuthResponse
    ClientAuthentication
    CodeChallengeMethodKind
    OAuth2Manager
    OAuthContract
    TokenFailure
    TokenFailureKind
    TokenRequestParams
    TokenRequestResult
    TokenResponse
```

```
Microsoft.UI.Text

    RichEditTextDocument
        GetMathML
        GetMathMode
        SetMathML
        SetMathMode

    TextApiContract
```

```
Microsoft.UI.Xaml

    XamlIsland
    XamlRoot
        ContentIsland
```

```
Microsoft.UI.Xaml.Controls

    TitleBar
    TitleBarAutomationPeer
    TitleBarTemplateSettings
```

```
Microsoft.Windows.ApplicationModel.Background

    BackgroundTaskBuilder
    BackgroundTaskContract
```

```
Microsoft.Windows.ApplicationModel.Background.UniversalBGTask

    Task
    UniversalBackgroundTaskContract
```

```
Microsoft.Windows.ApplicationModel.WindowsAppRuntime

    ReleaseInfo
    RuntimeCompatibilityChange
    RuntimeCompatibilityContract
    RuntimeCompatibilityOptions
    RuntimeInfo
    VersionInfoContract
    WindowsAppRuntimeVersion
```

```
Microsoft.Windows.BadgeNotifications

    BadgeNotificationGlyph
```

```
    BadgeNotificationManager
    BadgeNotificationsContract
```

```
Microsoft.Windows.Media.Capture

    CameraCaptureUI
    CameraCaptureUIContract
    CameraCaptureUIMaxPhotoResolution
    CameraCaptureUIMaxVideoResolution
    CameraCaptureUIMode
    CameraCaptureUIPhotoCaptureSettings
    CameraCaptureUIPhotoFormat
    CameraCaptureUIVideoCaptureSettings
    CameraCaptureUIVideoFormat
```

Bug fixes

This release includes the following bug fixes:

- Changed `SplitButton` so touch input now matches the behavior of mouse input. For more info, see GitHub issue #178.
- Changed cascading menus so sub menus now open immediately if clicked. For more info, see GitHub issue #939.
- Fixed an issue where opening a `ComboBox` which is in a flyout closes all flyouts. For more info, see GitHub issue #1467.
- Fixed an issue where `SwipeControl` would randomly crash in a `ListView`. For more info, see GitHub issue #2527.
- Fixed an issue where drag-and-drop only a `ListViewItem` would leave it in the wrong visual state. For more info, see GitHub issue #3458.
- Fixed an issue in `StackLayout` so that it respects the ItemsRepeater.HorizontalAlignment and ItemsRepeater.VerticalAlignment properties (when StackLayout.Orientation is Vertical and Horizontal respectively). The old layout behaved as if the ItemsRepeater alignment was Stretch. With the fix, the layout results in items aligned to the right when the Right alignment is used, for example. For more info, see GitHub issue #3842.
- Fixed an issue where deleting items in the `ItemsRepeater`'s source would not generate items which moved up into view. For more info, see GitHub issue #6661.
- Fixed an issue where the right Alt key would not show keytips for Access Keys. For more info, see GitHub issue #8447. **Note:** This may result in key events for the right Alt key no longer being delivered to handles in the app or controls.

- Fixed a crash where `UniformGridLayout` would sometimes pick a wrong layout anchor and cause infinite layout passes when scrolling backwards. For more info, see GitHub issue #9199.
- Fixed an issue where setting `NavigationFailedEventArgs.Handled` to True would still throw an exception. For more info, see GitHub issue #9632.
- Fixed an issue where `TabView` would not apply any specified `CornerRadius`. For more info, see GitHub issue #9846.
- Fixed a potential layout cycle crash in `StackLayout`. For more info, see GitHub issue #9852.
- Fixed a potential crash in `ItemsView` when removing items. For more info, see GitHub issue #9868.

Related topics

- Stable channel
- Experimental channel
- Install tools for the Windows App SDK
- Create your first WinUI 3 project
- Use the Windows App SDK in an existing project
- Deploy apps that use the Windows App SDK

Latest experimental channel release notes for the Windows App SDK

Article • 02/06/2025

> **ⓘ Important**
>
> The experimental channel is **not supported** for use in production environments, and apps that use the experimental releases cannot be published to the Microsoft Store.

The experimental channel includes releases of the Windows App SDK with experimental channel features in early stages of development. APIs for experimental features have the Experimental attribute. If you call an experimental API in your code, you will receive a build-time warning. All APIs in the experimental channel are subject to extensive revisions and breaking changes. Experimental features and APIs may be removed from subsequent releases at any time.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on experimental releases, see Install tools for preview and experimental channels of the Windows App SDK.

Experimental channel release note archive:

- Experimental channel release notes for the Windows App SDK 1.6
- Experimental channel release notes for the Windows App SDK 1.5
- Experimental channel release notes for the Windows App SDK 1.4
- Experimental channel release notes for the Windows App SDK 1.3
- Experimental channel release notes for the Windows App SDK 1.2
- Experimental channel release notes for the Windows App SDK 1.0
- Experimental channel release notes for the Windows App SDK 0.8

Version 1.7 Experimental (1.7.0-experimental3)

Use on-device AI with Windows Copilot Runtime APIs

> **ⓘ Important**
>
> The underlying ML models required for these APIs currently require your device to be running the latest Windows 11 Insider Preview Build on the Dev Channel. Additionally, these APIs require your device to be a Copilot+ PC. See **Copilot+ PCs Developer Guide** to learn more about these devices. APIs will throw an exception when called on devices lacking the necessary support.

The Windows Copilot Runtime (WCR) offers several AI-powered features and APIs for you to easily, efficiently, and responsibly use on-device AI models in your Windows apps. In this release we are making available several scenario focused APIs for you to leverage powerful capabilities without the need to find, run, or optimize your own Machine Learning (ML) models.

Learn more about responsible development practices used during WCR API development that you can also apply as you create AI-assisted features in the Developing Responsible Generative AI Applications and Features on Windows guidance.

Phi Silica

With **Phi Silica**, Microsoft's most powerful NPU-tuned local language model, you can generate text responses to broad user prompts with built in content moderation. You can also specify it to perform common tasks like summarizing a piece of text, rewriting a piece of text for clarity, and converting text to a table format. Phi Silica is optimized for efficiency and performance on Windows Copilot+ PCs devices while still offering many of the capabilities found in Large Language Models (LLMs).

See Get started with Phi Silica in the Windows App SDK and API ref for Phi Silica in the Windows App SDK for more information.

Text Recognition (OCR)

Text recognition, also known as optical character recognition (OCR), APIs in Windows Copilot Runtime (WCR) can detect and extract text within images and convert it into machine readable character streams. These APIs can identify characters, words, lines, polygonal text boundaries, and provide confidence levels for each match. The set of WCR AI-assisted APIs benefit from NPU-assisted acceleration to perform faster and more accurately than the legacy Windows.Media.Ocr.OcrEngine APIs.

See Get Started with Text Recognition (OCR) in the Windows App SDK and API ref for AI-backed Text Recognition (OCR) in the Windows App SDK for more information.

Image Super Resolution

Using the 'ImageScaler' APIs you can increase the sharpness and clarity of an image and upscale the image up to 8x its original resolution.

See What can I do with Image Super Resolution? and API ref for AI imaging features in the Windows App SDK to get started.

Image Description

The Image Description APIs can be used to generate a text description of an image. The APIs are configurable to specify the length and type of the text description. Image descriptions may include a short caption or a long description for users with accessibility needs.

> **Note**
>
> When calling ImageDescriptionGenerator.DescribeAsync() in a Debug build, an error may occur that can be mitigated by continuing the build in Visual Studio.

See What can I do with Image Description? and API ref for AI imaging features in the Windows App SDK to learn more.

Image Segmentation

Using Image Segmentation APIs you can identify specific objects within an image. The model takes both an image and a "hints" object and returns a mask of the identified object.

See What can I do with Image Segmentation? and API ref for AI imaging features in the Windows App SDK to get started.

New APIs for Windowing

New `AppWindow` APIs make it easier to control your app windows and create a great experience. New capabilities include using `EnablePlacementPersistence` to automatically remember the size and position of your windows, using `SetTaskBarIcon` and `SetTitleBarIcon` to independently set the taskbar and titlebar icons, using `AppWindowTitleBar.PreferredTheme` to set the light/dark theme of the titlebar, and using `OverlappedPresenter.PreferredMinimum/MaximumSize` to set a minimum or maximum size for the window.

Other notable changes

- The missing C# projections for the new `BadgeNotifications` have been added so these APIs are now usable from C#.
- A class registration issue which prevented using the new `AppNotificationConferencingConfig` API has been fixed. Note that this enhanced user experience for video or audio calling in notifications is only available on the latest Windows Insider releases of Windows.

New APIs

This release includes the following new and modified experimental APIs compared to 1.7-experimental2:

```
Microsoft.Graphics.Imaging

    ImageBuffer
    ImageBufferContract
    ImageObjectExtractor
    ImageObjectExtractorContract
    ImageObjectExtractorHint
    ImageScaler
    ImageScalerContract
    PixelFormat
```

```
Microsoft.UI.Content

    ChildSiteLink
        AutomationOption
        ProcessKeyboardInput
        ProcessPointerInput

    ContentAutomationOptions
    ContentIsland
        AutomationOption
        ProcessKeyboardInput
        ProcessPointerInput

    ContentSite
        ProcessKeyboardInput
        ProcessPointerInput

    ContentSiteView
        AutomationOption
```

```
        ProcessKeyboardInput
        ProcessPointerInput

    DesktopPopupSiteBridge
        AutomationOption

    IContentSiteAutomation
        AutomationOption

    IContentSiteInput
    ReadOnlyDesktopSiteBridge
        ProcessKeyboardInput
        ProcessPointerInput
```

```
Microsoft.UI.Windowing

    AppWindow
        EnablePlacementPersistence
        EnablePlacementPersistence
        GetCurrentPlacement
        SaveCurrentPlacement
        SetPlacement
        SetTaskBarIcon
        SetTaskBarIcon
        SetTitleBarIcon
        SetTitleBarIcon

    AppWindowTitleBar
        PreferredTheme

    OverlappedPresenter
        PreferredMaximumSize
        PreferredMinimumSize
        SetPreferredBounds

    PlacementPersistenceBehaviorFlags
    TitleBarTheme
```

```
Microsoft.Windows.AI.ContentModeration

    ContentFilterOptions
    ContentFilterOptionsContract
    ImageContentFilterSeverity
    SeverityLevel
    TextContentFilterSeverity
```

```
Microsoft.Windows.AI.Generative

    ImageDescriptionContract
    ImageDescriptionGenerator
    ImageDescriptionScenario
    LanguageModel
    LanguageModelContext
    LanguageModelContract
    LanguageModelOptions
    LanguageModelResponse
    LanguageModelResponseStatus
    LanguageModelSkill
```

```
Microsoft.Windows.SemanticSearch

    EmbeddingVector
    SemanticSearchContract
```

```
Microsoft.Windows.Vision

    BoundingBox
    DetectedLineStyle
    OrientationDetectionOptions
    RecognizedLine
    RecognizedLineStyle
    RecognizedText
    RecognizedWord
    TextRecognitionContract
    TextRecognizer
    TextRecognizerOptions
```

```
Microsoft.Windows.Workloads

    WorkloadPriority
    WorkloadsContract
```

Version 1.7 Experimental (1.7.0-experimental2)

Background Task Registration

A new `BackgroundTaskBuilder` API enables registering background tasks for Windows App SDK apps. For more info, see GitHub #4831.

New Notifications Features

New badge notification support allows showing a number or glyph badge on your app in the taskbar. For more info, see GitHub #4926.

> **Important**
>
> In this release, the C# projections are missing for the new `BadgeNotifications` APIs, which prevents using them from C#. The APIs are available in C++.

Video or audio calling can have an enhanced user experience in notifications. For more info, see GitHub #4783.

> **Important**
>
> This functionality is only available on the latest Windows Insider releases of Windows.

Other notable changes

- `RichEditBox` now supports math mode, via `RichEditTextDocument.SetMathMode` and `RichEditTextDocument.SetMath`.
- New `CompatibilityOptions` support will allow more control over how servicing changes affect apps. For more info, see GitHub #4976.

New APIs

This release includes the following new and modified experimental APIs:

```
Microsoft.Security.Authentication.OAuth

    AuthFailure
    AuthRequestParams
    AuthRequestResult
    AuthResponse
    ClientAuthentication
    CodeChallengeMethodKind
```

```
    OAuth2Manager
    OAuthContract
    TokenFailure
    TokenFailureKind
    TokenRequestParams
    TokenRequestResult
    TokenResponse
```

```
Microsoft.UI.Composition

    CompositionNotificationDeferral
    CompositionProjectedShadow
        MaxOpacity
        MinOpacity
        OpacityFalloff

    CompositionProjectedShadowCaster
        AncestorClip
        Mask

    CompositionProjectedShadowDrawOrder
    CompositionProjectedShadowReceiver
        DrawOrder
        Mask
```

```
Microsoft.UI.Composition.Experimental

    ExpCompositionVisualSurface
    ExpExpressionNotificationProperty
    IExpCompositionPropertyChanged
    IExpCompositionPropertyChangedListener
    IExpCompositor
    IExpVisual
```

```
Microsoft.UI.Content

    AutomationTreeOptions
    ChildSiteLink
    ContentAppWindowBridge
    ContentDisplayOrientations
    ContentEnvironmentStateChangedEventArgs
        DidDisplayScaleChange

    ContentExternalBackdropLink
    ContentExternalOutputLink
```

```
    ContentIsland
        AutomationTreeOption
        Children
        Connected
        ConnectionInfo
        ConnectRemoteEndpoint
        CreateForSystemVisual
        Disconnected
        FindAllForSystemCompositor
        FragmentRootAutomationProvider
        GetBySystemVisual
        InputCapabilities
        IsRemoteEndpointConnected
        LocalToClientTransformMatrix
        LocalToParentTransformMatrix
        NextSiblingAutomationProvider
        ParentAutomationProvider
        Popups
        PreviousSiblingAutomationProvider
        Root

    ContentIslandEnvironment
        CurrentOrientation
        DisplayScale
        NativeOrientation
        ThemeChanged

    ContentIslandStateChangedEventArgs
        DidLocalToClientTransformMatrixChange
        DidLocalToParentTransformMatrixChange

    ContentSite
        InputCapabilities
        LocalToClientTransformMatrix
        LocalToParentTransformMatrix
        SetContentNodeParent
        TryGetAutomationProvider

    ContentSiteAutomationProviderRequestedEventArgs
    ContentSiteEnvironment
        CurrentOrientation
        DisplayScale
        NativeOrientation
        NotifyThemeChanged

    ContentSiteEnvironmentView
        DisplayScale

    ContentSiteView
        AutomationTreeOption
        InputCapabilities
        LocalToClientTransformMatrix
        LocalToParentTransformMatrix

    CoreWindowSiteBridge
```

```
    CoreWindowTopLevelWindowBridge
    DesktopChildSiteBridge
        AcceptRemoteEndpoint
        ConnectionInfo
        CreateWithDispatcherQueue
        IsRemoteEndpointConnected
        RemoteEndpointConnecting
        RemoteEndpointDisconnected
        RemoteEndpointRequestedStateChanged

    DesktopPopupSiteBridge
    DesktopSiteBridge
        TryCreatePopupSiteBridge

    EndpointConnectionEventArgs
    EndpointRequestedStateChangedEventArgs
    IContentIslandEndpointConnectionPrivate
    IContentNodeOwner
    IContentSiteAutomation
    IContentSiteBridgeEndpointConnectionPrivate
    IContentSiteInput
    IContentSiteLink
    IContentSiteLink2
    InputCapabilities
    PopupWindowSiteBridge
    ProcessStarter
    ReadOnlyDesktopSiteBridge
    SystemVisualSiteBridge
```

```
Microsoft.UI.Input

    InputFocusNavigationHost
        GetForSiteLink

    InputKeyboardSource
        GetForWindowId

    InputLayoutPolicy
    InputLightDismissAction
        GetForIsland

    InputPointerActivationBehavior
    InputPointerSource
        ActivationBehavior
        DirectManipulationHitTest
        GetForVisual
        GetForWindowId
        RemoveForVisual
        TouchHitTesting
        TrySetDeviceKinds
```

```
    ProximityEvaluation
    TouchHitTestingEventArgs
```

```
Microsoft.UI.Input.Experimental

    ExpInputSite
    ExpPointerPoint
```

```
Microsoft.UI.Text

    RichEditTextDocument
        GetMath
        SetMath
        SetMathMode
```

```
Microsoft.UI.Windowing

    AppWindow
        DefaultTitleBarShouldMatchAppModeTheme

    DisplayArea
        GetMetricsFromWindowId
```

```
Microsoft.UI.Xaml

    XamlIsland
    XamlRoot
        TryGetContentIsland
```

```
Microsoft.UI.Xaml.Automation.Peers

    AutomationEvents
        Notification

    InkCanvasAutomationPeer
    PagerControlAutomationPeer
```

```
Microsoft.UI.Xaml.Controls

    ContentDialogPlacement
        UnconstrainedPopup

    DoInkPresenterWork
    ElementFactory
    FlowLayout
    FlowLayoutAnchorInfo
    FlowLayoutLineAlignment
    FlowLayoutState
    IApplicationViewSpanningRects
    IndexPath
    InkCanvas
    ISelfPlayingAnimatedVisual
    ItemContainer
        CanUserInvoke
        CanUserInvokeProperty
        CanUserSelect
        CanUserSelectProperty
        ItemInvoked
        MultiSelectMode
        MultiSelectModeProperty

    ItemContainerInteractionTrigger
    ItemContainerInvokedEventArgs
    ItemContainerMultiSelectMode
    ItemContainerUserInvokeMode
    ItemContainerUserSelectMode
    LayoutPanel
    NumberBox
        InputScope
        InputScopeProperty
        TextAlignment
        TextAlignmentProperty

    PagerControl
    PagerControlButtonVisibility
    PagerControlDisplayMode
    PagerControlSelectedIndexChangedEventArgs
    PagerControlTemplateSettings
    ProgressRing
        DeterminateSource
        DeterminateSourceProperty
        IndeterminateSource
        IndeterminateSourceProperty

    RecyclePool
    RecyclingElementFactory
    ScrollingScrollStartingEventArgs
    ScrollingZoomStartingEventArgs
    ScrollView
        ScrollStarting
```

```
        ZoomStarting

    SelectionModel
    SelectionModelChildrenRequestedEventArgs
    SelectionModelSelectionChangedEventArgs
    SelectTemplateEventArgs
    StackLayout
        IsVirtualizationEnabled
        IsVirtualizationEnabledProperty

    StackLayoutState
    TitleBar
    TitleBarAutomationPeer
    TitleBarTemplateSettings
    UniformGridLayoutState
```

```
Microsoft.UI.Xaml.Controls.Primitives

    ScrollPresenter
        ScrollStarting
        ZoomStarting
```

```
Microsoft.Windows.ApplicationModel.Background

    BackgroundTaskBuilder
    BackgroundTaskContract
```

```
Microsoft.Windows.ApplicationModel.Background.UniversalBGTask

    Task
```

```
Microsoft.Windows.ApplicationModel.WindowsAppRuntime

    CompatibilityChange
    CompatibilityContract
    CompatibilityOptions
    DeploymentManager
        Repair

    DeploymentStatus
        PackageRepairFailed
```

```
    ReleaseInfo
    RuntimeInfo
    VersionInfoContract
    WindowsAppRuntimeVersion
```

```
Microsoft.Windows.AppNotifications

    AppNotification
        ConferencingConfig

    AppNotificationConferencingConfig
```

```
Microsoft.Windows.AppNotifications.Builder

    AppNotificationBuilder
        AddCameraPreview

    AppNotificationButton
        SetSettingStyle

    AppNotificationButtonSettingStyle
```

```
Microsoft.Windows.BadgeNotifications

    BadgeNotificationGlyph
    BadgeNotificationManager
    BadgeNotificationsContract
```

```
Microsoft.Windows.Media.Capture

    CameraCaptureUI
    CameraCaptureUIContract
    CameraCaptureUIMaxPhotoResolution
    CameraCaptureUIMaxVideoResolution
    CameraCaptureUIMode
    CameraCaptureUIPhotoCaptureSettings
    CameraCaptureUIPhotoFormat
    CameraCaptureUIVideoCaptureSettings
    CameraCaptureUIVideoFormat
```

```
Microsoft.Windows.Storage

    ApplicationData
        GetForUnpackaged
```

Version 1.7 Experimental (1.7.0-experimental1)

This is the latest release of the experimental channel.

To download, retarget your WinAppSDK NuGet version to `1.7.241114004-experimental1`.

New CameraCaptureUI API

A new CameraCaptureUI API makes it easier to capture photos and videos in your WinAppSDK app. For more info, see GitHub issue #4721.

New Authentication API

A new `OAuth2Manager` API provides a streamlined solution for web authentication, offering OAuth 2.0 capabilities with full feature parity across all Windows platforms supported by WinAppSDK. For more info, see GitHub issue #4772.

New Background Task support

A new `BackgroundTaskBuilder` API brings integrated support for background task registration to your WinAppSDK apps. For more info, see GitHub issue #4822.

New APIs for 1.7-experimental1

This release includes the following new and modified experimental APIs:

C#

```
Microsoft.Security.Authentication.OAuth

    AuthFailure
    AuthRequestParams
    AuthRequestResult
    AuthResponse
    ClientAuthentication
    CodeChallengeMethodKind
```

```
    OAuth2Manager
    OAuthContract
    TokenFailure
    TokenFailureKind
    TokenRequestParams
    TokenRequestResult
    TokenResponse
```

C#

```
Microsoft.UI.Composition

    CompositionNotificationDeferral
    CompositionProjectedShadow
        MaxOpacity
        MinOpacity
        OpacityFalloff

    CompositionProjectedShadowCaster
        AncestorClip
        Mask

    CompositionProjectedShadowDrawOrder
    CompositionProjectedShadowReceiver
        DrawOrder
        Mask
```

C#

```
Microsoft.UI.Composition.Experimental

    ExpCompositionVisualSurface
    ExpExpressionNotificationProperty
    IExpCompositionPropertyChanged
    IExpCompositionPropertyChangedListener
    IExpCompositor
    IExpVisual
```

C#

```
Microsoft.UI.Content

    AutomationOptions
    ChildContentLink
    ContentAppWindowBridge
    ContentDisplayOrientations
    ContentEnvironmentStateChangedEventArgs
        DidDisplayScaleChange

    ContentExternalBackdropLink
    ContentExternalOutputLink
```

```
ContentIsland
    Children
    Compositor
    Connected
    ConnectionInfo
    ConnectRemoteEndpoint
    Create
    Disconnected
    FindAllForCompositor
    FragmentRootAutomationProvider
    GetByVisual
    IsRemoteEndpointConnected
    NextSiblingAutomationProvider
    ParentAutomationProvider
    PreviousSiblingAutomationProvider
    Root
    TransformMatrix

ContentIslandEnvironment
    AutomationOption
    CurrentOrientation
    DisplayScale
    NativeOrientation
    ThemeChanged

ContentSite
    Compositor
    SetContentNodeParent
    SetIsInputPassThrough
    SiteVisual
    TransformMatrix
    TryGetAutomationProvider

ContentSiteAutomationProviderRequestedEventArgs
ContentSiteEnvironment
    CurrentOrientation
    DisplayScale
    NativeOrientation
    NotifyThemeChanged

ContentSiteView
    TransformMatrix

CoreWindowSiteBridge
CoreWindowTopLevelWindowBridge
DesktopChildSiteBridge
    AcceptRemoteEndpoint
    ConnectionInfo
    IsRemoteEndpointConnected
    RemoteEndpointConnecting
    RemoteEndpointDisconnected
    RemoteEndpointRequestedStateChanged

DesktopSiteBridge
    TryCreatePopupSiteBridge
```

```
    EndpointConnectionEventArgs
    EndpointRequestedStateChangedEventArgs
    IContentIslandEndpointConnectionPrivate
    IContentLink
    IContentNodeOwner
    IContentSiteBridge2
    IContentSiteBridgeAutomation
    IContentSiteBridgeEndpointConnectionPrivate
    PopupWindowSiteBridge
    ProcessStarter
    ReadOnlyDesktopSiteBridge
    SystemVisualSiteBridge
```

C#

```
Microsoft.UI.Input

    InputKeyboardSource
        GetForWindowId

    InputLayoutPolicy
    InputLightDismissAction
        GetForIsland

    InputPointerActivationBehavior
    InputPointerSource
        ActivationBehavior
        DirectManipulationHitTest
        GetForVisual
        GetForWindowId
        RemoveForVisual
        TouchHitTesting
        TrySetDeviceKinds

    ProximityEvaluation
    TouchHitTestingEventArgs
```

C#

```
Microsoft.UI.Input.Experimental

    ExpInputSite
    ExpPointerPoint
```

C#

```
Microsoft.UI.Windowing

    AppWindow
        DefaultTitleBarShouldMatchAppModeTheme
```

```
    DisplayArea
        GetMetricsFromWindowId
```

C#

```
Microsoft.UI.Xaml

    XamlIsland
    XamlRoot
        TryGetContentIsland
```

C#

```
Microsoft.UI.Xaml.Automation.Peers

    AutomationEvents
        Notification

    InkCanvasAutomationPeer
    PagerControlAutomationPeer
```

C#

```
Microsoft.UI.Xaml.Controls

    ContentDialogPlacement
        UnconstrainedPopup

    DoInkPresenterWork
    ElementFactory
    FlowLayout
    FlowLayoutAnchorInfo
    FlowLayoutLineAlignment
    FlowLayoutState
    IApplicationViewSpanningRects
    IndexPath
    InkCanvas
    ISelfPlayingAnimatedVisual
    ItemContainer
        CanUserInvoke
        CanUserInvokeProperty
        CanUserSelect
        CanUserSelectProperty
        ItemInvoked
        MultiSelectMode
        MultiSelectModeProperty

    ItemContainerInteractionTrigger
    ItemContainerInvokedEventArgs
    ItemContainerMultiSelectMode
```

```
    ItemContainerUserInvokeMode
    ItemContainerUserSelectMode
    LayoutPanel
    NumberBox
        InputScope
        InputScopeProperty
        TextAlignment
        TextAlignmentProperty

    PagerControl
    PagerControlButtonVisibility
    PagerControlDisplayMode
    PagerControlSelectedIndexChangedEventArgs
    PagerControlTemplateSettings
    ProgressRing
        DeterminateSource
        DeterminateSourceProperty
        IndeterminateSource
        IndeterminateSourceProperty

    RecyclePool
    RecyclingElementFactory
    ScrollingScrollStartingEventArgs
    ScrollingZoomStartingEventArgs
    ScrollView
        ScrollStarting
        ZoomStarting

    SelectionModel
    SelectionModelChildrenRequestedEventArgs
    SelectionModelSelectionChangedEventArgs
    SelectTemplateEventArgs
    StackLayout
        IsVirtualizationEnabled
        IsVirtualizationEnabledProperty

    StackLayoutState
    TitleBar
    TitleBarAutomationPeer
    TitleBarTemplateSettings
    UniformGridLayoutState
```

C#

```
Microsoft.UI.Xaml.Controls.Primitives

    ScrollPresenter
        ScrollStarting
        ZoomStarting
```

C#

```
Microsoft.Windows.ApplicationModel.Background

    BackgroundTaskBuilder
    BackgroundTaskContract
```

C#

```
Microsoft.Windows.ApplicationModel.Background.UniversalBGTask

    Task
```

C#

```
Microsoft.Windows.ApplicationModel.WindowsAppRuntime

    DeploymentManager
        Repair

    DeploymentStatus
        PackageRepairFailed

    ReleaseInfo
    RuntimeInfo
    VersionInfoContract
```

C#

```
Microsoft.Windows.Media.Capture

    CameraCaptureUI
    CameraCaptureUIContract
    CameraCaptureUIMaxPhotoResolution
    CameraCaptureUIMaxVideoResolution
    CameraCaptureUIMode
    CameraCaptureUIPhotoCaptureSettings
    CameraCaptureUIPhotoFormat
    CameraCaptureUIVideoCaptureSettings
    CameraCaptureUIVideoFormat
```

C#

```
Microsoft.Windows.Storage

    ApplicationData
        GetForUnpackaged
```

Bug fixes

This release includes the following bug fixes:

- Changed `SplitButton` so touch input now matches the behavior of mouse input. For more info, see GitHub issue #178.
- Changed cascading menus so sub menus now open immediately if clicked. For more info, see GitHub issue #939.
- Fixed an issue where opening a `ComboBox` which is in a flyout closes all flyouts. For more info, see GitHub issue #1467.
- Fixed an issue where `SwipeControl` would randomly crash in a `ListView`. For more info, see GitHub issue #2527.
- Fixed an issue where drag-and-drop only a `ListViewItem` would leave it in the wrong visual state. For more info, see GitHub issue #3458.
- Fixed an issue in `StackLayout` so that it respects the ItemsRepeater.HorizontalAlignment and ItemsRepeater.VerticalAlignment properties (when StackLayout.Orientation is Vertical and Horizontal respectively). The old layout behaved as if the ItemsRepeater alignment was Stretch. With the fix, the layout results in items aligned to the right when the Right alignment is used, for example. For more info, see GitHub issue #3842.
- Fixed an issue where deleting items in the `ItemsRepeater`'s source would not generate items which moved up into view. For more info, see GitHub issue #6661.
- Fixed an issue where the right Alt key would not show keytips for Access Keys. For more info, see GitHub issue #8447. **Note:** This may result in key events for the right Alt key no longer being delivered to handles in the app or controls.
- Fixed a crash where `UniformGridLayout` would sometimes pick a wrong layout anchor and cause infinite layout passes when scrolling backwards. For more info, see GitHub issue #9199.
- Fixed an issue where setting `NavigationFailedEventArgs.Handled` to True would still throw an exception. For more info, see GitHub issue #9632.
- Fixed an issue where `TabView` would not apply any specified `CornerRadius`. For more info, see GitHub issue #9846.
- Fixed a potential layout cycle crash in `StackLayout`. For more info, see GitHub issue #9852.
- Fixed a potential crash in `ItemsView` when removing items. For more info, see GitHub issue #9868.

Related topics

- Stable channel
- Preview channel
- Install tools for the Windows App SDK
- Create your first WinUI 3 project
- Use the Windows App SDK in an existing project
- Deploy apps that use the Windows App SDK

Stable channel release notes for the Windows App SDK 1.6

Article • 03/19/2025

The stable channel provides releases of the Windows App SDK that are supported for use by apps in production environments. Apps that use the stable release of the Windows App SDK can also be published to the Microsoft Store.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Downloads for the Windows App SDK

> **Note**
>
> The Windows App SDK Visual Studio Extensions (VSIX) are no longer distributed as a separate download. They are available in the Visual Studio Marketplace inside Visual Studio.

Version 1.6.6 (1.6.250228001)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.6 release.

- Fixed an issue where a child window posting WM_NCMOUSELEAVE to the parent window would result in a loop that blocks new mouse input events.
- Fixed a crash which would occur on the next AppWindow.Changed event after a WebView2 process failure.
- Fixed a potential crash when using an Accessibility tool and closing a window.
- Fixed an issue where a textbox would not accept key input if given focus by clicking in the area of the clear button of the textbox. For more info, see GitHub issue #7703.

- Fixed an issue where a tooltip is not shown for the Minimize button in the titlebar when using `ExtendsContentIntoTitleBar=true`. For more info, see GitHub issue #9149.

This release includes the following new APIs:

A new `IsPlaceholderContent` property on `WidgetInfo` and `WidgetUpdateRequestOptions` enables a Widget provider to indicate that it would display placeholder content if rendered. For example, a Widget that shows weather information should set IsPlaceholderContent to true if the user has not yet specified a weather location and the Widget is merely showing weather information for a default location like Seattle. When a Widget is marked as placeholder, certain hosts may decide to hide the Widget or prioritize other Widgets.

```
Microsoft.Windows.Widgets.Providers

    WidgetInfo
        IsPlaceholderContent

    WidgetUpdateRequestOptions
        IsPlaceholderContent
```

Version 1.6.5 (1.6.250205002)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.6 release.

- Fixed several memory leak issues.

Version 1.6.4 (1.6.250108002)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.6 release.

- Fixed an issue with text selection highlighting in a multi-line TextBox. For more info, see GitHub issue #9965.
- Fixed an issue where the DDLM package would sometimes not install, preventing launch of unpackaged apps. For more info, see GitHub issue #3855.
- Fixed a potential crash in Detours in some scenarios. For more info, see GitHub issue #4937.

- Fixed another potential issue where a menu off a CommandBar may incorrectly open up instead of down when the CommandBar is at the bottom of the window.
- Fixed a potential crash when running on older graphics hardware.
- Fixed a potential crash in pointer event handling when closing a window.
- Fixed a potential crash caused by `CUIAWindow::InitIds` sometimes triggering reentrancy.
- Fixed a potential crash when using `CompositionCapabilities.Changed` event.
- Fixed an issue with some Unicode characters displaying as squares in TextBox/RichEditBox.
- Fixed `PackageDeploymentManager.EnsurePackage*Async()` handling of `options.RegisterNewerIfAvailable`. For more info, see GitHub issue #4864.

Version 1.6.3 (1.6.241114003)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.6 release.

- Fixed an issue where reading the `AppWindow.ExtendsContentIntoTitleBar` property turns on custom titlebar rendering. For more info, see GitHub issue #9988.
- Fixed a potential crash during destruction of a `TextBox`/`RichEditBox`. For more info, see GitHub issue #9070.
- Fixed an issue where `PackageDeploymentManager.IsPackageReadyOrNewerAvailable()` failed. For more info, see GitHub issue #4817.
- Fixed an issue where `ScrollViewer` would leak.
- Added detection for a rare scenario where the app stops rendering and never recovers.
- Fixed an issue where `PackageDeploymentManager.RegisterPackageSetAsync()` requires URI when it should be optional to register by PackageFamilyName.
- Fixed an issue that prevented apps from being installed or uninstalled. For more info, see GitHub issue #4881.

This release includes the following new APIs which allow for providers of Widgets to incorporate web content in their Widgets:

```
C#

Microsoft.Windows.Widgets.Providers
  IWidgetManager2
  IWidgetProviderMessage
  IWidgetResourceProvider
  WidgetManager
     SendMessageToContent
```

```
WidgetMessageReceivedArgs
WidgetResourceRequest
WidgetResourceRequestedArgs
WidgetResourceResponse
```

Version 1.6.1 (1.6.240923002)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.6 release.

- Fixed a crash when using FocusVisualKind.Reveal(). For more info, see GitHub issue #9966.
- Fixed noisy C++ exceptions from Bcp47Langs.dll. For more info, see GitHub issue #4691. Note that this fix removes the synchronization with `Windows.Globalization.ApplicationLanguages.PrimaryLanguageOverride`.
- Fixed an issue where an extra `Unloaded` event was raised immediately after showing a `ContentDialog`. For more info, see GitHub issue #8402.
- Fixed an issue where a CommandBar menu might have incorrectly opened up even when there was room for it to open down.
- Fixed some issues where input to `InputNonClientPointerSource` regions was not handled correctly when the top-level window was running in right-to-left mode.
- Fixed the compile-time check for the Windows SDK framework version to handle the slightly different framework name used for .NET 9.

Version 1.6

The following sections describe new and updated features and known issues for version 1.6.

In an existing Windows App SDK 1.5 app, you can update your Nuget package to 1.6.240829007 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Downloads for the Windows App SDK.

Required project changes for 1.6

C++ project changes

When updating a C++ project to 1.6, you'll need to add a project reference to the `Microsoft.Web.WebView2` package. If you update via NuGet Package Manager in Visual

Studio, this dependency will be added for you.

C# project changes

In 1.6, Windows App SDK managed apps require Microsoft.Windows.SDK.NET.Ref `*.*.*.38` or later, which can be specified via WindowsSdkPackageVersion in your `csproj` file. For example:

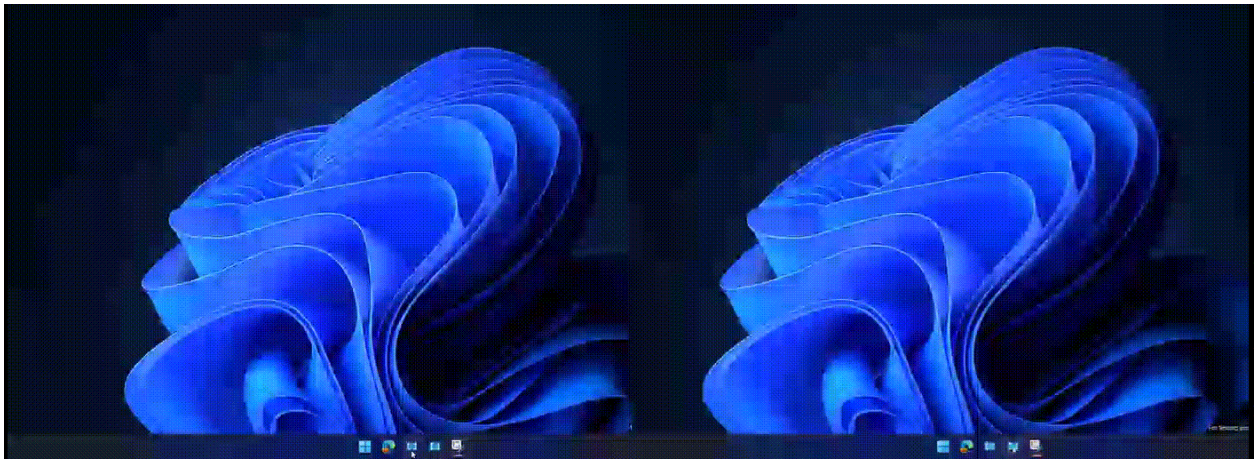
XML

```
<Project Sdk="Microsoft.NET.Sdk">
   <PropertyGroup>
       <OutputType>WinExe</OutputType>
       <TargetFramework>net8.0-windows10.0.22621.0</TargetFramework>
       <TargetPlatformMinVersion>10.0.17763.0</TargetPlatformMinVersion>
       <WindowsSdkPackageVersion>10.0.22621.38</WindowsSdkPackageVersion>
   <PropertyGroup>
   ...
```

In addition, Windows App SDK managed apps should update to Microsoft.Windows.CsWinRT `2.1.1` (or later).

> **Note**
>
> These manual references will no longer be needed once the next .NET SDK servicing update is released.

Native AOT support



The .NET `PublishAot` project property is now supported for native Ahead-Of-Time compilation. For details on Native AOT, see Native AOT Deployment. Because AOT builds

on Trimming support, much of the following trimming-related guidance applies to AOT as well.

For `PublishAot` support and trimming support, in addition to the C# project changes described in the previous section you'll also need a package reference to Microsoft.Windows.CsWinRT `2.1.1` (or later) to enable the source generator from that package until the next .NET SDK servicing update is released when it will no longer be required.

For more info, see the CsWinRT Trimming / AOT support doc and the CsWinRT 2.1.1 Release Notes.

Because the Windows App SDK invokes publishing targets when F5 deploying, we recommend enabling `PublishAot` at NuGet restore time by adding this to your `csproj` file:

```xml
<PublishAot>true</PublishAot>
```

Resolving AOT Issues

In this release, the developer is responsible for ensuring that all types are properly rooted to avoid trimming (such as with reflection-based `{Binding}` targets). Later releases will enhance both C#/WinRT and the XAML Compiler to automate rooting where possible, alert developers to trimming risks, and provide mechanisms to resolve.

Partial Classes

C#/WinRT also includes `PublishAot` support in version 2.1.1. To enable a class for AOT publishing with C#/WinRT, it must first be marked `partial`. This allows the C#/WinRT AOT source analyzer to attribute the classes for static analysis. Only classes (which contain methods, the targets of trimming) require this attribute.

Unsafe Code Error

The CsWinRT source generator might generate code that makes use of `unsafe`. If you hit such an error during compilation or a diagnostic warning for it (CS0227 for "Unsafe code may only appear if compiling with /unsafe"), you should set EnableUnsafeBlocks to true. For more info, see GitHub issue CsWinRT #1721.

WebView2 not yet AOT compatible

The WebView2 projections in `Microsoft.Web.WebView2` package version 1.0.2651.64 are not yet AOT compatible. This will be fixed in an upcoming release of the `Microsoft.Web.WebView2` package, which you can then reference in your project.

Reflection-Free Techniques

To enable AOT compatibility, reflection-based techniques should be replaced with statically typed serialization, AppContext.BaseDirectory, typeof(), etc. For details, see Introduction to trim warnings.

Rooting Types

Until full support for `{Binding}` is implemented, types may be preserved from trimming as follows:

Given project `P` consuming assembly `A` with type `T` in namespace `N`, which is only dynamically referenced (so normally trimmed), `T` can be preserved via:

`P.csproj`:

XML

```xml
<ItemGroup>
    <TrimmerRootDescriptor Include="ILLink.Descriptors.xml" />
</ItemGroup>
```

`ILLink.Descriptors.xml`:

XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<linker>
    <assembly fullname="A">
        <type fullname="N.T" preserve="all" />
    </assembly>
</linker>
```

For complete root descriptor XML expression syntax, see Root Descriptors.

> ⓘ Note

Dependency packages that have not yet adopted AOT support may exhibit runtime issues.

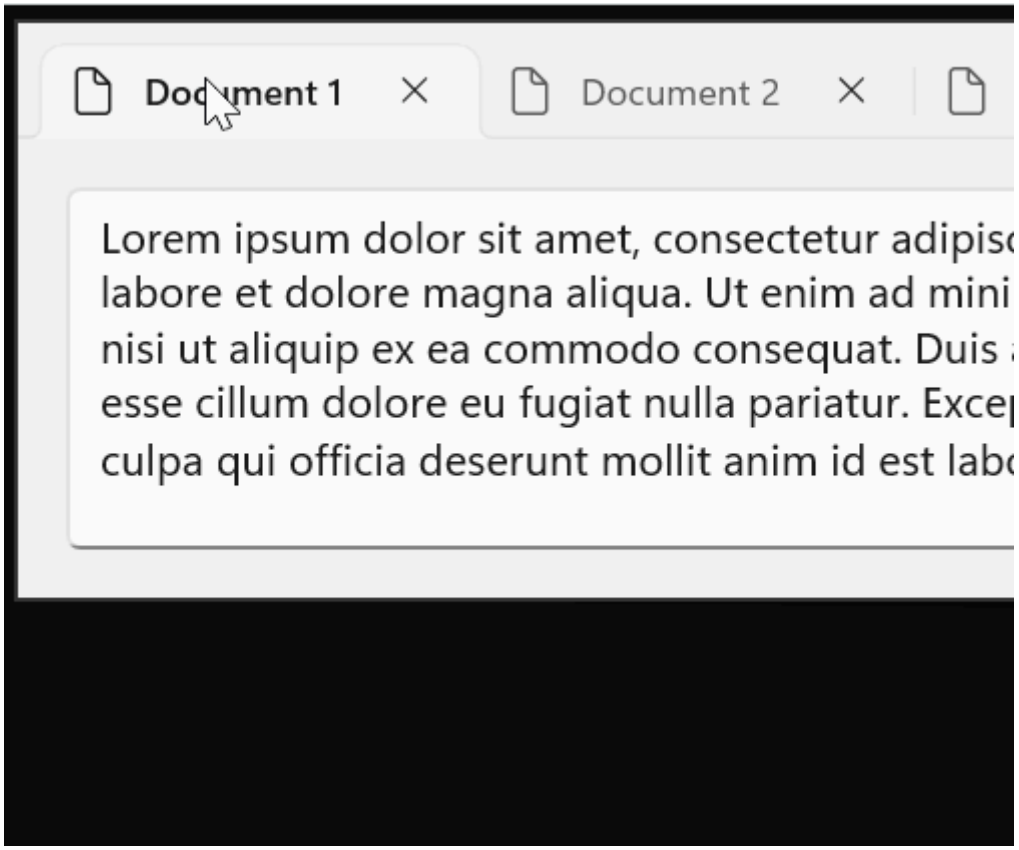
Decoupled WebView2 versioning

The Windows App SDK now consumes the Edge WebView2 SDK as a NuGet reference rather than embedding a hardcoded version of the Edge WebView2 SDK. The new model allows apps to choose a newer version of the `Microsoft.Web.WebView2` package instead of being limited to the version with which the Windows App SDK was built. The new model also allows apps to reference NuGet packages which also reference the Edge WebView2 SDK. For more info, see GitHub issue #5689.

New Package Deployment APIs

The Package Management API has received several enhancements including Is*ReadyOrNewerAvailable*(), EnsureReadyOptions.RegisterNewerIfAvailable, Is*Provisioned*(), IsPackageRegistrationPending(), and several bug fixes. See PackageManagement.md and Pull Request #4453 for more details.

Improved TabView tab tear-out



`TabView` supports a new `CanTearOutTabs` mode which provides an enhanced experience for dragging tabs and dragging out to a new window. When this new option is enabled, tab dragging is very much like the tab drag experience in Edge and Chrome where a new window is immediately created during the drag, allowing the user to drag it to the edge of the screen to maximize or snap the window in one smooth motion. This implementation also doesn't use drag-and-drop APIs, so it isn't impacted by any limitations in those APIs. Notably, tab tear-out is supported in processes running elevated as Administrator.

Other notable changes

- Added a new `ColorHelper.ToDisplayName()` API, filling that gap from UWP.
- Added a new `Microsoft.Windows.Globalization.ApplicationLanguages` class, which notably includes a new `PrimaryLanguageOverride` feature. For more info, see GitHub issue #4523.
- Unsealed `ItemsWrapGrid`. This should be a backward-compatible change.
- `PipsPager` supports a new mode where it can wrap between the first and last items.

- `RatingControl` is now more customizable, by moving some hard-coded style properties to theme resources. This allows apps to override these values to better customize the appearance of RatingControl.

- WinUI 3 has changed to the typographic model for font selection rather than the legacy weight/stretch/style model. The typographic model is required for some newer fonts, including Segoe UI Variable, and enables enhanced font capabilities. Some older fonts which rely on the weight/stretch/style model for selection may not be found with the typographic model.

Known Issues

- If the debugger is set to break on all C++ exceptions, it will break on some noisy exceptions on start-up in the BCP47 (Windows Globalization) code. For more info, see GitHub issue #4691.
- Component library packages which reference the WinAppSDK 1.6 package will not correctly get the referenced WebView2 package contents. For more info, see WebView2Feedback #4743. A workaround is to add a direct reference to the `Microsoft.Web.WebView2` package where needed.
- Apps compiled with Native AOT might sometimes experience a hanging issue after page navigation due to a race condition in the .NET runtime's GC thread. For more info, see .NET issue #104582.
- The initial release of 1.6.0 introduced an issue with one of our dependencies that we expect will be resolved in an upcoming release of the .NET SDK. If you experience an error with the version of your Microsoft.Windows.SDK.NET reference, you'll need to explicitly reference the version of the .NET SDK that is specified by your error message. For example, if the error says you need version 10.0.19041.38, add the following to your `.csproj` file:
  - `<WindowsSdkPackageVersion>10.0.19041.38</WindowsSdkPackageVersion>`.

Bug Fixes

- Fixed a crash when setting `InfoBar.IsOpen` in .xaml. For more info, see GitHub issue #8391.
- Fixed an issue where HTML elements would lose pointer capture when the mouse moved outside of the `WebView2` bounds. For more info, see GitHub issue #8677.
- Fixed an issue where drag and drop into a flyout with `ShouldConstrainToRootBounds=false` did not work. For more info, see GitHub issue #9276.
- Fixed an issue where `ms-appx://` references did not work when `PublishSingleFile` is enabled. For more info, see GitHub issue #9468.
- Fixed an issue where debugger symbols weren't working correctly for some binaries. For more info, see GitHub issue #4633.
- Fixed a potential crash when subclassing `NavigationView`.
- Fixed an issue where table borders in a `RichEditBox` would not correctly erase when scrolling or reducing the size of the table.
- Fixed an issue where flyouts from `MediaTransportControls` had a fully transparent background.
- Fixed an issue where dragging into a WebView2 would fail or drop in the wrong location on display scale factors other than 100% or when system text scaling is enabled.

- Fixed an issue where `TextBox`/`RichEditBox` would not announce to Accessibility tools when input is blocked due to being at the `MaxLength` limit.
- Fixed a few issues around handling of custom titlebar scenarios. For more info, see GitHub issues #7629, #9670, #9709 and #8431.
- Fixed an issue where `InfoBadge` icon was not visible. For more info, see GitHub issue #8176.
- Fixed an issue with icons sometimes showing in the wrong position in `CommandBarFlyout`. For more info, see GitHub issue #9409.
- Fixed an issue with keyboard focus in menus when opening or closing a sub menu. For more info, see GitHub issue #9519.
- Fixed an issue with `TreeView` using the incorrect `IsExpanded` state when recycling items. For more info, see GitHub issue #9549.
- Fixed an issue when using an ElementName binding in an `ItemsRepeater.ItemTemplate`. For more info, see GitHub issue #9715.
- Fixed an issue with the first item in an `ItemsRepeater` sometimes having an incorrect position. For more info, see GitHub issue #9743.
- Fixed an issue with `InputNonClientPointerSource` sometimes breaking input to the min/max/close buttons. For more info, see GitHub issue #9749.
- Fixed a compile error when using Microsoft.UI.Interop.h with clang-cl. For more info, see GitHub issue #9771.
- Fixed an issue where the `CharacterReceived` event was not working in `ComboBox`/`TextBox`. For more info, see GitHub issue #9786.
- Fixed an issue where duplicate `KeyUp` events were raised for arrow and tab keys. For more info, see GitHub issue #9399.
- Fixed an issue where the `PowerManager.SystemSuspendStatusChanged` event was unusable to get the `SystemSuspendStatus`. For more info, see GitHub issue #2833.
- Fixed an issue where initial keyboard focus was not correctly given to a `WebView2` when that was the only control in the window.
- Fixed an issue when using `ExtendsContentIntoTitleBar=true` where the Min/Max/Close buttons did not correctly appear in the UI Automation, which prevented Voice Access from showing numbers for those buttons.
- Fixed an issue where an app might crash in a lock check due to unexpected reentrancy.
- Fixed an issue where `Hyperlink` colors did not correctly update when switching into a high contrast theme.
- Fixed an issue where changing the collection of a `ListView` in a background window may incorrectly move that window to the foreground and take focus.

- Fixed an issue where calling `ItemsRepeater.StartBringIntoView` could sometimes cause items to disappear.
- Fixed an issue where touching and dragging on a `Button` in a `ScrollViewer` would leave it in a pressed state.
- Updated IntelliSense, which was missing information for many newer types and members.
- Fixed an issue where clicking in an empty area of a `ScrollViewer` would always move focus to the first focusable control in the `ScrollViewer` and scroll that control into view. For more info, see GitHub issue #597.
- Fixed an issue where the `Window.Activated` event sometimes fired multiple times. For more info, see GitHub issue #7343.
- Fixed an issue where setting the `NavigationViewItem.IsSelected` property to `true` prevented its children from showing when expanded. For more info, see GitHub issue #7930.
- Fixed an issue where `MediaPlayerElement` would not properly display captions with `None` or `DropShadow` edge effects. For more info, see GitHub issue #7981.
- Fixed an issue where the `Flyout.ShowMode` property was not used when showing the flyout. For more info, see GitHub issue #7987.
- Fixed an issue where `NumberBox` would sometimes have rounding errors. For more info, see GitHub issue #8780.
- Fixed an issue where using a library compiled against an older version of WinAppSDK could hit an error trying to find a type or property. For more info, see GitHub issue #8810.
- Fixed an issue where initial keyboard focus was not set when launching a window. For more info, see GitHub issue #8816.
- Fixed an issue where `FlyoutShowMode.TransientWithDismissOnPointerMoveAway` didn't work after the first time it was shown. For more info, see GitHub issue #8896.
- Fixed an issue where some controls did not correctly template bind `Foreground` and `Background` properties. For more info, see GitHub issue #7070, #9020, #9029, #9083 and #9102.
- Fixed an issue where `ThemeResource`s used in `VisualStateManager` setters wouldn't update on theme change. This commonly affected controls in flyouts. For more info, see GitHub issue #9198.
- Fixed an issue where `WebView` would lose key focus, resulting in extra blur/focus events and other issues. For more info, see GitHub issue #9288.
- Fixed an issue where `NavigationView` could show a binding error in debug output. For more info, see GitHub issue #9384.

- Fixed an issue where SVG files defining a negative viewbox no longer rendered. For more info, see GitHub issue #9415.
- Fixed an issue where changing `ItemsView.Layout` orientation caused an item to be removed. For more info, see GitHub issue #9422.
- Fixed an issue where scrolling a `ScrollView` generated a lot of debug output. For more info, see GitHub issue #9434.
- Fixed an issue where `MapContorl.InteractiveControlsVisible` did not work properly. For more info, see GitHub issue #9486.
- Fixed an issue where `MapControl.MapElementClick` event didn't properly fire. For more info, see GitHub issue #9487.
- Fixed an issue where x:Bind didn't check for null before using a weak reference, which could result in a crash. For more info, see GitHub issue #9551.
- Fixed an issue where changing the `TeachingTip.Target` property didn't correctly update its position. For more info, see GitHub issue #9553.
- Fixed an issue where dropdowns did not respond in WebView2. For more info, see GitHub issue #9566.
- Fixed a memory leak when using `GeometryGroup`. For more info, see GitHub issue #9578.
- Fixed an issue where scrolling through a very large number of items from an `ItemRepeater` in a `ScrollView` could cause blank render frames. For more info, see GitHub issue #9643.
- Fixed an issue where `SceneVisual` wasn't working.

New APIs in 1.6.0

Version 1.6.0 includes the following new APIs compared to the stable 1.5 release:

C#

```
Microsoft.UI

    ColorHelper
        ToDisplayName
```

C#

```
Microsoft.UI.Input

    EnteredMoveSizeEventArgs
    EnteringMoveSizeEventArgs
    ExitedMoveSizeEventArgs
    InputNonClientPointerSource
        EnteredMoveSize
```

```
        EnteringMoveSize
        ExitedMoveSize
        WindowRectChanged
        WindowRectChanging

    MoveSizeOperation
    WindowRectChangedEventArgs
    WindowRectChangingEventArgs
```

C#

```
Microsoft.UI.Xaml

    XamlRoot
        CoordinateConverter
```

C#

```
Microsoft.UI.Xaml.Automation.Peers

    ScrollPresenterAutomationPeer
```

C#

```
Microsoft.UI.Xaml.Controls

    PipsPager
        WrapMode
        WrapModeProperty

    PipsPagerWrapMode
    TabView
        CanTearOutTabs
        CanTearOutTabsProperty
        ExternalTornOutTabsDropped
        ExternalTornOutTabsDropping
        TabTearOutRequested
        TabTearOutWindowRequested

    TabViewExternalTornOutTabsDroppedEventArgs
    TabViewExternalTornOutTabsDroppingEventArgs
    TabViewTabTearOutRequestedEventArgs
    TabViewTabTearOutWindowRequestedEventArgs
```

C#

```
Microsoft.Windows.Globalization
```

```
    ApplicationLanguages
```

C#

```
Microsoft.Windows.Management.Deployment

    EnsureReadyOptions
        RegisterNewerIfAvailable

    PackageDeploymentFeature
    PackageDeploymentManager
        IsPackageDeploymentFeatureSupported
        IsPackageProvisioned
        IsPackageProvisionedByUri
        IsPackageReadyOrNewerAvailable
        IsPackageReadyOrNewerAvailableByUri
        IsPackageSetProvisioned
        IsPackageSetReadyOrNewerAvailable

    PackageReadyOrNewerAvailableStatus
```

C#

```
Microsoft.Windows.Storage

    ApplicationData
    ApplicationDataContainer
    ApplicationDataContract
    ApplicationDataCreateDisposition
    ApplicationDataLocality
```

Related topics

- Preview channel
- Experimental channel
- Install tools for the Windows App SDK
- Create your first WinUI 3 project
- Use the Windows App SDK in an existing project
- Deploy apps that use the Windows App SDK

Stable channel release notes for the Windows App SDK 1.5

Article • 09/05/2024

The stable channel provides releases of the Windows App SDK that are supported for use by apps in production environments. Apps that use the stable release of the Windows App SDK can also be published to the Microsoft Store.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Downloads for the Windows App SDK

> **Note**
>
> The Windows App SDK Visual Studio Extensions (VSIX) are no longer distributed as a separate download. They are available in the Visual Studio Marketplace inside Visual Studio.

Version 1.5.9 (1.5.250108004)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.5 release.

- Fixed another potential issue where a menu off a CommandBar may incorrectly open up instead of down when the CommandBar is at the bottom of the window.
- Fixed a potential crash when running on older graphics hardware.

Version 1.5.8 (1.5.241107002)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.5 release.

- Fixed a potential crash when the input queue on the thread appears to be in a bad state.
- Fixed some potential crashes when using `{x:Bind}` by checking every component for null. Apps wanting this fix need to recompile with this release.

Version 1.5.7 (1.5.241001000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.5 release.

- Added detection for a rare scenario where the app stops rendering and never recovers.
- Fixed a potential crash when a UI thread uninitializes.
- Fixed a potential issue where a menu off a CommandBar might have incorrectly opened up instead of down when the CommandBar was at the bottom of the window.

Version 1.5.6 (1.5.240802000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.5 release.

- Fixed an issue with `InputNonClientPointerSource` sometimes breaking input to the Min/Max/Close buttons. For more info, see GitHub issue #9749.
- Fixed a potential crash when closing a window while using a touchpad.

Version 1.5.5 (1.5.240627000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.5 release.

- Fixed a potential crash during image loading when the image source is changed before loading completes.
- Fixed a potential crash when using `InputNonClientPointerSource`.
- Fixed an issue where a window containing only a WebView2 control doesn't correctly set initial keyboard focus into the WebView2, leaving it unusable by keyboard and accessibility tools. For more info, see GitHub issue WebView2Feedback#2330.

Version 1.5.4 (1.5.240607001)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.5 release.

- Fixed an issue with the fix for GitHub issue #8857 to properly merge a library's resources.pri into the app's resources.pri.
- Fixed an issue where WebView2 sent extra `blur` and `focus` events on every click. For more info, see GitHub issue #9288. This also fixes problems with dropdowns not responding, as seen in GitHub issue #9566.
- Fixed an issue where creating a new window or popup canceled any in-progress drag operation. For more info, see GitHub issue #9360.
- Fixed an issue where SVG files defining a negative viewbox no longer rendered. For more info, see GitHub issue #9415.
- Fixed an issue where `x:Bind` didn't check for **null** before using a weak reference, which could result in a crash. For more info, see GitHub issue #9551.
- Fixed an issue where a resize cursor was incorrectly shown at titlebar edges when a window was maximized, which also caused incorrect behavior when clicking or dragging in those areas. For more info, see GitHub issue #8665.

Version 1.5.3 (1.5.240428000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.5 release.

- Fixed an issue where `ItemsRepeater` sometimes caused its window to be brought to the front.
- Fixed a potential crash when the transparency state of an `AcrylicBrush` was updated.
- Fixed an issue where `ItemsRepeater` would not generate enough items if it was in a `ShouldConstrainToRootBounds="false"` popup that was taller or wider than the main window.
- Fixed an issue where error **MSB4086** could occur at build time if the project file didn't specify a *TargetPlatformVersion*. This error was hit as part of GitHub issue #9531.

Version 1.5.2 (1.5.240404000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.5 release and a new feature for Widget feed providers.

Widget feed provider updates

- Feed providers are now able to show announcements in the Widgets Dashboard:
  - The new `FeedManager.TryShowAnnouncement` API allows a Feed Provider to request that the Widget Dashboard show their announcements. To show your announcement, create a `FeedAnnouncement` object and call the `FeedManager.TryShowAnnouncement` API.
  - Feed announcements will have their own badges displayed in the taskbar that can be customized using the `FeedAnnouncement` API.
- Feed providers can now opt in to receive Analytics and Error Reports information about their provider as a user interacts with it. To opt in, implement the `IFeedProviderAnalytics` and/or `IFeedProviderErrors` interfaces by your provider.

The feed provider updates includes the following new APIs:

C#

```
Microsoft.Windows.Widgets.Feeds.Providers

    FeedAnalyticsInfoReportedArgs
    FeedErrorInfoReportedArgs
    FeedManager
        SendMessageToContent
        TryShowAnnouncement

    FeedMessageReceivedArgs
    FeedResourceRequest
    FeedResourceRequestedArgs
    FeedResourceResponse
    IFeedAnnouncementInvokedTarget
    IFeedManager2
    IFeedProviderAnalytics
    IFeedProviderErrors
    IFeedProviderMessage
    IFeedResourceProvider
```

C#

```
Microsoft.Windows.Widgets.Notifications

    AnnouncementActionKind
    AnnouncementTextColor
    FeedAnnouncement
    FeedAnnouncementInvokedArgs
```

Bug fixes

- Fixed an issue where `DispatcherTimer` would not fire at the correct time if the primary display wasn't running at 60 Hz.
- Fixed an issue where a textbox input method editor (IME) window would be incorrectly positioned when running at a scale factor other than 100%. For more info, see GitHub issue #9382.
- Fixed an issue where a drag-and-drop operation started from another app might not allow the correct Copy/Move/Link drop operations.
- Fixed a potential crash when a ListView's `ItemsSource` is reset and given focus right away.
- Fixed a problem where `ShouldConstrainToRootBounds="false"` popups/flyouts don't apply any inherited scale transform. For more info, see GitHub issues #9118 and #9433.
- Fixed a potential crash when a Composition object is destroyed with an animation still running.
- Fixed a potential crash on shutdown in `NavigationView::RevokeNavigationViewItemRevokers`.

Version 1.5.1 (1.5.240311000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.5 release.

- Fixed an issue where self-contained apps may fail to build due to not being able to copy "map.html".
- Fixed an issue where `MapControl` would fail to initialize due to rejecting valid tokens. For more info, see GitHub issue #9324.
- Fixed an issue where `MapControl` loaded with a blue background. For more info, see GitHub issue #9377.
- Fixed an issue where clicking on the chevron of a `NavigationViewItem` did not correctly expand or collapse on a single click. This also caused menus to show as blank when clicking the chevron in `PaneDisplayMode="Top"` mode. For more info, see GitHub issues #9423 and #9426.
- Fixed an issue where tapping on a `NavigationViewItem` with touch or pen would prevent the item from responding to any future input. For more info, see GitHub issue #9429.
- Fixed a crash when clicking on an item in the `NavigationView.PaneFooter` area. For more info, see GitHub issue #9396.
- Fixed an issue where icons in menus were sometimes showing in the wrong place. For more info, see GitHub issue #9409.

- Fixed an issue where acrylic does not show on menus until switching away from the window and back. For more info, see GitHub issue #9406.
- Fixed a crash which could occur during `TextBox`/`RichEditBox` initialization. For more info, see GitHub issue #9216.
- Fixed some noisy exceptions which `NavigationView` threw and caught on destruction.
- Fixed an issue where a "pinch to zoom" gesture would sometimes show up as a pan or tap due to an incorrect `PointerCaptureLost` message firing.

Version 1.5

The following sections describe new and updated features and known issues for version 1.5.

In an existing Windows App SDK 1.4 app, you can update your Nuget package to 1.5.240227000 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Downloads for the Windows App SDK.

XAML Islands runtime and shutdown updates

There is a behavioral difference between WinAppSDK 1.4 and WinAppSDK 1.5 for XAML Islands-based apps when the last XAML Window on any thread is closed.

- In WinAppSDK 1.4, the XAML runtime always exits the thread's event loop when the last XAML window on a thread is closed.
- In WinAppSDK 1.5:
  - If your app is a WinUI Desktop app, the default behavior is still the same as in WinAppSDK 1.4.
  - If you're using XAML for the DesktopWindowXamlSource ("XAML Islands") API, the default behavior is now that XAML does not automatically exit the thread's event loop.
  - In both modes, you can change this behavior by setting the `Application.DispatcherShutdownMode` property.

For more information, see the documentation for the `Application.DispatcherShutdownMode` property when available. This completes GitHub proposal #8492.

There is a behavioral difference between WinAppSDK 1.4 and WinAppSDK 1.5 for XAML Islands-based apps in the lifetime of the XAML runtime:

- In WinAppSDK 1.4, the XAML runtime shuts down on a thread if either all `WindowsXamlManager` and `DesktopWindowXamlSource` objects on a given thread are closed or shut down, or the `DispatcherQueue` running on that thread is shut down (in this case, the XAML runtime shuts down during the `DispatcherQueue.FrameworkShutdownStarting` stage).
- In WinAppSDK 1.5, the XAML runtime shuts down on a thread only when the DispatcherQueue running on that thread is shut down (the Xaml runtime always shuts down during the `DispatcherQueue.FrameworkShutdownStarting` stage).

For more information, see the documentation for the `WindowsXamlManager` class when available.

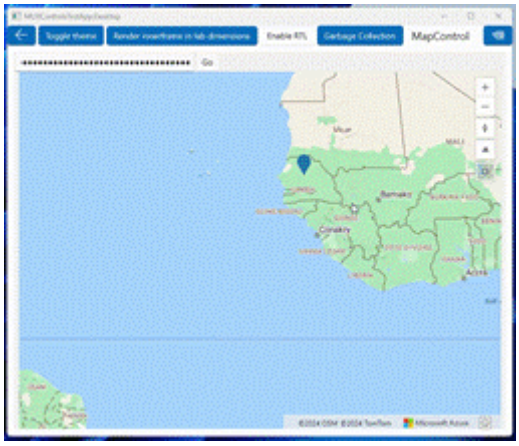
There is a behavioral difference in `WindowsXamlManager.InitializeForCurrentThread()`:

- In WinAppSDK 1.4, `WindowsXamlManager.InitializeForCurrentThread()` returns a unique instance of a `WindowsXamlManager` object with each call.
- In WinAppSDK 1.5, `WindowsXamlManager.InitializeForCurrentThread()` returns an existing instance if one already exists on the thread. `Close/Dispose()` is now ignored.

WinUI Maps control

The WinUI `Maps` control is now available! This control is powered by WebView2 and Azure Maps, providing the following features:

- Panning and zooming with either the map buttons or touch.
- Changing the style of the map to satellite, terrain, or street view.
- Programmatically adding interactable pins with developer-customizable icons to the map.
- Developer customization for where the map is centered on initial load.
- Control for developers over hiding or showing the buttons for panning, zooming, and map styles.

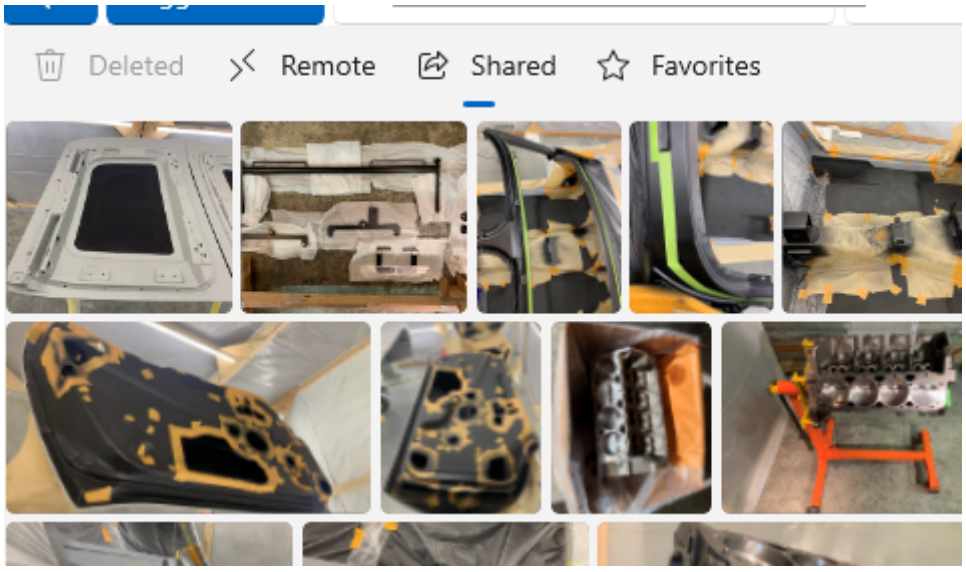


> **Note**
>
> To use the `Maps` control, you'll need an Azure Maps key. To create the key, see the Azure Maps documentation page for creating a web app.

The `Maps` control is entirely new and we welcome your feedback to evaluate its future direction!

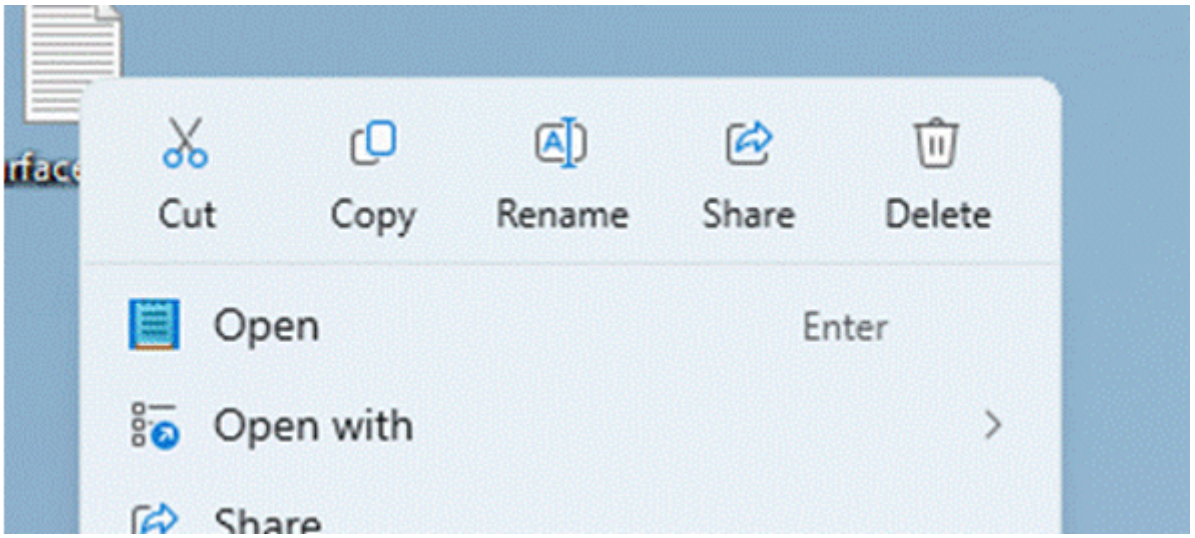
New SelectorBar control

In 1.5, we've added a new `SelectorBar` control for enabling users to switch between multiple views of data. This control was previously known as "SegmentedControl" on our 1.5 roadmap.



Labels in the CommandBarFlyout primary commands

The visuals of the `CommandBarFlyout` have been updated to display a text label for the items in the primary commands area if the `Label` property has been set on the `AppBarButton`. Previously, the primary commands in the `CommandBarFlyout` area only displayed an icon, but now they can show both an icon and a label for improved usability.



WebView2 support for custom environment/options

The WinUI `WebView2` control now exposes the ability to customize the underlying `CoreWebView2` object with a custom `CoreWebView2Environment` and `CoreWebView2ControllerOptions`. This enables the app author to specify an alternate path from which to load the WebView2Runtime, choose to use a different *UserDataFolder*, or set options such as *IsPrivateModeEnabled* and *ScriptLocale*.

Support for .NET 8

We added support for .NET 8 in a recent 1.4 servicing release, but that still kept the warning for the use of platform-specific RIDs. In 1.5, we completed that work so that the warning is no longer present.

Improved debugging and source availability

We now inject Github source server information for code in the microsoft-ui-xaml repo into our public symbols, allowing debuggers to automatically download source code. We also made other fixes and improvements to our symbols across the entire WinAppSDK to improve the debugging experience.

Improved functionality for debugging layout cycles

Debugging layout cycles in a WinUI app can be a challenge, so in 1.5 the `DebugSettings` object now exposes options to enable improved logging and breakpoints for the layout process to make it easier to debug and fix layout cycles in the app.

Other new features from across the WinAppSDK

- Added support for the PublishSingleFile deployment model. For more info about PublishSingleFile, see the Single-file deployment documentation.
- Added improvements to screen reader support, text scaling support, and other accessibility features.
- Various stability and performance improvements based on our prioritized GitHub bug backlog.

New features being released separately

New versions of the WinAppSDK Visual Studio Templates for C# and C++ are being released through the Visual Studio Marketplace and they will appear a few weeks after the release of 1.5. With the new version, the templates may now be released independently of WinAppSDK releases, giving us much more flexibility in getting updates to customers.

Other previously planned features

In 1.5, we made progress on the following features that we announced on our roadmap, but did not complete them. They will continue into the 1.6 timeframe.

- Tabbed windows
- Drag-n-drop support for WebView2
  - To clarify, in 1.5 dragging *into* WebView2 is supported, such as dragging a PNG from File Explorer into Bing for Visual Search. Work is ongoing in 1.6 to support dragging *out* of WebView2.
- Investigations into the table view and ink controls

Dynamic lighting has been removed from the roadmap for the time being.

Known issues

- When using libraries which contain resources such as .xaml files, you may hit an error message at runtime indicating that those resources cannot be found. In this case, it might be necessary to insert `<ShouldComputeInputPris>true</ShouldComputeInputPris>` in the project file to ensure those resources get included.
- Clicking on the chevron of a `NavigationViewItem` no longer correctly expands or collapses on a single click. Double-clicking still works, as does clicking elsewhere on the `NavigationViewItem`.

Bug fixes

- Fixed an issue where `StackPanel` applied spacing to collapsed items. For more info, see GitHub issue #916.
- Fixed problems with scrolling controls no longer working after closing another app window. Fore more info, see GitHub issues #9292 and #9355.
- Fixed a crash when setting `DebugSettings.EnableFrameRateCounter` to *true* before the first frame rendered. For more info, see GitHub issue #2835.
- Fixed a potential compile error for C++ where some headers did not include necessary dependencies. Note that the change of `#include` order might impact some apps, such as possibly causing a compile error for `IInspectable` if the app is using a version of C++/WinRT older than 2023. For more info, see GitHub issue #9014.
- Fixed an issue where `ElementName` bindings didn't work inside the `ItemsRepeater` `DataTemplate`. For more info, see GitHub issue #560.
- Fixed crashes when running an app under Visual Studio with the in-app toolbar enabled. Visual Studio 17.8 Preview 2 or later is required to fully get the fixes. For more info, see GitHub issue #8806.
- Fixed an issue where `AnnotatedScrollbar` could sometimes crash when quickly scrolling.
- Fixed an issue where menu text would sometimes get truncated.
- Fixed an issue where teaching tips did not receive proper focus. For more info, see GitHub issue #3257.
- Fixed an issue that crashed the application when setting the `TailVisibility` of a `TeachingTip` to *Collapsed* on startup. For more info, see GitHub issue #8731.
- Fixed an issue with how PRI files were handled when using libraries. For more info, see GitHub issue #8857.
- Fixed an issue from the 1.5-experimental2 release where the projection DLL was not generated. For more info, see GitHub issue #4152.
- Fixed an issue where the ellipsis button on the text formatting popup of the `RichEditBox` was not displaying the list of actions properly. For more info, see GitHub issue #9140.
- Fixed an issue where `ListView` didn't handle keyboard accelerators properly. For more info, see GitHub issue #8063.
- Fixed an access violation issue with using `AccessKey` to close a window. For more info, see GitHub issue #8648.
- Fixed a crash when using an `AccessKey` to close a window. For more info, see GitHub issue #9002.
- Fixed an issue affecting text alignment in a `MenuFlyoutItem` within a `MenuBar`. For more info, see GitHub issue #8755.

- Fixed an issue where highlighted text would not remain highlighted upon right-click. For more info, see GitHub issue #1801.
- Fixed an issue causing inactive windows to crash the app when closed. For more info, see GitHub issue #8913.
- Fixed an issue that could hang applications when scrolling with the middle mouse button and left-clicking immediately afterwards. For more info, see GitHub issue #9233.
- Fixed an issue causing apps to crash on startup when using a custom `NavigationViewItem`. For more info, see GitHub issue #8814.
- Fixed a `NavigationView` issue where the ellipsis button would incorrectly generate an error. For more info, see GitHub issue #8380.
- Fixed an issue where a `SystemBackdrop` would not render properly in a multi-window app. For more info, see GitHub issue #8423.
- Fixed a duplication issue when inserting into the beginning of an `ObservableCollection`. For more info, see GitHub issue #8370.

Related topics

- Latest preview channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Stable channel release notes for the Windows App SDK 1.4

Article • 05/23/2024

The stable channel provides releases of the Windows App SDK that are supported for use by apps in production environments. Apps that use the stable release of the Windows App SDK can also be published to the Microsoft Store.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Downloads for the Windows App SDK

> **Note**
>
> The Windows App SDK Visual Studio Extensions (VSIX) are no longer distributed as a separate download. They are available in the Visual Studio Marketplace inside Visual Studio.

Version 1.4.7 (1.4.240802001)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.4 release.

- Fixed an issue where `ItemsRepeater` would not generate enough items if it was in a `ShouldConstrainToRootBounds="false"` popup which was taller or wider than the main window.
- Fixed a potential crash on shutdown if an `AnnotatedScrollBar` label update was pending.

Version 1.4.6 (1.4.240512000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.4 release.

- Fixed a potential crash when processing input.
- Fixed an issue where a drag-and-drop operation that started from another app might not allow the correct Copy/Move/Link drop operations.
- Fixed WinUI source server information for debugging to properly point to the microsoft-ui-xaml GitHub repo.
- Fixed an issue with the fix for GitHub issue #8857 to properly merge a library's `resources.pri` into the app's `resources.pri`.

Version 1.4.5 (1.4.240211001)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.4 release.

- Fixed an issue that could hang applications when clicking a mouse button while scrolling with the mouse wheel. For more info, see GitHub issue #9233.
- Fixed an issue with duplicate assets when referencing a chain of NuGet packages. For more info, see GitHub issue #8857.
- Fixed several `BreadcrumbBar` issues including a memory leak, a crash when the ellipsis menu is empty, and the ellipsis menu being incorrectly constrained within the window.
- Fixed a potential crash on shutdown when releasing graphics resources.

Version 1.4.4 (1.4.231219000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.4 release.

- Fixed a WinUI 3 diagnostics security issue.
- Fixed an input issue where the password box didn't show the on-screen keyboard when activated via touch. For more info, see GitHub issue #8946.
- Fixed an issue that caused the `Microsoft.UI.Xaml.Controls.dll` file size to grow unexpectedly.
- Fixed a `CommandBarFlyout` issue that could cause crashes when setting focus.
- Updated Windows App SDK support for .NET 8 RID-specific asset handling.
- Fixed an issue causing some swapchains to be positioned or stretched incorrectly.

Version 1.4.3 (1.4.231115000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.4 release.

- Fixed an issue where a menu could appear without a background for a short period of time.
- Fixed a crash that may occur in specific multi-monitor scenarios.
- Fixed an issue where a context menu could appear off-screen.
- Fixed an issue with Window styles and maximizing behavior. For more info, see GitHub issue #8996.
- Fixed an issue with Islands where focus could be unexpectedly grabbed from another control.
- Fixed an issue with tab order on `NavigationView`.
- Fixed a rendering issue where a white bar might be visible at the top of the titlebar. For more info, see GitHub issue #8947.
- Various performance fixes.

Version 1.4.2 (1.4.231008000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.4 release.

- Fixed a crashing issue in explorer.exe caused by excessive memory and object allocation.
- Fixed a titlebar interaction issue that prevented the back button from working properly.
- Fixed an issue that caused a warning to be generated for a source file being included multiple times.
- Fixed an issue impacting context menu performance.
- Fixed a .lnk shortcut issue that made the target .exe always point to the same location for packages in the WindowsApps folder.
- Fixed a DWriteCore issue affecting proper rendering of Indic text in certain fonts.
- Fixed an issue in a List View that prevented proper keyboard navigation to and from nested selected items with *Tab/Shift + Tab*.
- Fixed an issue that broke scrolling ComboBox items by touch after expanding the ComboBox a second time. For more info, see GitHub issue #8831.
- Fixed an issue where WinAppSDK packages did not include WinUI's localized resources for some languages.
- Fixed an inconsistency between how File Explorer and XAML display a user's preferred language.
- Fixed a craftsmanship issue in File Explorer causing a thin line to show under the active tab.

- Fixed an issue where some framework-provided keyboard accelerators were not properly localized. For more info, see GitHub issue #2023.
- Fixed an issue with RepeatButton controls that were repeatedly scrolling when tapped.
- Fixed the WinAppSDK installer .exe to have proper resource version info.

Version 1.4.1 (1.4.230913002)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.4 release.

- Fixed performance issues to improve the time to first frame.
- Fixed an issue where menus didn't respect `RequestedTheme`. For example, it was possible for this issue to lead to white text on a white background. For more info, see GitHub issue #8756.
- Fixed an issue that caused acrylic backgrounds to sometimes become fully transparent in some menus.
- Fixed an issue where XAML sometimes caused Windows to unnecessarily repaint the desktop wallpaper.
- Fixed support for `TabNavigation = Local` and `TabNavigation = Cycle` for `ListView` and `GridView`, which now enables navigating between headers and items with TAB in addition to arrow keys.
- Fixed some noisy exceptions when dismissing a tooltip. For more info, see GitHub issue #8699.

New and updated features and known issues for version 1.4

The following sections describe new and updated features and known issues for version 1.4.

In an existing Windows App SDK 1.3 app, you can update your Nuget package to 1.4.230822000 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Latest Windows App SDK downloads.

Custom titlebar + AppWindow titlebar merger

The WinUI 3 custom titlebar uses the AppWindow titlebar implementation, along with the NonClientInputPointerSource APIs, under the hood in the Windows App SDK 1.4. As

a result, both titlebar implementations now behave the same way with the same features and limitations. This is fully backwards compatible in all supported cases - any app with a custom-defined titlebar will behave as before. But, it's now easier for WinUI 3 developers who might be new to custom titlebars to understand and use them by taking advantage of these new features:

- A better default scenario where the developer doesn't define a titlebar element specifically (replacing the fallback titlebar from WinUI 2)
- Distinct drag regions in the titlebar, enabling you to create multiple drag regions and place clickable controls on any part of the non-client area (titlebar area)
- App-wide draggable regions that can be put anywhere in the app or make the whole app draggable
- Better theming support that replaces resource-based theming
  - Since drag regions are transparent, they follow the app theme every time
- More customization: hide the min, max, and close buttons; place system icons in the titlebar; or have different regions act as caption buttons that receive NCHITTEST responses
- More developer freedom that enables you to mix and match with AppWindow titlebar APIs, such as using higher-level WinUI 3 APIs for most scenarios but with AppWindow APIs mixed in for lower-level control

Widgets updates

Three new interfaces have been added for Widget Providers to implement: `IWidgetProvider2`, `IWidgetProviderAnalytics`, and `IWidgetProviderErrors`. `IWidgetProvider2` allows providers to respond to the *Customize* action invoked by the user, which is identical to what is available for 1st party Widgets. The `IWidgetProviderAnalytics` and `IWidgetProviderErrors` interfaces are used by providers to gather telemetry for their widgets; analytics and failure events about widgets are communicated to the respective widget providers. The `WidgetCustomizationRequestedArgs`, `WidgetAnalyticsInfoReportedArgs`, and `WidgetErrorInfoReportedArgs` classes are used to communicate relevant information to support new functionalities.

XAML Islands no longer experimental

XAML Islands and the underlying ContentIslands platform are no longer experimental.

- Currently XAML Islands are only tested for use in C++ apps. This release does not include any convenient wrapper elements for use in WPF or WinForms.

- `DesktopWindowXamlSource` and related types have been added in the Microsoft.UI.Xaml.Hosting namespace for XAML Islands. `XamlRoot.ContentIslandEnvironment` was added to help access the underlying Island information for an element.
- Many new types have been introduced in the Microsoft.UI.Content namespace and the Microsoft.UI.Input namespace as the underlying support for XAML Islands or for using this ContentIslands functionality without XAML.
- A new `DragDropManager` (plus related types) has been added in the Microsoft.UI.Input.DragDrop namespace for Island scenarios.

ItemsView

We're introducing a new list control called the `ItemsView` and a corresponding concrete `ItemContainer` class. `ItemContainer` is a lightweight container with built-in selection states and visuals, which can easily wrap desired content and be used with `ItemsView` for a collection control scenario.

- The new `ItemsView` control displays a data collection. `ItemsView` is similar to the `ListView` and `GridView` controls, but is built using the `ItemsRepeater`, `ScrollView`, `ItemContainer` and `ItemCollectionTransitionProvider` components. It offers the unique ability to plug in custom `Layout` or `ItemCollectionTransitionProvider` implementations. Another key advantage is the ability to switch the layout on the fly while preserving items selection. The inner `ScrollView` control also offers features unavailable in `ListView`/`GridView`'s `ScrollViewer` control such as the ability to control the animation during programmatic scrolls.
  - A new `ItemTransitionProvider` property on `ItemsRepeater` (and the new `ItemsView` control) lets you specify an `ItemCollectionTransitionProvider` object to control transition animations on that control. A `CreateDefaultItemTransitionProvider` method has also been added to `Layout`, which enables a layout object to provide a fallback transition to accompany it if you do not provide one explicitly on the `ItemsView` control.
  - A new `IndexBasedLayoutOrientation` property on `Layout` where the layout orientation, if any, of items is based on their index in the source collection. The default value is `IndexBasedLayoutOrientation.None`. Custom layouts set this property by calling the new (protected) `SetIndexBasedLayoutOrientation` method.
  - A new `VisibleRect` property on `VirtualizingLayoutContext` gets the visible viewport rectangle within the `FrameworkElement` associated with the `Layout`. The protected virtual `VirtualizingLayoutContext.VisibleRectCore` method can be

overridden to provide the value that will be returned from the `VisibleRect` property.

- The new `LinedFlowLayout` class is typically used to lay out the items of the `ItemsView` collection control. It is particularly useful for displaying collection of pictures. It does so by laying them out from left to right, and top to bottom, in lines of equal height. The pictures fill a horizontal line and then wrap to a next line. Pictures may be cropped at the left and right edges to fit into a line. They may also be expanded horizontally and cropped at the top and bottom edges to fill a line when the stretching mode is employed.

New features from across the WinAppSDK

- A new `ThemeSettings` class that allows Win32 WinRT apps to detect when the system's High Contrast setting has changed, similar to UWP's AccessibilitySettings class. See the ThemeSettings API spec on GitHub for more information.
- `AccessKeyManager.EnterDisplayMode` is a new method to display access keys for the current focused element of a provided root. Access keys are in "display mode" when showing a key tip to invoke a command, such as pressing the Alt key in Paint to show what keys correspond to what controls. This method allows for programmatically entering display mode.
- `Application.ResourceManagerRequested` provides a mechanism to provide a different `IResourceManager` to resolve resource URIs for scenarios when the default `ResourceManager` won't work. For more information, see the Application.ResourceManagerRequested API spec on GitHub.
- The version of the WebView2 SDK was updated from 1661.34 to 1823.32.
- `Popup/FlyoutBase.IsConstrainedToRootBounds = false` is now supported, allowing a popup/flyout to extend outside the bounds of the parent window. A `SystemBackdrop` property has been added to these types to support having acrylic in these unconstrained popups. Menus by default use this to have acrylic.
- `Closed`, `FrameworkClosed`, and `IsClosed` have been added to `DesktopAcrylicController` and `MicaController` to improve handling during object/thread shutdown.
- `DesktopAcrylicController.Kind` can now be set to choose between some standard acrylic appearances.
- `DispatcherQueue` has some new events and helpers to facilitate better organized shutdown and for apps using Islands to easily run a standard supported event loop.
- `InputNonClientPointerSource` in the Microsoft.UI.Input namespace can be used for custom titlebar scenarios to define non-client area regions. Code can register for

corresponding events like hover and click events on these regions.

- `AppWindow` has some new helpers to get and associate with a `DispatcherQueue`.
- The new `TreeView.SelectionChanged` event allows developers to respond when the user or code-behind changes the set of selected nodes in the `TreeView` control.
- The new `ScrollView` control provides a new alternative to `ScrollViewer`. This new control is highly aligned in behavior and API with the existing `ScrollViewer` control, but is based on `InteractionTracker`, has new features such as animation-driven view changes, and is also designed to ensure full functionality of `ItemsRepeater`. See A more flexible ScrollViewer · Issue #108 · microsoft/microsoft-ui-xaml (github.com) for more details. Various new types, including `ScrollPresenter`, are part of the overall `ScrollView` model.
- The new `AnnotatedScrollBar` control extends a regular scrollbar's functionality by providing an easy way to navigate through a large collection of items. This is achieved through a clickable rail with labels that act as markers. It also allows for a more granular understanding of the scrollable content by displaying a tooltip when hovering over the clickable rail.

Known issues

- When using `ExtendsContentIntoTitleBar = true`, clicks at the top-left corner of the window by default always show the system window menu (Minimize/Close/etc.) rather than letting the pointer input through to the content of the window. This, for example, means that a Back button in that area of the titlebar will not work. A workaround for this issue is to set `AppWindow.TitleBar.IconShowOptions = Microsoft.UI.Windowing.IconShowOptions.HideIconAndSystemMenu` on the Window's AppWindow.
- There are some new continuable exceptions when hiding `ShouldConstrainToRootBounds="False"` popups/flyouts. That includes hiding tooltips, as reported here: Dismissing a tooltip throws 4 native exceptions · Issue #8699 · microsoft/microsoft-ui-xaml (github.com)
- In 1.4, the min/max/close caption buttons for `ExtendsContentIntoTitleBar = true` are now drawn by AppWindow rather than XAML. This is by design, but it can impact apps that were overriding XAML's internal styles to hide or do extra customization of these buttons, such as in this report: Cannot hide caption button on titlebar · Issue #8705 · microsoft/microsoft-ui-xaml (github.com)
- There was a breaking change in .NET 8 to how it handles the runtime identifier graph: [Breaking change]: Projects targeting .NET 8 and higher will by default use a smaller, portable RID graph. · Issue #36527 · dotnet/docs (github.com). Because of this issue and because .NET 8 has not officially released yet, the Windows App

SDK 1.4 does not officially support .NET 8. However, if you would still like to target the pre-release version of .NET 8 with this version of the App SDK, we recommend the following steps:

  - Setting UseRidGraph to true is recommended. You'll also need to update the `<RuntimeIdentifiers>` property in the `.csproj` file to `<RuntimeIdentifiers>win-x86;win-x64;win-arm64</RuntimeIdentifiers>`, as well as update each `Propeties\*pubxml` file to switch from `win10` to `win` in the `<RuntimeIdentifier>` property (for example, `<RuntimeIdentifier>win-x86</RuntimeIdentifier>`).

- With Windows App SDK 1.4, the target `GenerateDeploymentManagerCS` in `Microsoft.WindowsAppSDK.DeploymentManager.CS.targets` was renamed to `GenerateBootstrapCS`.
- `MenuFlyout` background doesn't use the application's requested theme:
  - MenuFlyoutItem text doesn't sync with system theme. · Issue #8678 · microsoft/microsoft-ui-xaml (github.com)
  - 1.4 preview: Flyout backdrop theme is derived from system theme and not from its associated element · Issue #8756 · microsoft/microsoft-ui-xaml (github.com)

Bug fixes

- Fixed an issue where calling the `Microsoft.Windows.AppLifecycle.AppInstance.Restart("")` API caused unpackaged apps to crash. For more info, see GitHub issue #2792.
- Fixed an installer crashing issue that was introduced in 1.4-experimental1. For more info, see GitHub issue #3760.
- Fixed an issue where text strikethrough was not removed properly in a TextBlock. For more info, see GitHub issue #1093.
- Fixed an issue causing incorrect Shift + Tab navigation in a Panel with *TabFocusNavigation* set to "Once." For more info, see GitHub issue #1363.
- Fixed an issue in C++/WinRT that prevented `{x:Bind}` from working properly with a property of a named XAML control. For more info, see GitHub issue #2721.
- Fixed a runtime AccessViolation issue in WinUI Desktop apps caused by setting `DebugSettings.EnableFrameRateCounter = true`. For more info, see GitHub issue #2835.
- Fixed an issue where `XamlTypeInfo.g.cpp` did not include needed headers. For more info, see GitHub issue #4907.
- Fixed a crashing issue caused by simultaneous multi-touch and mouse input. For more info, see GitHub issue #7622.
- Fixed an issue that prevented an active WinUI 3 app window from scrolling when the system setting to disable scrolling of inactive windows on mouse over was in effect. For more info, see GitHub issue #8764.

- Fixed a crash when trying to subclass `MediaPlayerElement`.
- Fixed some crash and memory leak issues in `TreeView`.
- Fixed an app hang issue that could happen when using keyboard to navigate in `RadioButtons`.
- Fixed a crash when using the keyboard to navigate in a `PipsPager`.
- Fixed WebView2 content to scale with the "Text size" Accessibility setting in the Settings app.
- Fixed a crash that could occur when animations were running when the display turned off.
- Fixed a performance issue introduced in 1.3 that added a ~10% overhead to first layout/render.

Related topics

- Latest preview channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Stable channel release notes for the Windows App SDK 1.3

Article • 05/23/2024

The stable channel provides releases of the Windows App SDK that are supported for use by apps in production environments. Apps that use the stable release of the Windows App SDK can also be published to the Microsoft Store.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Downloads for the Windows App SDK

> **Note**
>
> The Windows App SDK Visual Studio Extensions (VSIX) are no longer distributed as a separate download. They are available in the Visual Studio Marketplace inside Visual Studio.

Version 1.3.3 (1.3.230724000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.3 release.

- Fixed an issue where the mouse would sometimes stop working when a dialog box was closed.
- Fixed a deployment issue that prevented apps from installing due to a mismatch of package versions on the system. For more information, see GitHub issue #3740.
- Fixed an issue affecting context menu positioning in Windows App SDK 1.3.
- Fixed an issue causing some WinUI3 apps, in some situations, to crash when the app was closed because XAML shut itself down too early.

- Fixed an issue where font icons were not mirroring properly in right-to-left languages. For more information, see GitHub issue #7661.
- Fixed an issue causing an app to crash on shutdown when resources were torn down in a bad order. For more information, see GitHub issue #7924.

Version 1.3.2 (1.3.230602002)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.3 release.

- Fixed a crash when setting a Protected Cursor.
- Fixed a performance issue in XamlMetadataProvider during app startup. For more information, see GitHub issue #8281.
- Fixed an issue with hyperlinks and touch in a RichTextBlock. For more information, see GitHub issue #6513.
- Fixed an issue with scrolling and touchpads in WebView2. For more information, see GitHub issue #7772.
- Fixed an issue where an update of Windows App SDK sometimes required a restart of Visual Studio. For more information, see GitHub issue #3554.
- Fixed a noisy exception on shutdown when running in a debugger.

Version 1.3.1 (1.3.230502000)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.3 release.

- Fixed issue causing apps to crash when setting the SystemBackdrop if the Content was null. For more information, see GitHub issue #8416.
- Fixed issue causing apps to crash when setting the Window Title in XAML, a new capability added in 1.3.0. For more information, see GitHub issue #3689.
- Fixed issue where a window incorrectly took focus when its content changed.
- Fixed an issue with creating C++ projects with the WinAppSDK 1.3 project templates.
- Updated templates on Visual Studio Marketplace

New and updated features and known issues for version 1.3

The following sections describe new and updated features and known issues for version 1.3.

In an existing Windows App SDK 1.2 app, you can update your Nuget package to 1.3.230331000 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Latest Windows App SDK downloads.

XAML Backdrop APIs

With properties built in to the XAML Window, Mica & Background Acrylic backdrops are now easier to use in your WinUI 3 app. See the System Backdrop and Mica Backdrop API docs for more information about the Xaml Backdrop properties.

```
C#

public MainWindow()
{
    this.InitializeComponent();

    this.SystemBackdrop = new MicaBackdrop();
}
```

Window.AppWindow

By replacing several lines of boilerplate code, you're now able to use AppWindow APIs directly from a **Window** through Window.AppWindow.

New features from across WinAppSDK

- `ApplicationModel.DynamicDependency`: `PackageDependency.PackageGraphRevisionId` that replaces the deprecated MddGetGenerationId.
- Environment Manager: EnvironmentManager.AreChangesTracked to inform you whether changes to the environment manager are able to be tracked in your application.
- A new event, DebugSettings.XamlResourceReferenceFailed is now raised when a referenced Static/ThemeResource lookup can't be resolved. This event gives access to a trace that details where the framework searched for that key in order to better enable you to debug Static & ThemeResource lookup failures. For more information, see the Tracing XAML resource reference lookup failures API spec on GitHub.

Other updates

- See our WinAppSDK 1.3 milestone on the WinAppSDK GitHub for additional issues addressed in this release.
- See our WinUI 3 in WinAppSDK 1.3 milestone on the microsoft-ui-xaml GitHub for additional issues addressed in this release.
- With the latest experimental VSIX, you're now able to convert your app between unpackaged and packaged through the Visual Studio menu instead of in your project file.

Known issue

Due to a recent change to the xaml compiler, an existing project that upgrades to 1.3 may experience a build error like the following within Visual Studio:

Console

```
>
C:\Users\user\\.nuget\packages\microsoft.windowsappsdk\\**1.3.230331000**\bu
ildTransitive\Microsoft.UI.Xaml.Markup.Compiler.interop.targets(537,17):
error MSB4064: The "PrecompiledHeaderFile" parameter is not supported by the
"CompileXaml" task loaded from assembly: Microsoft.UI.Xaml.Markup.Compiler,
Version=1.0.0.0, Culture=neutral, PublicKeyToken=de31ebe4ad15742b from the
path:
C:\Users\user\\.nuget\packages\microsoft.windowsappsdk\\**1.2.230118.102**\t
ools\net472\Microsoft.UI.Xaml.Markup.Compiler.dll. Verify that the parameter
exists on the task, the <UsingTask> points to the correct assembly, and it
is a settable public instance property.
```

This is caused by Visual Studio using a cached xaml compiler task dll from 1.2, but driving it with incompatible MSBuild logic from 1.3, as seen in the error text above. The workaround is to shut down Visual Studio, restart it, and reload the solution.

Related topics

- Latest preview channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Stable channel release notes for the Windows App SDK 1.2

Article • 05/23/2024

The stable channel provides releases of the Windows App SDK that are supported for use by apps in production environments. Apps that use the stable release of the Windows App SDK can also be published to the Microsoft Store.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Downloads for the Windows App SDK

> **Note**
>
> The Windows App SDK Visual Studio Extensions (VSIX) are no longer distributed as a separate download. They are available in the Visual Studio Marketplace inside Visual Studio.

Version 1.2.5 (1.2.230313.1)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.2 release.

- Fixed issue causing apps to crash during Composition shutdown.
- Fixed issue causing apps to continue running animations even when the screen is off.
- Fixed issue causing mouse and touch input to fail in WebView2 when mouse and keyboard input occurred simultaneously. For more information, see GitHub issue #3266.

Version 1.2.4 (1.2.230217.4)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.2 release.

- Fixed issue causing self-contained apps to not be able to set UAC Settings. For more information, see GitHub issue #3376.
- Fixed issue causing push notifications to return an inaccurate Expiration time with `PushNotificationChannel::ExpirationTime`. For more information, see GitHub issue #3300.
- Fixed issue causing negative numbers to be considered "invalid" when passing a double as a parameter into an x:Bind function.
- Several fixes to update the WinUI VSIX. These updates included simplifying the project template dipAwareness in app.manifest, removing the UWP templates, updating localized resource files, adding the phone id to unblock store submission, and removing the copyright notice and license. For more info see GitHub issues #5659, #3205, #3323, #3322, #3143.

Version 1.2.3 (1.2.230118.102)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.2 release.

- Fixed issue causing WinUI 3 apps to crash when multiple windows are closed.
- Fixed issue causing a crash on app close when two or more references to the ThreadPoolTimer interface are called. For more information, see GitHub issues #7260 and #7239.
- Fixed issue causing all Single-project MSIX apps to run as full trust. For more information, see GitHub issue #7766.

Version 1.2.2 (1.2.221209.1)

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.2 release.

- Fixed issue that caused the Store and side-load packages (e.g. from installer, NuGet, and bootstrapper) to fail to install if the other is already installed. For more information, see GitHub issue #3168.
- Fixed issue causing missing elasticity effects and animation curves when scrolling with a touchpad. For more information, see GitHub issue #7874.
- Fixed issue in ListView causing memory leaks.
- Fixed issue causing the Button template to not respect the Foreground property after mouse hover. For more information, see GitHub issue #7208.

- Fixed issue causing an unneeded exception when there is no MediaPlaybackItem in a MediaElement.
- Fixed issue causing a white frame to appear in MediaPlayerElement on content transitions.
- Fixed additional issues causing App.UnhandledException to not catch exceptions from other threads. For more information, see GitHub issues #1259 and #5221.

Version 1.2.1 (1.2.221116.1)

This is a servicing release of the Windows App SDK that includes a critical bug fix for the 1.2 release.

Fixed issue that caused a crash on startup in C++ WinUI 3 apps when adding a WebView2 or TextBox control. For more information see GitHub issues #7911 & #3117.

new and updated features and known issues for version 1.2

The following sections describe new and updated features and known issues for 1.2.

> **Note**
>
> Visual Studio 2019 and .NET 5 is no longer supported for building C# apps (see Windows App SDK 1.2 moving to C# WinRT 2.0). You will need Visual Studio 2022 and one of the following .NET SDK versions: 6.0.401 (or later), 6.0.304, 6.0.109. When released, WinAppSDK 1.2 will support .NET 7 as well.
>
> To update your .NET SDK version, install the latest version of Visual Studio 2022 or visit .NET Downloads. When updating your NuGet package without the required .NET SDK version, you will see an error like: "*This version of WindowsAppSDK requires .NET 6+ and WinRT.Runtime.dll version 2.0 or greater.*". To update the project from .NET 5.0 to .NET 6.0, open the project file and change "TargetFramework" to `net6.0` and "Target OS version" to the appropriate value (such as `net6.0-windows10.0.19041.0`).

Third-party Widgets in Windows

The widgets board was first introduced in Windows 11 and was limited to displaying built-in widgets. Widgets are small UI containers that display text and graphics on the widgets board, and are associated with an app installed on the device. With Windows App SDK, as third party developers you can now create widgets for your packaged Win32 apps and test them locally on the Windows 11 widgets board.

For more information about widgets, check out Widgets Overview.

To get started developing widgets for your app, check out the Widget service providers development docs and Widgets design fundamentals for prerequisites, guidance and best practices.

Prerequisites for this release include:

- Developer mode enabled on the development machine.
- The development machine is running a version of Windows from the Dev Channel of Windows Insider Preview (WIP) that is greater than or equal to 25217 with widgets board version 521.20060.1205.0 or above.

Known limitations when developing Widgets:

- Third-party Widgets can only be tested locally on devices enrolled in WIP for this preview release.
- Widgets can only be created for packaged, Win32 apps. Widgets for Progressive Web Apps (PWA) are planned to be supported as part of Microsoft Edge 108.

DisplayInformation

Windows desktop apps can now support High Dynamic Range (HDR) and Auto Color Management (ACM) through the DisplayInformation class in WinAppSDK. The DisplayInformation class enables you to monitor display-related information for an application view. This includes events to allow clients to monitor for changes in the application view affecting which display(s) the view resides on, as well as changes in displays that can affect the application view.

WinUI 3

WinUI 3 apps can play audio and video with the **MediaPlayerElement** and **MediaTransportControls** media playback controls. For more info on how and when to use media controls, see Media players.

WinUI 3 has been updated with the latest controls, styles, and behaviors from WinUI 2.8. These updates include the addition of the **InfoBadge** control, improvements to

accessibility and high contrast mode, as well as bug fixes across controls. For more details, see the release notes for WinUI 2.7 and WinUI 2.8.

Fixed issues:

- Acrylic backdrop material with DesktopAcrylicController is now supported in Windows 10 apps. For more information, check out issue 7112 on GitHub.
- Fixed various issues that caused routing of App.UnhandledException to fail. For more information, check out issue 5221 on GitHub. Regarding the remaining issues, workarounds are documented at the following GitHub issues and will be resolved in a future 1.2 release:
  - App_UnhandledException's UnhandledExceptionEventArgs.Exception can only be fetched once - should be cached
  - Do_Abi_* event/callback handlers should wrap invocations with RoReportUnhandledError
- Fixed issue causing ListView styles to regress and change from WinAppSDK 1.1. For more information, check out issue 7666 on GitHub.
- Fixed issue causing the incorrect Mica fallback background color to appear when the app is inactive. For more information, check out issue 7801 on GitHub.

Known limitations:

- When creating a new WinUI 3 project with Visual Studio 2022 17.4.0, it will reference a preview version of the WinAppSDK. Use NuGet Package Manager to update the reference to this release.
- Setting MediaPlayerElement.Source to relative URI (ms-appx/ms-resource) fails in unpackaged apps. The recommended workaround is to convert the relative ms-appx:/// URI to a fully resolved file:/// URI.

Trimming for apps developed with .NET

.NET developers can now publish trimmed WinAppSDK apps. With CsWinRT 2.0, the C#/WinRT projections distributed in WinAppSDK are now trimmable. Publishing your app trimmed can reduce the disk footprint of your app by removing any unused code from trimmable binaries. Apps may also see a startup performance improvement. With a basic Hello World app, we have seen a ~80% disk footprint improvement and a ~7% startup performance improvement when published trimmed. With WinUI gallery, we have seen a ~45% disk footprint improvement.

For more details on how to enable trimming, trimming limitations (such as reflection against trimmable types), and trim warnings, see Trim self-contained deployments and

executables. Developers should thoroughly test their apps after trimming to ensure everything works as expected. For more information, check out issue 2478 on GitHub.

Support for Visual Studio Arm64

As early as Project Reunion (now WinAppSDK) 0.5, apps developed with WinAppSDK were able to run on Arm64. Starting with Visual Studio 17.3 Preview 2, you can develop native applications with WinAppSDK on Arm64 devices.

To get started developing on an Arm64 device, see Windows on Arm and Arm64 Visual Studio.

Notifications

AppNotificationBuilder introduced as an alternative to XML payload for creating and defining App Notifications.

For usage information, see the AppNotificationBuilder spec on GitHub.

Also see Quickstart: App notifications in the Windows App SDK for an example of how to create a desktop Windows application that sends and receives local app notifications.

Breaking change:

For push notifications, when making a channel request call, apps will need to use the Azure Object ID instead of the Azure App ID. See Quickstart: Push notification in the Windows App SDK for details on finding your Azure Object ID.

Fixed issue:

PushNotificationManager.IsSupported will perform a check for elevated mode. It will return `false` if the app is elevated.

Known limitations (Notifications):

- In **AppNotificationScenario**, `Urgent` is only supported for Windows builds 19041 and later. You can use **AppNotificationBuilder.IsUrgentScenarioSupported** to check whether the feature is available at runtime.
- In **AppNotificationButton**, `hint-toolTip` and `hint-buttonStyle` are only supported for builds 19041 and later. You can use **IsButtonStyleSupported** and **IsToolTipSupported** to check whether the feature is available at runtime.
- In **MediaPlayerElement**, when used in XAML markup for an unpackaged app, the **Source** property cannot be set with an ms-appx or ms-resource URI. As an alternative, set the Source using a file URI, or set from code.

Windowing

Full title bar customization is now available on Windows 10, version 1809 and later through the **AppWindowTitleBar** class. You can set **AppWindowTitleBar.ExtendsContentIntoTitleBar** to `true` to extend content into the title bar area, and **SetDragRectangles** to define drag regions (in addition to other customization options).

If you've been using the **AppWindowTitleBar.IsCustomizationSupported** property to check whether you can call the **AppWindowTitleBar** APIs, it now returns `true` on supported Windows App SDK Windows 10 versions (1809 and later).

Known limitations (Windowing):

Basic title bar customizations are not supported on Windows 10. These include **BackgroundColor**, **InactiveBackgroundColor**, **ForegroundColor**, **InactiveForegroundColor** and **IconShowOptions**. If you call these properties, they will be ignored silently. All other **AppWindowTitleBar** APIs work in Windows 10, version 1809 and later. For the caption button color APIs (among others) and **Height**, **ExtendsContentIntoTitleBar** must be set to `true`, otherwise they will also be ignored silently.

Access control

Introduced security.accesscontrol.h with the **GetSecurityDescriptorForAppContainerNames** function to ease and streamline named object sharing between packaged processes and general Win32 APIs. This method takes a list of Package Family Names (PFNs) and access masks, and returns a security descriptor. For more information, see the GetSecurityDescriptorForAppContainerNames spec on GitHub.

Other limitations and known issues

> **Important**
>
> When you reference WinAppSDK 1.2 from a project you might see an error similar to: "*Detected package downgrade: Microsoft.Windows.SDK.BuildTools from 10.0.22621.1 to 10.0.22000.194.*", which is caused by incompatible references to the package from the app project and the WinAppSDK package. To resolve this you can update the reference in the project to a more recent and compatible version of Microsoft.Windows.SDK.BuildTools.

- Unit tests may fail with a `REGDB_E_CLASSNOTREG` error in the Tests output pane in Visual Studio. As a workaround, you can add `<WindowsAppContainer>true</WindowsAppContainer>` to your project file.
- .NET PublishSingleFile isn't supported.
- Bootstrapper and Undocked RegFree WinRT auto-initializer defaults is (now) only set for projects that produce an executable (OutputType=Exe or WinExe). This prevents adding auto-initializers into class library DLLs and other non-executables by default.
  - If you need an auto-initializer in a non-executable (e.g. a test DLL loaded by a generic executable that doesn't initialize the Bootstrapper) you can explicitly enable an auto-initializer in your project via `<WindowsAppSdkBootstrapInitialize>true</WindowsAppSdkBootstrapInitialize>` or `<WindowsAppSdkUndockedRegFreeWinRTInitialize>true</WindowsAppSdkUndockedRegFreeWinRTInitialize>`.
- Microsoft.WindowsAppRuntime.Release.Net.dll is always the Arm64 binary and does not work for x86 and x64 apps. When explicitly calling the Bootstrap API do not use the Microsoft.WindowsAppRuntime.Release.Net.dll assembly. As a workaround you can include version constants in this source file distributed with the NuGet package: '..\include\WindowsAppSDK-VersionInfo.cs' or use the auto-initializer.

Related topics

- Latest preview channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Stable channel release notes for the Windows App SDK 1.1

Article • 05/23/2024

The stable channel provides releases of the Windows App SDK that are supported for use by apps in production environments. Apps that use the stable release of the Windows App SDK can also be published to the Microsoft Store.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Downloads for the Windows App SDK

> **Note**
>
> The Windows App SDK Visual Studio Extensions (VSIX) are no longer distributed as a separate download. They are available in the Visual Studio Marketplace inside Visual Studio.

Version 1.1

The latest available release of the 1.1.x lineage of the stable channel of the Windows App SDK is version 1.1.5. 1.1.x supports all stable channel features (see the **Features available by release channel** section in Windows App SDK release channels).

Version 1.1.5

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.1 release.

Bug fixes (1.1.5)

- Fixed issue where Acrylic does not work if Mica is enabled. For more information, see issue 7200 on GitHub.
- Fixed issue causing apps that depend on the WindowsAppRuntime installer (e.g. unpackaged apps) to fail to run on Windows 10 ARM64 machines. For more information, see issue 2564 on GitHub.

Version 1.1.4

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.1 release.

Bug fixes (1.1.4)

- Fixed regression from 1.0.x causing ListView, TreeView, and other 'List' controls to crash when scrolling with many items. For more information, see issue 7230 on GitHub.
- Fixed issue with DispatcherQueue causing enqueued callbacks to no longer be invoked.
- Fixed issue causing an app crash when calling `DeploymentManager.Initialize` multiple times in same app session.
- Fixed issue causing C# apps to fail to build on Arm64 Visual Studio. For more information, see issue 7140 on GitHub.
- Fixed intermittent crash in XAML imaging code due to incorrect failure handling.
- Fixed memory leak issue when attaching an event handler in ItemsRepeater with a parent UserControl. For more information, see issue 6123 on GitHub.
- Fixed issue causing a build failure in Visual Studio 17.3 when an app project is configured to enable automatic updates of its package when it's sideloaded (i.e. .appinstaller). For more information, see issue 2773.
- Fixed issue causing Store-distributed packaged apps that call Initialize (necessary e.g. for Push) to call it redundantly as DeploymentManager::GetStatus returned `Package Install Needed` when main and singleton packages are already installed. This caused a perf degradation on app launch.
- Fixed issue causing an exception in single instance apps when the cleanup event was intended to be ignored if it couldn't be opened. For more information, see the PR on GitHub.

Version 1.1.3

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.1 release.

Bug fixes (1.1.3)

- Fixed related set of issues where XAML crashes when including a ProgressBar, ProgressRing, PipsPager, PersonPicture, or Expander control in the first page of your app. For more information see issue 7164 on GitHub.
- Fixed issue causing the x64 installer to fail to install the Windows App SDK runtime. For more information see issue 2713 on GitHub.
- Fixed issue causing the WindowsAppRuntime to fail to install if a higher version of the runtime is installed. For more information see discussion 2708 on GitHub.

Version 1.1.2

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.1 release.

Bug fixes (1.1.2)

- Fixed issue where XAML crashes when closing a window while a dialog is open. For more information see issue 1032 on GitHub.
- Added `<auto-generated>` tag in C# files to prevent StyleCop warnings. For more information see issue 4526 on GitHub.
- Fixed issue causing an access violation error and crash when calling MddBootstrapInitialize when the matching framework package isn't installed. For more information see issue 2592 on GitHub.
- Fixed issue where the C# WinUI 3 item templates were missing in Visual Studio. For more information see issue 7148 on GitHub.
- Fixed issue where the WindowsAppRuntime installer fails when run as System user. For more information see issue 2546 on GitHub.

Version 1.1.1

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.1 release.

Bug fixes (1.1.1)

- Fixed issue causing apps to sometimes crash during a drag and drop operation. For more information see issue 7002 on GitHub.
- Fixed issue causing the title bar to disappear when switching AppWindowPresenterKind from FullScreen to Default.

- Fixed issue where Bootstrapper APIs like `ApiInformation.IsPropertyPresent` and `ApiInformation.IsMethodPresent` would cause unhandled exceptions in apps that aren't packaged. For more information see issue 2382 on GitHub.
- Fixed issue causing app freeze when maximizing application with pen input.

New and updated features and known issues for 1.1

The following sections describe new and updated features, limitations, and known issues for 1.1.

> **Note**
>
> For C# developers, one of the following .NET SDK versions (or later) is required: 6.0.202, 6.0.104, 5.0.407, 5.0.213. To update your .NET SDK version, visit .NET Downloads or update to the latest version of Visual Studio. Without the required .NET SDK version, when updating your NuGet package you will see an error like: *"This version of WindowsAppSDK requires WinRT.Runtime.dll version 1.6 or greater."*.

App Lifecycle & Restart

Apps are now able to initiate an explicit restart with specific arguments & state building off of the existing RegisterApplicationRestart API to register with the OS to restart in update, hang & reboot scenarios.

New features:

- Any packaged or unpackaged desktop app can terminate and restart itself on command, and have access to an arbitrary command-line string for the restarted instance using the `AppInstance.Restart()` API.
  - This is a lifted and synchronous version of the UWP `RequestRestartAsync()` API which enables restarting with arguments and returns an `AppRestartFailureReason` if the restart is unsuccessful.
  - Check out the Restart API docs on GitHub for usage & reference information.

WinUI 3

WinUI 3 is the native user experience (UX) framework for Windows App SDK. This release includes new features from WinAppSDK 1.0 as well as several stability improvements from 1.0 & 1.1 preview releases.

New features:

- Mica and Background Acrylic are now available for WinUI 3 applications.
  - For more information about these materials, check out Materials in Windows 11. Check out our sample code for applying Mica in C++ applications at Apply Mica or Acrylic materials in desktop apps for Windows 11 and in C# applications on GitHub as part of the WinUI 3 Gallery.
- First introduced in 1.0.1, we have stabilized and enabled the creation of **multiple windows on the same thread** in WinUI 3 applications. See issue 5918 for more information.

Fixed bugs:

- Fixed issue when using Mica where the app would crash when a window is divided equally by two screens. See issue 7079 on GitHub for more info.
- Fixed issue causing C# apps with WebView2 to crash on launch when the C/C++ Runtime (CRT) isn't installed by upgrading the WebView2 SDK from 1020.46 to 1185.39.
- Fixed issue causing some rounded corners to show a gradient when they should be a solid color. For more information see issue 6076 & issue 6194 on GitHub.
- Fixed issue where updated styles were missing from generic.xaml.
- Fixed layout cycle issue causing an app to crash when scrolling to the end of a ListView. For more information see issue 6218 on GitHub.
- Fixed issue where users are unable to drop an element when drag-and-drop is enabled. For more information see issue 7008 on GitHub.

Known limitations:

- When using a custom title bar, the caption controls do not change color on theme change.
- XAML crashes when a user closes a window while a dialog is open.

Deployment

New features:

- Packaged apps can now force deploy the Windows App SDK runtime packages using the **DeploymentManager.Initialize(DeploymentInitializeOptions) API** or using the --force option with the Windows App Runtime installer.
- There are additional functional extension categories, UnlockedDEHs, available for packaged apps. Check out the 1.1 Preview 3 release notes for more details. These require the Windows App SDK framework package to be installed. See Latest Windows App SDK downloads to install the runtime.

- Self-contained deployment is supported. Check out the Windows App SDK deployment overview for the differences between framework-dependent and self-contained deployment, and how to get started.
- The Bootstrapper API required for apps that don't deploy with MSIX include new options for improved usability and troubleshooting. Please view our documentation for C# apps, Bootstrapper C# APIs and for C++ apps, mddbootstrapheader.h header. For more details, see Use the Windows App SDK runtime for apps packaged with external location or unpackaged.

Known limitations:

- Running the Windows App Runtime installer (WindowsAppRuntimeInstall.exe) requires sideloading to be enabled. See issue 2469 on GitHub for more information.
- Creating an MSIX package through the Visual Studio Project menus can crash Visual Studio in some scenarios. This issue will be fixed in Visual Studio version 17.3 Preview 2 and serviced to 17.2. If you encounter this issue, you can work around it by generating an MSIX from the command line, switching to an unpackaged project, or reverting back to Windows App SDK 1.0.
- Self-contained applications packaged with MSIX are unsupported on 1809 causing app crash on launch.

Elevation

Apps are now able to run with elevated privilege.

Known limitations:

- Elevated support requires the following OS servicing update:
  - Win11 - May 10, 2022—KB5013943 (OS Build 22000.675)
  - Win10 - May 10, 2022—KB5013942 (OS Builds 19042.1706, 19043.1706, and 19044.1706)
- App and Push Notifications for an elevated unpackaged app is not supported.
- Elevated WinUI 3 apps crash when dragging an element during a drag-and-drop interaction.

Environment Variable Manager

Environment Variable Manager is a new API introduced in Windows App SDK 1.1. Environment Variable Manager allows developers to access, and modify environment variables in the process, user, and machine scope from one API surface.

If Environment Variable Manager is used from a packaged application, all environment variable operations are recorded. When the package is removed all environment variable operations are reverted.

New features:

- Get and set environment variables in the process, user, and machine scope.
- Automatic environment variable revert when a package that uses environment variable manager is removed.
- Includes specific APIs for PATH and PATHEXT.

Known limitations:

- Only available on Windows 11

MRT Core

MRT Core is a streamlined version of the modern Windows Resource Management System that is distributed as part of the Windows App SDK.

Fixed issues:

- An issue causing resources not to be indexed by default when a resource file is added using the VS UI is fixed in .NET SDK 6.0.300. If using an older .NET SDK version, please continue to use the workaround documented in 1.0's release notes. See issue 1786 on GitHub for additional information.
- An issue causing the resource URI to not be built correctly in unpackaged C++ WinUI 3 apps was fixed in Visual Studio 2022 17.2. If using an older Visual Studio version, please update Visual Studio to 17.2 to receive this fix.

Known limitations:

- In .NET projects, resource files copy-pasted into the project folder aren't indexed on F5 if the app was already built. As a workaround, rebuild the app. See issue 1503 on GitHub for more info.

For more information, see Manage resources with MRT Core.

Notifications

Developers of packaged (including packaged with external location) and unpackaged apps can now send Windows notifications.

New features:

- Support for app notifications for packaged and unpackaged apps.
  - Developers can send app notifications, also known as toast notifications, locally or from their own cloud service. See App notifications overview.
- Support for push notifications for packaged and unpackaged apps.
  - Developers can send raw notifications and app notifications from their own cloud service. See Push notifications overview.

Known limitations:

- Sending notifications from an elevated app is not supported. `PushNotificationManager::IsSupported()` will not perform a check for elevated mode.

Windowing

For easier programming access to functionality that's implemented in USER32.dll (see Windows and messages), this release surfaces more of that functionality in AppWindow itself.

New features:

- Apps with existing windows have more control over how a window is shown, by calling `AppWindow.ShowOnceWithRequestedStartupState`—the equivalent of `ShowWindow(SW_SHOWDEFAULT)`.
- Apps can show, minimize, or restore a window while specifying whether the window should be activated or not at the time the call is made.
- Apps can now determine specific dimensions for their window's client area size in Win32 coordinates without needing to calculate the non-client area sizing to get a specific client area size.
- Additional WinRT APIs are available to support z-order management of windows based off of SetWindowPos's hWndInsertAfter functionality.
- Apps drawing custom title bars with AppWindowTitleBar.ExtendsContentIntoTitleBar can set a `PreferredTitleBarHeight` option. You now have a choice between a standard height title bar, or a tall title bar that provides more room for interactive content. See Title bar in the Fluent design guidelines for advice about when to use a tall title bar.

Fixed issues:

- When full screen presenter is invoked the first time, the window now fits on the entire screen correctly. See issue 1853 on GitHub for more info.

- Windows created with `AppWindow::GetFromWindowId` have the `OverlappedPresenter` as the default presenter but does not have restrictions in terms of changes to window styles coming from other APIs. Windows created with AppWindow::Create will have the default Presenter guardrails in place from the start. See issue 2049 on GitHub for more info.
- Using the `OverlappedPresenter.SetBorderAndTitlebar` API to hide caption buttons and borders would result in a 1px top border when maximized. This has been resolved. See issue 1693 on GitHub for more info.

Known limitations:

- When using the **AppWindowTitlebar** API to customize the colors of the standard title bar, the icon and text is misaligned compared to the standard title bar. For more info, see GitHub issue 2459.

- When solving GitHub issue 2049 (seen above), we introduced the following bug: if you apply any **AppWindowPresenter** to an **AppWindow** that you've retrieved from **GetFromWindowId**, then change a window style that's being tracked by that Presenter through calling USER32 APIs, and then try to revert back to the window's previous state by re-applying the default Presenter, the result is a window that has no title bar. If you rely on any Presenter in your app, and use calls to USER32 for changing window styles at the time that a non-default Presenter is applied, then you might need to add a workaround to ensure correct window behavior until this bug is serviced. You can use the following code snippet as a template for how to work around the issue:

```
C#

AppWindow m_appWindow;
OverlappedPresenter m_defaultPresenter;

private void EnterFullScreen_Click(object sender, RoutedEventArgs e)
{
    // Capture the default presenter.
    m_defaultPresenter = m_appWindow.Presenter as OverlappedPresenter;

    // Opt in the default overlapped presenter so it can control
various aspects of the AppWindow.
    m_defaultPresenter.IsAlwaysOnTop =
m_defaultPresenter.IsAlwaysOnTop;
    m_defaultPresenter.IsResizable = m_defaultPresenter.IsResizable;
    m_defaultPresenter.IsMinimizable =
m_defaultPresenter.IsMinimizable;
    m_defaultPresenter.IsMaximizable =
m_defaultPresenter.IsMaximizable;

m_defaultPresenter.SetBorderAndTitleBar(m_defaultPresenter.HasBorder,
```

```
m_defaultPresenter.HasTitleBar);

    m_appWindow.SetPresenter(AppWindowPresenterKind.FullScreen);
}

private void ExitFullScreen_Click(object sender, RoutedEventArgs e)
{
    m_appWindow.SetPresenter(AppWindowPresenterKind.Default);
}
```

C#/WinRT

C# Windows Runtime Components, including WinUI custom controls, are now supported. This enables component authors to distribute C#-authored runtime components to any WinRT compatible language (e.g., C++/WinRT). See Walkthrough—Create a C# component with WinUI 3 controls, and consume it from a C++/WinRT app that uses the Windows App SDK and the sample on GitHub to get started.

Other limitations and known issues

- Apps that reference a package that depends on WebView2 (like Microsoft.Identity.Client) fail to build. This is caused by conflicting binaries at build time. See issue 2492 on GitHub for more information.
- Using `dotnet build` with a WinAppSDK C# class library project may see a build error "Microsoft.Build.Packaging.Pri.Tasks.ExpandPriContent task could not be loaded". To resolve this issue set `<EnableMsixTooling>true</EnableMsixTooling>` in your project file.
- The default WinAppSDK templates note that the MaxVersionTested="10.0.19041.0" when it should be "10.0.22000.0". For full support of some features, notably UnlockedDEHs, update the MaxVersionTested to "10.0.22000.0" in your project file.

Related topics

- Latest preview channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Stable channel release notes for the Windows App SDK 1.0

Article • 05/23/2024

The stable channel provides releases of the Windows App SDK that are supported for use by apps in production environments. Apps that use the stable release of the Windows App SDK can also be published to the Microsoft Store.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Downloads for the Windows App SDK

> **Note**
>
> The Windows App SDK Visual Studio Extensions (VSIX) are no longer distributed as a separate download. They are available in the Visual Studio Marketplace inside Visual Studio.

Version 1.0.4

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.0 release.

Bug fixes (1.0.4)

- Fixed issue causing AppBars, when used as Page.TopAppBar or Page.BottomAppBar to not render on screen.
- Fixed issue where apps with a package name of 12 characters or less that use a WinUI control from MUXControls.dll will immediately crash. For more information, see issue 6360 on GitHub.

- Fixed touch input issues causing problems with keyboard shortcuts and other scenarios. For more information, see issue 6291 on GitHub.
- Fixed issue causing apps packaged with MSIX or deployed as self-contained to fail to deploy.
- Fixed issue causing apps to sometimes crash during a drag and drop operation. For more information see issue 7002 on GitHub.

Version 1.0.3

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.0 release.

Bug fixes (1.0.3)

- Fixed issue causing C# apps with WebView2 to crash on launch when the C/C++ Runtime (CRT) isn't installed.
- Fixed touch input issues causing problems with keyboard shortcuts and other scenarios. For more information, see issue 6291 on GitHub.

Note: We don't usually add functionality in a servicing release, but this release's WebView2 fix required us to update to the latest version of the WebView2 SDK (1020.46 to 1185.39). See Release Notes for the WebView2 SDK for additional information on WebView2 1.0.1185.39 and Distribute your app and the WebView2 Runtime for additional information on the WebView2 Runtime.

Version 1.0.2

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 1.0 release.

Bug fixes (1.0.2)

- Fixed layout cycle issue causing an app to crash when scrolling to the end of a ListView. For more information see issue 6218 on GitHub.
- Fixed issue causing C# apps to crash on launch when the C/C++ Runtime (CRT) isn't installed. However, the CRT is still required for C# apps using WebView2. For more information see issue 2117 on GitHub.
- Fixed issue where applications with Single-project MSIX did not generate a .appinstaller file. For more information see issue 1821 on GitHub.
- Fixed issue where WinUI applications did not support .NET 6 `dotnet build`.

Version 1.0.1

This is a servicing release of the Windows App SDK that includes critical bug fixes and multi-window support for the 1.0 release.

Bug fixes (1.0.1)

- Fixed issue causing the MddBootstrapAutoinitializer to not compile with enabled ImplicitUsings. For more information see issue 1686 on GitHub.
- Fixed issue where focus in WebView2 would be unexpectedly lost causing input and selection issues. For more information, see issue 5615 & issue 5570 on GitHub.
- Fixed issue causing the in-app toolbar in Visual Studio to be unclickable when using a custom title bar in a WinUI 3 app.
- Fixed issue causing Snap Layout to not appear when using a custom title bar in a WinUI 3 app. For more information, see issue 6333 & issue 6246 on GitHub.
- Fixed issue causing an exception when setting Window.ExtendsContentIntoTitleBar property when Window.SetTitlebar has been called with a still-loading UIElement.
- Fixed issue where Single-project MSIX apps did not support `dotnet build`.
- Fixed issue causing unpackaged apps to not install after installing a packaged app. For more information, see issue 1871 on GitHub.
- Fixed issue reducing performance during mouse drag operations.
- Fixed crash when calling GetWindowIdFromWindow() in unpackaged apps. For more information, see discussion 1891 on GitHub.

The limitations and known issues for version 1.0 also apply to version 1.0.1.

Additionally, for apps with custom title bars, we have made changes in this release (and fixed numerous issues) that include fixes to the glass window used for drag&drop operations. The recommendation is to use the default values and behaviors (give them a try!). If your title bar used margins so that the default caption buttons were interactive, we recommend visualizing your drag region by setting the background of your title bar to red and then adjusting the margins to extend the drag region to the caption controls.

New features

We have stabilized and enabled the creation of **multiple windows on the same thread** in WinUI 3 applications. See issue 5918 for more information.

Version 1.0

The following sections describe new and updated features, limitations, and known issues for 1.0.

WinUI 3

WinUI 3 is the native user experience (UX) framework for Windows App SDK. In this release we've added multiple new features from Windows App SDK 0.8 and stabilized issues from 1.0 Preview releases.

New features and updates:

- We've added new controls (PipsPager, Expander, BreadcrumbBar) and updated existing controls to reflect the latest Windows styles from WinUI 2.6.
- Single-project MSIX packaging is supported in WinUI by creating a new application using the "Blank App, Packaged..." template.
- We now support deploying WinUI 3 apps that aren't packaged on Windows versions 1809 and above. Please view Create your first WinUI 3 (Windows App SDK) project for additional information.
- WinUI 3 projects can now set their target version down to Windows 10, version 1809. Previously, they could only be set as low as version 1903.
- In-app toolbar, Hot Reload, & Live Visual Tree for WinUI packaged apps are supported in Visual Studio 2022 Preview 5 and GA.

Important limitations:

- Known issues for **both packaged and unpackaged WinUI applications**:
  - *Run-time error in C++ or C# apps that reference a C++ Windows Runtime Component:*
    - To resolve, add the below target to the end of the Windows Runtime Component's .vcxproj:

XML

```
<Target Name="GetPriIndexName">
<PropertyGroup>
    <!-- Winmd library targets use the default root namespace of the
project for the App package name -->
    <PriIndexName Condition="'$(RootNamespace)' !=
''">$(RootNamespace)</PriIndexName>
    <!-- If RootNamespace is empty fall back to TargetName -->
    <PriIndexName Condition="$(PriIndexName) == ''">$(TargetName)
</PriIndexName>
</PropertyGroup>
</Target>
```

    - The expected error will be similar to *WinRT originate error - 0x80004005 : 'Cannot locate resource from 'ms-appx:///BlankPage.xaml'.'.*

- Known issues for **WinUI applications with Single-project MSIX** (Blank App, Packaged template):
    - *Missing Package & Publish menu item until you restart Visual Studio:* When creating a new app with Single-project MSIX in both Visual Studio 2019 and Visual Studio 2022 using the Blank App, Packaged (WinUI 3 in Desktop) project template, the command to publish the project doesn't appear in the menu until you close and re-open Visual Studio.
    - A C# app with Single-project MSIX will not compile without the "C++ (v14x) Universal Windows Platform Tools" optional component installed. See Install tools for the Windows App SDK for additional information.
    - *Potential run-time error in an app with Single-project MSIX that consumes types defined in a referenced Windows Runtime Component:* To resolve, manually add activatable class entries to the appxmanifest.xml.
        - The expected error in C# applications is "COMException: Class not registered (0x80040154 (REGDB_E_CLASSNOTREG)).
        - The expected error in C++/WinRT applications is "winrt::hresult_class_not_registered".

- Known issues for **WinUI 3 apps that aren't packaged** (unpackaged apps):
    - Some APIs require package identity, and aren't supported in unpackaged apps, such as:
        - ApplicationData
        - StorageFile.GetFileFromApplicationUriAsync
        - StorageFile.CreateStreamedFileFromUriAsync
        - ApiInformation (not supported on Windows 10)
        - Package.Current
        - Any API in the Windows.ApplicationModel.Resources namespace
        - Any API in the Microsoft.Windows.ApplicationModel.Resources namespace

- Known issues for **packaging and deploying WinUI applications**:
    - The `Package` command is not supported in WinUI apps with Single-project MSIX (Blank App, Packaged template). Instead, use the `Package & Publish` command to create an MSIX package.
    - To create a NuGet package from a C# Class Library with the `Pack` command, ensure the active `Configuration` is `Release`.
    - The `Pack` command is not supported in C++ Windows Runtime Components to create a NuGet package.

For more info, or to get started developing with WinUI, see:

- WinUI
- Create your first WinUI 3 (Windows App SDK) project

Windowing

The Windows App SDK provides an AppWindow class that evolves the previous easy-to-use Windows.UI.WindowManagement.AppWindow preview class and makes it available to all Windows apps, including Win32, WPF, and WinForms.

New Features:

- AppWindow is a high-level windowing API that allows for easy-to-use windowing scenarios that integrates well with the Windows user experience and with other apps. Represents a high-level abstraction of a system-managed container of the content of an app. This is the container in which your content is hosted, and represents the entity that users interact with when they resize and move your app on screen. For developers familiar with Win32, the AppWindow can be seen as a high-level abstraction of the HWND.
- DisplayArea represents a high-level abstraction of a HMONITOR, follows the same principles as AppWindow.
- DisplayAreaWatcher allows a developer to observe changes in the display topology and enumerate DisplayAreas currently defined in the system.

For more info, see Manage app windows (Windows App SDK).

Input

These are the input APIs that support WinUI and provide a lower level API surface for developers to achieve more advanced input interactions.

New Features:

- Pointer APIs: PointerPoint, PointerPointProperties, and PointerEventArgs to support retrieving pointer event information with XAML input APIs.
- InputPointerSource API: Represents an object that is registered to report pointer input, and provides pointer cursor and input event handling for XAML's SwapChainPanel API.
- Cursor API: Allows developers to change the cursor bitmap.
- GestureRecognizer API: Allows developers to recognize certain gestures such as drag, hold, and click when given pointer information.

Important limitations:

- All PointerPoint static factory functions have been removed: **GetCurrentPoint**, **GetCurrentPointTransformed**, **GetIntermediatePoints**, and **GetIntermediatePointsTransformed**.
- The Windows App SDK doesn't support retrieving **PointerPoint** objects with pointer IDs. Instead, you can use the **PointerPoint** member function **GetTransformedPoint** to retrieve a transformed version of an existing **PointerPoint** object. For intermediate points, you can use the **PointerEventArgs** member functions **GetIntermediatePoints** and **GetTransformedIntermediatePoints**.
- Direct use of the platform SDK API **Windows.UI.Core.CoreDragOperation** will not work with WinUI applications.
- **PointerPoint** properties **RawPosition** and **ContactRectRaw** were removed because they referred to non-predicted values, which were the same as the normal values in the OS. Use **Position** and **ContactRect** instead. Pointer prediction is now handled with the **Microsoft.UI.Input.PointerPredictor** API object.

App Lifecycle

Most of the App Lifecycle features already exist in the UWP platform, and have been brought into the Windows App SDK for use by desktop app types, especially unpackaged Console apps, Win32 apps, Windows Forms apps, and WPF apps. The Windows App SDK implementation of these features cannot be used in UWP apps, since there are equivalent features in the UWP platform itself.

> **ⓘ Important**
>
> If you're working on a UWP app, then refer to Migrate from UWP to the Windows App SDK.

Non-UWP apps can also be packaged into MSIX packages. While these apps can use some of the Windows App SDK App Lifecycle features, they must use the manifest approach where this is available. For example, they cannot use the Windows App SDK **RegisterForXXXActivation** APIs and must instead register for rich activation via the manifest.

All the constraints for packaged apps also apply to WinUI apps, which are packaged, and there are additional considerations as described below.

Important considerations:

- Rich activation: GetActivatedEventArgs
  - *Unpackaged apps*: Fully usable.

    - *Packaged apps*: Usable, but these apps can also use the platform `GetActivatedEventArgs`. Note that the platform defines Windows.ApplicationModel.AppInstance whereas the Windows App SDK defines Microsoft.Windows.AppLifecycle.AppInstance. And while UWP apps can use the `ActivatedEventArgs` classes, such as `FileActivatedEventArgs` and `LaunchActivatedEventArgs`, apps that use the Windows App SDK AppLifecycle feature must use the interfaces not the classes (e.g, `IFileActivatedEventArgs`, `ILaunchActivatedEventArgs`, and so on).
    - *WinUi apps*: WinUI's App.OnLaunched is given a Microsoft.UI.Xaml.LaunchActivatedEventArgs, whereas the platform `GetActivatedEventArgs` returns a Windows.ApplicationModel.IActivatedEventArgs, and the WindowsAppSDK `GetActivatedEventArgs` returns a Microsoft.Windows.AppLifecycle.AppActivationArguments object which can represent a platform `LaunchActivatedEventArgs`.
    - For more info, see Rich activation with the app lifecycle API.

- Register/Unregister for rich activation
    - *Unpackaged apps*: Fully usable.
    - *Packaged apps*: Not usable use the app's MSIX manifest instead.
    - For more info, see Rich activation with the app lifecycle API.

- Single/Multi-instancing
    - *Unpackaged apps*: Fully usable.
    - *Packaged apps*: Fully usable.
    - *WinUI apps*: If an app wants to detect other instances and redirect an activation, it must do so as early as possible, and before initializing any windows, etc. To enable this, the app must define DISABLE_XAML_GENERATED_MAIN, and write a custom Main (C#) or WinMain (C++) where it can do the detection and redirection.
    - RedirectActivationToAsync is an async call, and you should not wait on an async call if your app is running in an STA. For Windows Forms and C# WinUI apps, you can declare Main to be async, if necessary. For C++ WinUI and C# WPF apps, you cannot declare Main to be async, so instead you need to move the redirect call to another thread to ensure you don't block the STA.
    - For more info, see App instancing with the app lifecycle API.

- Power/State notifications
    - *Unpackaged apps*: Fully usable.
    - *Packaged apps*: Fully usable.
    - For more info, see Power management with the app lifecycle API.

Known issue:

- File Type associations incorrectly encode %1 to be %251 when setting the Verb handler's command line template, which crashes unpackaged Win32 apps. You can manually edit the Registry value to be %1 instead as a partial workaround. If the target file path has a space in it, then it will still fail and there is no workaround for that scenario.
- These Single/Multi-instancing bugs will be fixed in an upcoming servicing patch:
  - AppInstance redirection doesn't work when compiled for x86
  - Registering a key, unregistering it, and re-registering it causes the app to crash

DWriteCore

DWriteCore is the Windows App SDK implementation of DirectWrite, which is the DirectX API for high-quality text rendering, resolution-independent outline fonts, and full Unicode text and layout support. DWriteCore is a form of DirectWrite that runs on versions of Windows down to Windows 10, version 1809 (10.0; Build 17763), and opens the door for you to use it cross-platform.

Features:

DWriteCore contains all of the features of DirectWrite, with a few exceptions.

Important limitations:

- DWriteCore does not contain the following DirectWrite features:
  - Per-session fonts
  - End-user defined character (EUDC) fonts
  - Font-streaming APIs
- Low-level rendering API support is partial.
- DWriteCore doesn't interoperate with Direct2D, but you can use IDWriteGlyphRunAnalysis and IDWriteBitmapRenderTarget.

For more information, see DWriteCore overview.

MRT Core

MRT Core is a streamlined version of the modern Windows Resource Management System that is distributed as part of the Windows App SDK.

Important limitations:

- In .NET projects, resource files copy-pasted into the project folder aren't indexed on F5 if the app was already built. As a workaround, rebuild the app. See issue 1503 for more info.

- In .NET projects, when a resource file is added to the project using the Visual Studio UI, the files may not be indexed by default. See issue 1786 for more info. To work around this issue, please remove the entries below in the CSPROJ file:

  XML

  ```xml
  <ItemGroup>
      <Content Remove="<image file name>" />
  </ItemGroup>
  <ItemGroup>
      <PRIResource Remove="<resw file name>" />
  </ItemGroup>
  ```

- For unpackaged C++ WinUI apps, the resource URI is not built correctly. To work around this issue, add the following in the vcxproj:

  XML

  ```xml
  <!-- Add the following after <Import
  Project="$(VCTargetsPath)\Microsoft.Cpp.props" /> -->

  <PropertyGroup>
      <AppxPriInitialPath></AppxPriInitialPath>
  </PropertyGroup>
  ```

For more information, see Manage resources with MRT Core.

Deployment

New Features and updates:

- You can auto-initialize the Windows App SDK through the `WindowsPackageType project` property to load the Windows App SDK runtime and call the Windows App SDK APIs. See Create your first WinUI 3 (Windows App SDK) project for instructions.
- Unpackaged apps can deploy Windows App SDK by integrating in the standalone Windows App SDK `.exe` installer into your existing MSI or setup program. For more info, see Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged.

- Unpackaged .NET apps can also use .NET wrapper for the bootstrapper API to dynamically take a dependency on the Windows App SDK framework package at run time. For more info about the .NET wrapper, see .NET wrapper library.
- Packaged apps can use the deployment API to verify and ensure that all required packages are installed on the machine. For more info about how the deployment API works, see Windows App SDK deployment guide for framework-dependent packaged apps.

Important limitations:

- The .NET wrapper for the bootstrapper API is only intended for use by unpackaged .NET applications to simplify access to the Windows App SDK.
- Only MSIX packaged apps that are full trust or have the packageManagement restricted capability have the permission to use the deployment API to install the main and singleton package dependencies. Support for partial-trust packaged apps will be coming in later releases.
- When F5 testing an x86 app which uses the DeploymentManager.Initialize method on an x64 system, ensure that the x64 framework is first installed by running the WindowsAppRuntimeInstall.exe. Otherwise, you will encounter a **NOT_FOUND** error due to Visual Studio not deploying the x64 framework, which normally occurs through Store deployment or sideloading.

Other limitations and known issues

- **No support for Any CPU build configuration**: When adding the Windows App SDK to an existing .NET application or component that supports **Any CPU**, you must specify the desired architecture: `x86`, `x64` or `arm64`.

- **Upgrading from .NET 5 to .NET 6**: When upgrading in the Visual Studio UI, you might run into build errors. As a workaround, manually update your project file's `TargetFrameworkPackage` to the below:

  ```
  XML
  <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
  ```

- **C# Single-project MSIX app doesn't compile if C++ UWP Tools aren't installed.** If you have a C# Single-project MSIX project, then you'll need to install the **C++ (v14x) Universal Windows Platform Tools** optional component.

- **Subsequent language VSIX fails to install into Visual Studio 2019 when multiple versions of Visual Studio 2019 are installed.** If you have multiple versions of Visual

Studio 2019 installed (e.g. Release and Preview) and then install the Windows App SDK VSIX for both C++ *and* C#, the second installation will fail. To resolve, uninstall the Single-project MSIX Packaging Tools for Visual Studio 2019 after the first language VSIX. View this feedback for additional information about this issue.

- An alternative to **DispatcherQueue.TryEnqueue** (for resuming execution on the dispatcher queue thread) is to use the resume_foreground helper function in the Windows Implementation Library (WIL):

    1. Add a reference to your project to the Microsoft.Windows.ImplementationLibrary NuGet package.
    2. Add `#include <wil/cppwinrt_helpers.h>` to your `pch.h`.
    3. Add `#include <winrt/Microsoft.UI.Dispatching.h>` to your `pch.h`.
    4. Now `co_await wil::resume_foreground(your_dispatcherqueue);`.

Related topics

- Latest preview channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Stable channel release notes for the Windows App SDK 0.8

Article • 05/23/2024

The stable channel provides releases of the Windows App SDK that are supported for use by apps in production environments. Apps that use the stable release of the Windows App SDK can also be published to the Microsoft Store.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Downloads for the Windows App SDK

> **Note**
>
> The Windows App SDK Visual Studio Extensions (VSIX) are no longer distributed as a separate download. They are available in the Visual Studio Marketplace inside Visual Studio.

Version 0.8

The latest available release of the 0.8.x lineage of the stable channel of the Windows App SDK is version 0.8.12.

> **Note**
>
> The Windows App SDK was previously known by the code name **Project Reunion**. Some SDK assets in version 0.8 and earlier still use the code name. Some areas of the documentation still use **Project Reunion** when referring to an existing asset or a specified earlier release.

Version 0.8.12

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 0.8.0 release.

> **Note**
>
> For C# developers, one of the following .NET SDK versions is required: 5.0.213, 5.0.407, 6.0.104, 6.0.202 (or later). To update your .NET SDK version, visit .NET Downloads or update to the latest version of Visual Studio. Without the required .NET SDK version, when updating your NuGet package you will see an error like: "*This version of WindowsAppSDK requires WinRT.Runtime.dll version 1.6 or greater.*".

Bug fixes (0.8.12)

- Fixed issue where apps with SwapChainPanel or WebView2 would unpredictably crash due to an access violation.

Version 0.8.11

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 0.8.0 release.

> **Note**
>
> For C# developers, one of the following .NET SDK versions is required: 5.0.213, 5.0.407, 6.0.104, 6.0.202 (or later). To update your .NET SDK version, visit .NET Downloads or update to the latest version of Visual Studio. Without the required .NET SDK version, when updating your NuGet package you will see an error like: "*This version of WindowsAppSDK requires WinRT.Runtime.dll version 1.6 or greater.*".

Bug fixes (0.8.11)

- Fixed regression causing the lost focus event to fire when selecting text using mouse.

Version 0.8.10

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 0.8.0 release.

> **Note**
>
> For C# developers, one of the following .NET SDK versions is required: 5.0.213, 5.0.407, 6.0.104, 6.0.202 (or later). To update your .NET SDK version, visit .NET Downloads or update to the latest version of Visual Studio. Without the required .NET SDK version, when updating your NuGet package you will see an error like: "*This version of WindowsAppSDK requires WinRT.Runtime.dll version 1.6 or greater.*".

Bug fixes (0.8.10)

- Fixed issues causing apps to sometimes crash during a drag and drop operation.

> **Note**
>
> Windows App SDK 0.8.9 was not released. The version released directly after 0.8.8 is 0.8.10.

Version 0.8.8

This is a servicing release of the Windows App SDK that includes critical bug fixes for the 0.8.0 release.

> **Note**
>
> For C# developers, one of the following .NET SDK versions (or later) is required: 6.0.202, 6.0.104, 5.0.407, 5.0.213. To update your .NET SDK version, visit .NET Downloads or update to the latest version of Visual Studio. Without the required .NET SDK version, when updating your NuGet package you will see an error like: "*This version of WindowsAppSDK requires WinRT.Runtime.dll version 1.6 or greater.*".

Bug fixes (0.8.8)

- Fixed touch input issues in TextBox regarding soft keyboard and general interaction. These issues also affected keyboard shortcuts. For more information, see issue 6291 on GitHub.
- Fixed issue where an app window would sometimes show as inactive when active.

- Fixed performance issue caused by UIA (UI Automation) running in external processes.
- Fixed app stability issue with pen input.
- Fixed issue where the render of png icons in a Menu are dramatically delayed because of UIA.

Version 0.8.7

This is a servicing release of the Windows App SDK that includes several performance updates for C#/.NET applications. To update to this version, you'll need to reference the latest Windows SDK package version. To do that, add the property `<WindowsSdkPackageVersion>10.0.<sdk_version>.24</WindowsSdkPackageVersion>` to your `.csproj` file with the SDK version your app is targeting from the `TargetFramework` property. For example:

XML

```
<Project Sdk="Microsoft.NET.Sdk">
   <PropertyGroup>
       <OutputType>WinExe</OutputType>
       <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
       <TargetPlatformMinVersion>10.0.17763.0</TargetPlatformMinVersion>
       <WindowsSdkPackageVersion>10.0.19041.24</WindowsSdkPackageVersion>
   <PropertyGroup>
   ...
```

This version of the Windows SDK projection will be available in an upcoming .NET 6 servicing release. After that .NET SDK update is available, you should remove the `<WindowsSdkPackageVersion>` property from your project file.

If you don't set this property, then you'll see an error like: `"Error: This version of Project Reunion requires WinRT.Runtime.dll version 1.6 or greater."`

Version 0.8.6

This is a servicing release of the Windows App SDK that includes several performance improvements for C#/.NET applications for the 0.8.0 release.

To update to this version of Windows App SDK, you will need to have the latest .NET SDK December update installed (see Download .NET and .NET 5 will reach End of Support on May 10, 2022). If you don't have the minimum required version of the .NET SDK installed, then you'll see an error like `"Error: This version of Project Reunion requires WinRT.Runtime.dll version 1.4 or greater."`

Bug Fixes (0.8.6)

For a detailed list of the performance improvements, see the C#/WinRT 1.4.1 release notes.

Version 0.8.5

This is a servicing release of the Windows App SDK that includes more critical bug fixes for the 0.8.0 release.

Bug fixes (0.8.5)

- Fixed issue that was causing WinUI apps using pointer input to crash.
- Fixed issue causing the title bar buttons (min, max, close) to not have rounded corners on Windows 11.
- Fixed issue causing the resizing layout options to not appear when hovering over maximize/restore button on Windows 11.
- Fixed issue causing a crashing exception where creating a PointCollection object. For more information, see issue 971 on Github.

The limitations and known issues for version 0.8 also apply to version 0.8.5, unless marked otherwise in the section below.

Version 0.8.4

This is a servicing release of the Windows App SDK that includes more critical bug fixes for the 0.8.0 release.

Bug fixes (0.8.4)

- Fixes for custom title bars so that ContentDialog doesn't cover it up, and the title bar buttons are rounded.
- Fix for a crash in image processing when the display scale is changed.
- Fixes clipping bugs where UI disappear or clipped incorrectly

The limitations and known issues for version 0.8 also apply to version 0.8.4, unless marked otherwise in the section below.

Version 0.8.3

This is a servicing release of the Windows App SDK that includes more critical bug fixes for the 0.8.0 release.

Bug fixes (0.8.3)

Keyboard focus was being lost when a window was minimized and then restored, requiring a mouse click to restore focus.

The limitations and known issues for version 0.8 also apply to version 0.8.3, unless marked otherwise in the section below.

Version 0.8.2

This is a servicing release of the Windows App SDK that includes more critical bug fixes for the 0.8.0 release.

Bug fixes (0.8.2)

- Windows App SDK and WinUI 3 are now supported in Visual Studio 2022 Preview 2 and later.
- For .NET apps, you may receive the following error when passing in an array of enums: `Object contains non-primitive or non-blittable data.`
- Writing using the HandWriting Panel inside a textbox causes a crash
- Icons/images always load at their 100% scale value rather than based on the monitor scale value
- Garbage collection of **EventSource<T>** causes subsequent failure to unsubscribe handlers (see GitHub issue for more details)
- Security fix – see CVE-2021-34533 for more details.
- SwapChainPanel.CompositionScaleChanged sometimes returning incorrect CompositionScale values after changing display scale

The limitations and known issues for version 0.8 also apply to version 0.8.2, unless marked otherwise in the section below.

Version 0.8.1

This is a servicing release of the Windows App SDK that includes a few critical bug fixes for the 0.8.0 release.

Bug fixes (0.8.1)

- Windows App SDK cannot run on the latest Windows Insider build
- Crash in EditableComboBox when entering a value that does not appear in dropdown
- WebView2 doesn't allow user to tab out once focused has been received
- Fully qualify Windows.Foundation.Metadata.DefaultOverload namespace in WinUI generated code to avoid namespace ambiguity
  - This fixes bug #5108.
- Security fix – see CVE-2021-34489 for more details.

The limitations and known issues for version 0.8 also apply to version 0.8.1, unless marked otherwise in the section below.

Version 0.8.0 Stable

New features and updates

This release supports all stable channel features.

WinUI 3:

This release includes many bug fixes and improved stabilization across WinUI 3. These are all of the new changes in WinUI 3 since the release of WinUI 3 - Project Reunion 0.5:

- The Pivot control has been added back in and can now be used in any WinUI 3 app.

- All bug fixes from Project Reunion v0.5.5, v0.5.6, and v0.5.7 are included with this release.

- New bug fixes, including:
  - Mouse right-click in TextBox crashes the application
  - NavigationView causes crash in UWP, Reunion 0.5 Preview
  - ProgressBar doesn't show difference between Paused and Error option
  - Crash in RichEditBox when copying/pasting/changing text style
  - Window caption buttons are misplaced when SetTitleBar is not set or null

  For the full list of bugs addressed in this release, see our GitHub repo.

- The `ColorHelper.ToDisplayName` API is no longer available.

- The following types have been removed:
  - `Microsoft.Graphics.IGeometrySource2D`
  - `Microsoft.Graphics.IGeometrySource2DInterop`

Use `Windows.Graphics.IGeometrySource2D` and `Windows.Graphics.IGeometrySource2DInterop` instead.

- All types in the `Microsoft.System` namespace have been moved to the `Microsoft.UI.Dispatching` namespace, including the DispatcherQueue class.
- The `AcrylicBrush.BackgroundSource` property has been removed, since `HostBackdrop` is not supported as a `BackgroundSource` in WinUI 3.

For more information on WinUI, see WinUI.

To see WinUI 3 controls and features in action, you can clone and build the WinUI 3 Gallery app from GitHub, or download the app from the Microsoft Store.

To get started developing with WinUI, check out the following articles:

- WinUI 3 templates in Visual Studio
- Create your first WinUI 3 (Windows App SDK) project
- Build a C# .NET app with WinUI 3 and Win32 interop
- WinUI 3 API Reference

DWriteCore:

This release of DWriteCore includes the following new and updated features. DWriteCore is introduced and described in the DWriteCore overview.

- DWriteCore now has support for underline—see **IDWriteTextLayout::GetUnderline** and **IDWriteTextLayout::SetUnderline**.
- Support for strikethrough—see **IDWriteTextLayout::GetStrikethrough** and **IDWriteTextLayout::SetStrikethrough**.
- Support for vertical text via **IDWriteTextLayout**—see Vertical text.
- All of the methods of the **IDWriteTextAnalyzer** and **IDWriteTextAnalyzer1** interfaces are implemented.
- The **DWriteCoreCreateFactory** free function creates a factory object that is used for subsequent creation of individual DWriteCore objects.

> **Note**
>
> DWriteCoreCreateFactory is functionally the same as the DWriteCreateFactory function exported by the system version of DirectWrite. The DWriteCore function has a different name to avoid ambiguity in the event that you link both `DWriteCore.lib` and `DWrite.lib`.

For DWriteCore and DirectWrite API reference, see DWriteCore API Reference and DirectWrite API Reference.

MRTCore:

- The **Build Action** for resources is automatically set when you add the resource to your project, reducing the need for manual project configuration.

Limitations

- This release is not currently supported on the Dev Channel of the Windows Insider Program. **This is fixed in version 0.8.1**.
- Desktop apps (C# or C++ desktop): This release is supported for use only in desktop apps (C++ or C#) that are packaged using MSIX. To use the Windows App SDK in unpackaged desktop apps, you must use the experimental release channel.

> **Important**
>
> If you're working on a UWP app, then refer to **Migrate from UWP to the Windows App SDK**.

Known issues

- WinUI 3 tooling such as Live Visual Tree, Live Property Explorer, and Hot Reload in version 0.8 and later requires Visual Studio 2019 16.11 Preview 3 and later.
- Apps currently using WinUI 3 and the Windows App SDK 0.8 cannot use class libraries that use Project Reunion 0.5. Update the class libraries to use the Windows App SDK 0.8.
- .NET apps must target build 18362 or later: Your TFM must be set to net6.0-windows10.0.18362 or later, and your packaging project's must be set to 18362 or later. For more info, see GitHub issue #921.
- You may encounter a crash when switching frequently between light and dark mode.
- For .NET apps, you may receive the following error when passing in an array of enums: `Object contains non-primitive or non-blittable data.` **This is fixed in version 0.8.2**.

- For .NET apps, there is currently no way to opt out of an image getting indexed as an app resource using the Visual Studio UI. To work around this, add a Directory.Build.targets (see Customize your build - Visual Studio for instructions) to the project and remove the image(s) as follows:
    - To remove specific images (note that the relative path is needed):

      XML

      ```
      <Project>
      <ItemGroup>
          <Content Remove="..\Bitmap1.bmp" />
      </ItemGroup>
      </Project>
      ```

    - To remove images based on metadata:

      XML

      ```
      <Project>
      <ItemGroup>
          <Content Remove="@(None->WithMetadataValue('Pack','true'))" />
      </ItemGroup>
      </Project>
      ```

    A fix for this issue is planned for an upcoming release - at that point, the above workarounds will no longer be needed.

Related topics

- Latest preview channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Stable channel release notes for the Windows App SDK 0.5

Article • 05/23/2024

The stable channel provides releases of the Windows App SDK that are supported for use by apps in production environments. Apps that use the stable release of the Windows App SDK can also be published to the Microsoft Store.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Downloads for the Windows App SDK

> **Note**
>
> The Windows App SDK Visual Studio Extensions (VSIX) are no longer distributed as a separate download. They are available in the Visual Studio Marketplace inside Visual Studio.

Version 0.5

The latest available release of the 0.5.x lineage of the stable channel of the Windows App SDK is version 0.5.9.

Download

New features and updates

This release supports all stable channel features.

Known issues and limitations

This release has the following limitations and known issues:

- **Desktop apps (C# or C++ desktop)**: This release is supported for use only in desktop apps (C++ or C#) that are packaged using MSIX. To use the Windows App SDK in unpackaged desktop apps, you must use the experimental release channel.
- **.NET apps must target build 18362 or later**: Your TFM must be set to `net6.0-windows10.0.18362` or later, and your packaging project's `<TargetPlatformVersion>` must be set to 18362 or later. For more info, see the known issue on GitHub.

> **Important**
>
> If you're working on a UWP app, then refer to **Migrate from UWP to the Windows App SDK**.

Related topics

- Latest preview channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Preview channel release notes for the Windows App SDK 1.5

Article • 08/13/2024

> **ⓘ Important**
>
> The preview channel is **not supported** for use in production environments, and apps that use the preview releases cannot be published to the Microsoft Store.

The preview channel includes releases of the Windows App SDK with preview channel features in late stages of development. Preview releases do not include experimental features and APIs but may still be subject to breaking changes before the next stable release.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on preview releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest preview channel release:

- Latest preview channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.5 Preview 1 (1.5.0-preview1)

This is the latest release of the preview channel for version 1.5.

In an existing Windows App SDK 1.4 (from the stable channel) app, you can update your Nuget package to 1.5.0-preview1 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Downloads for the Windows App SDK.

XAML Islands runtime and shutdown updates

- There is a behavioral difference between WinAppSDK 1.4 and WinAppSDK 1.5 for Xaml Islands-based apps when the last Xaml Window on any thread is closed.
  - In WinAppSDK 1.4, the Xaml runtime always exits the thread's event loop when the last Xaml window on a thread is closed.
  - In WinAppSDK 1.5:
    - If your app is a WinUI Desktop app, the default behavior is still the same as in WinAppSDK 1.4.
    - If you're using Xaml for the DesktopWindowXamlSource ("Xaml Islands") API, the default behavior is now that Xaml does not automatically exit the thread's event loop.
    - In both modes, you can change this behavior by setting the `Application.DispatcherShutdownMode` property.
  - For more information, see the documentation for the `Application.DispatcherShutdownMode` property when available.
- There is a behavioral difference between WinAppSDK 1.4 and WinAppSDK 1.5 for Xaml Islands-based apps in the lifetime of the Xaml runtime:
  - In WinAppSDK 1.4, the Xaml runtime shuts down on a thread if either all `WindowsXamlManager` and `DesktopWindowXamlSource` objects on a given thread are closed or shut down, or the `DispatcherQueue` running on that thread is shut down (the Xaml runtime shuts down during the `DispatcherQueue.FrameworkShutdownStarting` stage).
  - In WinAppSDK 1.5, the Xaml runtime shuts down on a thread only when the DispatcherQueue running on that thread is shut down (the Xaml runtime shuts down during the `DispatcherQueue.FrameworkShutdownStarting` stage).
  - For more information, see the documentation for the `WindowsXamlManager` class when available.

WinUI Maps control

The initial release of the WinUI `Maps` control is now available! This control is powered by WebView2 and Azure Maps, providing the following features:

- Panning and zooming with either the map buttons or touch.
- Changing the style of the map to satellite, terrain, or street view.
- Programmatically adding interactable pins with developer-customizable icons to the map.
- Developer customization for where the map is centered on initial load.
- Control for developers over hiding or showing the buttons for panning, zooming, and map styles.

> **Note**
>
> To use the `Maps` control, you'll need an Azure Maps key. To create the key, see the Azure Maps documentation page for creating a web app.

The `Maps` control is entirely new and we welcome your feedback to evaluate its future direction!

Other new features from across the WinAppSDK

- Added support for the PublishSingleFile deployment model. For more info about PublishSingleFile, see the Single-file deployment documentation.

Bug fixes

- Fixed an issue from the 1.5-experimental2 release where the projection DLL was not generated. For more info, see GitHub issue #4152.
- Fixed an issue where the ellipsis button on the text formatting popup of the `RichEditBox` was not displaying the list of actions properly. For more info, see GitHub issue #9140.
- Fixed an issue where `ListView` didn't handle keyboard accelerators properly. For more info, see GitHub issue #8063.
- Fixed an access violation issue with using `AccessKey` to close a window. For more info, see GitHub issue #8648.
- Fixed an issue affecting text alignment in a `MenuFlyoutItem` within a `MenuBar`. For more info, see GitHub issue #8755.
- Fixed an issue where highlighted text would not remain highlighted upon right-click. For more info, see GitHub issue #1801.
- Fixed an issue causing inactive windows to crash the app when closed. For more info, see GitHub issue #8913.
- Fixed an issue that could hang applications when scrolling with the middle mouse button and left-clicking immediately afterwards. For more info, see GitHub issue #9233.

New APIs in 1.5.0-preview1

Version 1.5-preview1 includes the following new APIs compared to the stable 1.4 release:

C#

```
Microsoft.Graphics.DirectX

    DirectXPixelFormat
        A4B4G4R4
```

C#

```
Microsoft.UI.Input

    FocusNavigationReason
    FocusNavigationRequest
    FocusNavigationRequestEventArgs
    FocusNavigationResult
    InputFocusController
        DepartFocus
        NavigateFocusRequested

    InputFocusNavigationHost
```

C#

```
Microsoft.UI.Xaml

    Application
        DispatcherShutdownMode

    DebugSettings
        LayoutCycleDebugBreakLevel
        LayoutCycleTracingLevel

    DispatcherShutdownMode
    LayoutCycleDebugBreakLevel
    LayoutCycleTracingLevel
```

C#

```
Microsoft.UI.Xaml.Controls

    MapControl
    MapControlMapServiceErrorOccurredEventArgs
    MapElement
    MapElementClickEventArgs
    MapElementsLayer
    MapIcon
    MapLayer
    SelectorBar
    SelectorBarItem
    SelectorBarSelectionChangedEventArgs
```

```
    WebView2
        EnsureCoreWebView2Async
        EnsureCoreWebView2Async
```

C#

```
Microsoft.UI.Xaml.Hosting

    WindowsXamlManager
        GetForCurrentThread
        XamlShutdownCompletedOnThread

    XamlShutdownCompletedOnThreadEventArgs
```

C#

```
Microsoft.Web.WebView2.Core

    CoreWebView2
        FrameId

    CoreWebView2AcceleratorKeyPressedEventArgs
        IsBrowserAcceleratorKeyEnabled

    CoreWebView2BrowserExtension
    CoreWebView2BrowsingDataKinds
        ServiceWorkers

    CoreWebView2CustomSchemeRegistration
        CoreWebView2CustomSchemeRegistration (String)
        AllowedOrigins
        SchemeName

    CoreWebView2Environment
        GetProcessExtendedInfosAsync

    CoreWebView2EnvironmentOptions
        AreBrowserExtensionsEnabled
        CustomSchemeRegistrations

    CoreWebView2Frame
        FrameId

    CoreWebView2FrameInfo
        FrameId
        FrameKind
        ParentFrameInfo

    CoreWebView2FrameKind
    CoreWebView2MouseEventKind
        NonClientRightButtonDown
        NonClientRightButtonUp
```

```
    CoreWebView2NavigationKind
    CoreWebView2NavigationStartingEventArgs
        NavigationKind

    CoreWebView2NewWindowRequestedEventArgs
        OriginalSourceFrameInfo

    CoreWebView2ProcessExtendedInfo
    CoreWebView2Profile
        AddBrowserExtensionAsync
        Delete
        Deleted
```

C#

```
Microsoft.Windows.Management.Deployment

    AddPackageOptions
    EnsureReadyOptions
    PackageDeploymentContract
    PackageDeploymentManager
    PackageDeploymentProgress
    PackageDeploymentProgressStatus
    PackageDeploymentResult
    PackageDeploymentStatus
    PackageRuntimeManager
    PackageSet
    PackageSetItem
    PackageSetItemRuntimeDisposition
    PackageSetRuntimeDisposition
    PackageVolume
    ProvisionPackageOptions
    RegisterPackageOptions
    RemovePackageOptions
    StagePackageOptions
    StubPackageOption
```

C#

```
Microsoft.Windows.Widgets.Feeds.Providers

    CustomQueryParametersRequestedArgs
    CustomQueryParametersUpdateOptions
    FeedDisabledArgs
    FeedEnabledArgs
    FeedManager
    FeedProviderDisabledArgs
    FeedProviderEnabledArgs
    FeedProviderInfo
```

```
    IFeedManager
    IFeedProvider
```

Related topics

- Latest stable channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Preview channel release notes for the Windows App SDK 1.4

Article • 05/23/2024

> **ⓘ Important**
>
> The preview channel is **not supported** for use in production environments, and apps that use the preview releases cannot be published to the Microsoft Store.

The preview channel includes releases of the Windows App SDK with preview channel features in late stages of development. Preview releases do not include experimental features and APIs but may still be subject to breaking changes before the next stable release.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on preview releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest preview channel release:

- Latest preview channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.4 Preview 2 (1.4.0-preview2)

This is the latest release of the preview channel for version 1.4.

In an existing Windows App SDK 1.3 (from the stable channel) app, you can update your Nuget package to 1.4.0-preview2 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Latest Windows App SDK downloads.

XAML Islands no longer experimental

XAML Islands and the underlying ContentIslands platform are no longer experimental.

- Currently XAML Islands are only tested for use in C++ apps. This release does not include any convenient wrapper elements for use in WPF or WinForms.
- `DesktopWindowXamlSource` and related types have been added in the Microsoft.UI.Xaml.Hosting namespace for XAML Islands. `XamlRoot.ContentIslandEnvironment` was added to help access the underlying Island information for an element.
- Many new types have been introduced in the Microsoft.UI.Content namespace and the Microsoft.UI.Input namespace as the underlying support for XAML Islands or for using this ContentIslands functionality without XAML.
- A new `DragDropManager` (plus related types) has been added in the Microsoft.UI.Input.DragDrop namespace for Island scenarios.

ItemsView updates

The new `ItemsView` class that was introduced in version 1.4-preview1 has been substantially updated with new properties and a new supporting class.

- The new `ItemsView` control displays a data collection. `ItemsView` is similar to the `ListView` and `GridView` controls, but is built using the `ItemsRepeater`, `ScrollView`, `ItemContainer` and `ItemCollectionTransitionProvider` components. It offers the unique ability to plug in custom `Layout` or `ItemCollectionTransitionProvider` implementations. Another key advantage is the ability to switch the layout on the fly while preserving items selection. The inner `ScrollView` control also offers features unavailable in `ListView`/`GridView`'s `ScrollViewer` control such as the ability to control the animation during programmatic scrolls.
  - A new `ItemTransitionProvider` property on `ItemsRepeater` (and the new `ItemsView` control) lets you specify an `ItemCollectionTransitionProvider` object to control transition animations on that control. A `CreateDefaultItemTransitionProvider` method has also been added to `Layout`, which enables a layout object to provide a fallback transition to accompany it if you do not provide one explicitly on the `ItemsView` control.
  - A new `IndexBasedLayoutOrientation` property on `Layout` where the layout orientation, if any, of items is based on their index in the source collection. The default value is `IndexBasedLayoutOrientation.None`. Custom layouts set this property by calling the new (protected) `SetIndexBasedLayoutOrientation` method.
  - A new `VisibleRect` property on `VirtualizingLayoutContext` gets the visible viewport rectangle within the `FrameworkElement` associated with the `Layout`. The

protected virtual `VirtualizingLayoutContext.VisibleRectCore` method can be overridden to provide the value that will be returned from the `VisibleRect` property.

- The new `LinedFlowLayout` class is typically used to lay out the items of the `ItemsView` collection control. It is particularly useful for displaying collection of pictures. It does so by laying them out from left to right, and top to bottom, in lines of equal height. The pictures fill a horizontal line and then wrap to a next line. Pictures may be cropped at the left and right edges to fit into a line. They may also be expanded horizontally and cropped at the top and bottom edges to fill a line when the stretching mode is employed.

New features and updates from across the WinAppSDK

- `Popup/FlyoutBase.IsConstrainedToRootBounds = false` is now supported, allowing a popup/flyout to extend outside the bounds of the parent window. A `SystemBackdrop` property has been added to these types to support having acrylic in these unconstrained popups. Menus by default use this to have acrylic.
- `Closed`, `FrameworkClosed`, and `IsClosed` have been added to `DesktopAcrylicController` and `MicaController` to improve handling during object/thread shutdown.
- `DesktopAcrylicController.Kind` can now be set to choose between some standard acrylic appearances.
- `DispatcherQueue` has some new events and helpers to facilitate better organized shutdown and for apps using Islands to easily run a standard supported event loop.
- `InputNonClientPointerSource` in the Microsoft.UI.Input namespace can be used for custom titlebar scenarios to define non-client area regions. Code can register for corresponding events like hover and click events on these regions.
- `AppWindow` has some new helpers to get and associate with a `DispatcherQueue`.
- The new `TreeView.SelectionChanged` event allows developers to respond when the user or code-behind changes the set of selected nodes in the `TreeView` control.
- The new `ScrollView` control provides a new alternative to `ScrollViewer`. This new control is highly aligned in behavior and API with the existing `ScrollViewer` control, but is based on `InteractionTracker`, has new features such as animation-driven view changes, and is also designed to ensure full functionality of `ItemsRepeater`. See A more flexible ScrollViewer · Issue #108 · microsoft/microsoft-ui-xaml (github.com) for more details. Various new types, including `ScrollPresenter`, are part of the overall `ScrollView` model.

- The new `AnnotatedScrollBar` control extends a regular scrollbar's functionality by providing an easy way to navigate through a large collection of items. This is achieved through a clickable rail with labels that act as markers. It also allows for a more granular understanding of the scrollable content by displaying a tooltip when hovering over the clickable rail.

New APIs in 1.4.0-preview2

Version 1.4-preview2 includes the following new APIs compared to the previous 1.4-preview1 release:

C#

```
Microsoft.UI

    ClosableNotifierHandler
    IClosableNotifier
```

C#

```
Microsoft.UI.Composition.SystemBackdrops

    DesktopAcrylicController
        Closed
        FrameworkClosed
        IsClosed
        Kind

    DesktopAcrylicKind
    MicaController
        Closed
        FrameworkClosed
        IsClosed
```

C#

```
Microsoft.UI.Content

    ContentCoordinateConverter
    ContentCoordinateRoundingMode
    ContentDeferral
    ContentEnvironmentSettingChangedEventArgs
    ContentEnvironmentStateChangedEventArgs
    ContentIsland
    ContentIslandAutomationProviderRequestedEventArgs
    ContentIslandEnvironment
    ContentIslandStateChangedEventArgs
    ContentLayoutDirection
```

```
    ContentSite
    ContentSiteEnvironment
    ContentSiteEnvironmentView
    ContentSiteRequestedStateChangedEventArgs
    ContentSiteView
    ContentSizePolicy
    DesktopChildSiteBridge
    DesktopSiteBridge
    IContentSiteBridge
```

C#

```
Microsoft.UI.Dispatching

    DispatcherExitDeferral
    DispatcherQueue
        EnqueueEventLoopExit
        EnsureSystemDispatcherQueue
        FrameworkShutdownCompleted
        FrameworkShutdownStarting
        RunEventLoop
        RunEventLoop

    DispatcherQueueController
        ShutdownQueue

    DispatcherRunOptions
```

C#

```
Microsoft.UI.Input

    CharacterReceivedEventArgs
    ContextMenuKeyEventArgs
    FocusChangedEventArgs
    InputActivationListener
        GetForIsland

    InputFocusChangedEventArgs
    InputFocusController
    InputKeyboardSource
        CharacterReceived
        ContextMenuKey
        GetCurrentKeyState
        GetForIsland
        GetKeyState
        KeyDown
        KeyUp
        SystemKeyDown
        SystemKeyUp

    InputNonClientPointerSource
```

```
    InputPointerSource
        GetForIsland

    InputPreTranslateKeyboardSource
    KeyEventArgs
    NonClientCaptionTappedEventArgs
    NonClientPointerEventArgs
    NonClientRegionKind
    NonClientRegionsChangedEventArgs
    PhysicalKeyStatus
    VirtualKeyStates
```

C#

```
Microsoft.UI.Input.DragDrop

    DragDropManager
    DragDropModifiers
    DragInfo
    DragOperation
    DragUIContentMode
    DragUIOverride
    DropOperationTargetRequestedEventArgs
    IDropOperationTarget
```

C#

```
Microsoft.UI.Windowing

    AppWindow
        AssociateWithDispatcherQueue
        Create
        DispatcherQueue
```

C#

```
Microsoft.UI.Xaml

    XamlRoot
        ContentIslandEnvironment
```

C#

```
Microsoft.UI.Xaml.Automation.Peers

    ItemsViewAutomationPeer
```

C#

```
Microsoft.UI.Xaml.Controls

    AnnotatedScrollBar
    AnnotatedScrollBarDetailLabelRequestedEventArgs
    AnnotatedScrollBarLabel
    AnnotatedScrollBarScrollingEventArgs
    AnnotatedScrollBarScrollingEventKind
    IndexBasedLayoutOrientation
    ItemCollectionTransition
    ItemCollectionTransitionCompletedEventArgs
    ItemCollectionTransitionOperation
    ItemCollectionTransitionProgress
    ItemCollectionTransitionProvider
    ItemCollectionTransitionTriggers
    ItemsRepeater
        ItemTransitionProvider
        ItemTransitionProviderProperty

    ItemsView
    ItemsViewItemInvokedEventArgs
    ItemsViewSelectionChangedEventArgs
    ItemsViewSelectionMode
    Layout
        CreateDefaultItemTransitionProvider
        IndexBasedLayoutOrientation
        SetIndexBasedLayoutOrientation

    LinedFlowLayout
    LinedFlowLayoutItemCollectionTransitionProvider
    LinedFlowLayoutItemsInfoRequestedEventArgs
    LinedFlowLayoutItemsJustification
    LinedFlowLayoutItemsStretch
    ScrollingAnchorRequestedEventArgs
    ScrollingAnimationMode
    ScrollingBringingIntoViewEventArgs
    ScrollingChainMode
    ScrollingContentOrientation
    ScrollingInputKinds
    ScrollingInteractionState
    ScrollingRailMode
    ScrollingScrollAnimationStartingEventArgs
    ScrollingScrollBarVisibility
    ScrollingScrollCompletedEventArgs
    ScrollingScrollMode
    ScrollingScrollOptions
    ScrollingSnapPointsMode
    ScrollingZoomAnimationStartingEventArgs
    ScrollingZoomCompletedEventArgs
    ScrollingZoomMode
    ScrollingZoomOptions
    ScrollView
    TreeView
        SelectionChanged
```

```
    TreeViewSelectionChangedEventArgs
    VirtualizingLayoutContext
        VisibleRect
        VisibleRectCore
```

C#

```
Microsoft.UI.Xaml.Controls.Primitives

    FlyoutBase
        SystemBackdrop
        SystemBackdropProperty

    IScrollController
    IScrollControllerPanningInfo
    Popup
        SystemBackdrop
        SystemBackdropProperty

    RepeatedScrollSnapPoint
    RepeatedZoomSnapPoint
    ScrollControllerAddScrollVelocityRequestedEventArgs
    ScrollControllerPanRequestedEventArgs
    ScrollControllerScrollByRequestedEventArgs
    ScrollControllerScrollToRequestedEventArgs
    ScrollPresenter
    ScrollSnapPoint
    ScrollSnapPointBase
    ScrollSnapPointsAlignment
    SnapPointBase
    ZoomSnapPoint
    ZoomSnapPointBase
```

C#

```
Microsoft.UI.Xaml.Hosting

    DesktopWindowXamlSource
    DesktopWindowXamlSourceGotFocusEventArgs
    DesktopWindowXamlSourceTakeFocusRequestedEventArgs
    WindowsXamlManager
    XamlSourceFocusNavigationReason
    XamlSourceFocusNavigationRequest
    XamlSourceFocusNavigationResult
```

Version 1.4 Preview 1 (1.4.0-preview1)

This is the latest release of the preview channel for version 1.4.

In an existing Windows App SDK 1.3 (from the stable channel) app, you can update your Nuget package to 1.4.0-preview1 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Latest Windows App SDK downloads.

Widgets updates

Three new interfaces have been added for Widget Providers to implement: `IWidgetProvider2`, `IWidgetProviderAnalytics`, and `IWidgetProviderErrors`. `IWidgetProvider2` allows providers to respond to the *Customize* action invoked by the user, which is identical to what is available for 1st party Widgets. The `IWidgetProviderAnalytics` and `IWidgetProviderErrors` interfaces are used by providers to gather telemetry for their widgets; analytics and failure events about widgets are communicated to the respective widget providers. The `WidgetCustomizationRequestedArgs`, `WidgetAnalyticsInfoReportedArgs`, and `WidgetErrorInfoReportedArgs` classes are used to communicate relevant information to support new functionalities.

New features from across the WinAppSDK

- A new `ThemeSettings` class that allows Win32 WinRT apps to detect when the system's High Contrast setting has changed, similar to UWP's AccessibilitySettings class. See the ThemeSettings API spec on GitHub for more information.
- `Popup/FlyoutBase.ShouldConstrainToRootBounds` is now supported to allow tooltips, menus, and other popups to extend outside the bounds of the main window. Preview 1 does not yet fully support having Acrylic or other SystemBackdrops on a popup/flyout; additional APIs and implementation for this will be included in the next 1.4 release.
- `AccessKeyManager.EnterDisplayMode` is a new method to display access keys for the current focused element of a provided root. Access keys are in "display mode" when showing a key tip to invoke a command, such as pressing the Alt key in Paint to show what keys correspond to what controls. This method allows for programmatically entering display mode.
- `Application.ResourceManagerRequested` provides a mechanism to provide a different `IResourceManager` to resolve resource URIs for scenarios when the default `ResourceManager` won't work. For more information, see the Application.ResourceManagerRequested API spec on GitHub.
- We're introducing a new list control called the `ItemsView` and a corresponding concrete `ItemContainer` class. `ItemContainer` is a lightweight container with built-

in selection states and visuals, which can easily wrap desired content and be used with `ItemsView` for a collection control scenario. `ItemsView` is still marked experimental in Preview 1 but will be included in the next 1.4 release.
- The version of the WebView2 SDK was updated from 1661.34 to 1823.32.

New APIs in 1.4.0-preview1

Version 1.4-preview1 includes the following new APIs compared to the stable 1.3 release:

```
C#
```

```
Microsoft.UI.System

    ThemeSettings
```

```
C#
```

```
Microsoft.UI.Xaml

    Application
        ResourceManagerRequested

    ResourceManagerRequestedEventArgs
```

```
C#
```

```
Microsoft.UI.Xaml.Automation.Peers

    ItemContainerAutomationPeer
```

```
C#
```

```
Microsoft.UI.Xaml.Controls

    ItemContainer
```

```
C#
```

```
Microsoft.UI.Xaml.Controls.Primitives

    CommandBarFlyoutCommandBar
        SystemBackdrop
        SystemBackdropProperty
```

C#

```
Microsoft.UI.Xaml.Input

    AccessKeyManager
        EnterDisplayMode
```

C#

```
Microsoft.Web.WebView2.Core

    CoreWebView2
        LaunchingExternalUriScheme
        MemoryUsageTargetLevel

    CoreWebView2File
    CoreWebView2LaunchingExternalUriSchemeEventArgs
    CoreWebView2MemoryUsageTargetLevel
    CoreWebView2PermissionKind
        WindowManagement

    CoreWebView2Profile
        CookieManager
        IsGeneralAutofillEnabled
        IsPasswordAutosaveEnabled

    CoreWebView2Settings
        IsReputationCheckingRequired

    CoreWebView2WebMessageReceivedEventArgs
        AdditionalObjects
```

C#

```
Microsoft.Windows.Widgets.Providers

    IWidgetProvider2
    IWidgetProviderAnalytics
    IWidgetProviderErrors
    WidgetAnalyticsInfoReportedArgs
    WidgetCustomizationRequestedArgs
    WidgetErrorInfoReportedArgs
```

Related topics

- Latest stable channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK

- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Preview channel release notes for the Windows App SDK 1.3

Article • 05/23/2024

> **ⓘ Important**
>
> The preview channel is **not supported** for use in production environments, and apps that use the preview releases cannot be published to the Microsoft Store.

The preview channel includes releases of the Windows App SDK with preview channel features in late stages of development. Preview releases do not include experimental features and APIs but may still be subject to breaking changes before the next stable release.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on preview releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest preview channel release:

- Latest preview channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.3 Preview 1 (1.3.0-preview1)

This is the latest release of the preview channel for version 1.3. This release includes previews for new features across WinAppSDK and several performance, security, accessibility and reliability bug fixes.

In an existing Windows App SDK 1.2 (from the stable channel) app, you can update your Nuget package to 1.3.0-preview1 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Latest Windows App SDK downloads.

XAML Backdrop APIs

With properties built in to the XAML Window, Mica & Background Acrylic backdrops are now easier to use in your WinUI 3 app.

See the Xaml Backdrop API spec on GitHub for more information about the **Window.SystemBackdrop** property.

```
C#

public MainWindow()
{
    this.InitializeComponent();

    this.SystemBackdrop = new MicaBackdrop();
}
```

Window.AppWindow

Replacing several lines of boilerplate code, you're now able to use AppWindow APIs directly from a **Window** through `Window.AppWindow`. See the Window.AppWindow API spec on GitHub for additional background and usage information.

New features from across WinAppSDK

- `ApplicationModel.DynamicDependency`: `PackageDependency.PackageGraphRevisionId` that replaces the deprecated MddGetGenerationId.
- Environment Manager: `EnvironmentManager.AreChangesTracked` to inform you whether changes to the environment manager are able to be tracked in your application. See the Environment Manager API spec on GitHub for more information.
- MRT Core: A new event, `Application.ResourceManagerInitializing` allows your app to provide its own implementation of the `IResourceManager` interface, and gives you access to the ResourceManager that WinUI uses to resolve resource URIs. See the IResourceManager API spec on GitHub for more information.
- With the latest experimental VSIX, you're now able to convert your app between unpackaged and packaged through the Visual Studio menu instead of in your project file.
- A new event, `DebugSettings.XamlResourceReferenceFailed` is now raised when a referenced Static/ThemeResource lookup can't be resolved. This event gives access to a trace that details where the framework searched for that key in order to better

enable you to debug Static & ThemeResource lookup failures. For more information, see the API spec and issues 4972, 2350, and 6073 on GitHub.

- Deployment: To manage and repair the Windows App Runtime, `DeploymentRepairOptions` is now available as part of the `DeploymentManager`. For more information, see the Repair section of the Deployment API Spec on GitHub.

Known issues

- The Pivot control causes a runtime crash with a XAML parsing error. See issue #8160 on GitHub for more info.
- When the DatePicker or TimePicker flyout is opened, the app crashes.
- The `WindowsAppRuntime.ReleaseInfo` and `WindowsAppRuntime.RuntimeInfo` APIs introduced in 1.3 releases are not yet supported as they contain a critical bug.

Related topics

- Latest stable channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Preview channel release notes for the Windows App SDK 1.2

Article • 05/23/2024

> **ⓘ Important**
>
> The preview channel is **not supported** for use in production environments, and apps that use the preview releases cannot be published to the Microsoft Store.

The preview channel includes releases of the Windows App SDK with preview channel features in late stages of development. Preview releases do not include experimental features and APIs but may still be subject to breaking changes before the next stable release.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on preview releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest preview channel release:

- Latest preview channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.2 Preview 2 (1.2.0-preview2)

This is the latest release of the preview channel for version 1.2.

In an existing Windows App SDK 1.1 (from the stable channel) app, you can update your Nuget package to 1.2.0-preview2 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Latest Windows App SDK downloads.

> **ⓘ Important**

Visual Studio 2019 and .NET 5 is no longer supported for building C# apps (see **Windows App SDK 1.2 moving to C# WinRT 2.0**). You will need Visual Studio 2022 and one of the following .NET SDK versions: 6.0.401 (or later), 6.0.304, 6.0.109.

To update your .NET SDK version, install the latest version of Visual Studio 2022 or visit **.NET Downloads**. When updating your NuGet package without the required .NET SDK version, you will see an error like: "*This version of WindowsAppSDK requires .NET 6+ and WinRT.Runtime.dll version 2.0 or greater.*". To update the project from .NET 5.0 to .NET 6.0, open the project file and change "TargetFramework" to `net6.0` and "Target OS version" to the appropriate value (such as `net6.0-windows10.0.19041.0`).

Third-party Widgets in Windows

The Widgets Board was first introduced in Windows 11 and was limited to displaying first party Widgets. Widgets are small UI containers that display text and graphics on the Widgets Board, and are associated with an app installed on the device. With Windows App SDK, as third party developers you can now create Widgets for your packaged Win32 apps and test them locally on the Windows 11 Widgets Board.

For more information about Widgets, check out Widgets overview.

To get started developing Widgets for your app, check out the Widget providers development docs and Widgets design fundamentals for prerequisites, guidance and best practices.

Prerequisites for this release include:

- Developer mode enabled on the development machine.
- The development machine is running a version of Windows from the Dev Channel of the Windows Insider Program (WIP) with Widgets Board version 521.20060.1205.0 or above.

Known limitations when developing Widgets

- Third-party Widgets can only be tested locally on devices enrolled in WIP for this preview release. In Windows App SDK 1.2.0, users on retail versions of Windows can begin acquiring 3P Widgets via Microsoft Store shipped versions of your app.
- Widgets can only be created for packaged, Win32 apps. Widgets for Progressive Web Apps (PWA) are planned to be supported as part of Microsoft Edge 108.

Trimming for apps developed with .NET

.NET developers are now able to publish their WinAppSDK apps trimmed. With CsWinRT 2.0, the C#/WinRT projections distributed in WinAppSDK are now trimmable. Publishing your app trimmed can reduce the disk footprint of your app by removing any unused code from trimmable binaries. Apps may also see a startup performance improvement. With a basic Hello World app, we have seen a ~80% disk footprint improvement and a ~7% startup performance improvement when published trimmed. With WinUI gallery, we have seen a ~45% disk footprint improvement.

For more details on how to enable trimming, trimming limitations (such as reflection against trimmable types), and trim warnings, see Trim self-contained deployments and executables. Developers should thoroughly test their apps after trimming to ensure everything works as expected. For more information, check out issue 2478 on GitHub.

DisplayInformation

Win32 apps can now support High Dynamic Range (HDR) through the DisplayInformation class in WinAppSDK. The DisplayInformation class enables you to monitor display-related information for an application view. This includes events to allow clients to monitor for changes in the application view affecting which display(s) the view resides on, as well as changes in displays that can affect the application view.

Fixed issues in WinUI 3

- Acrylic backdrop material via DesktopAcrylicController is now supported in Windows 10 apps. For more information, check out issue 7112 on GitHub.
- Fixed issue causing App.UnhandledException to fail to be routed to the application. For more information, check out issue 5221 on GitHub.
- Fixed issue causing ListView styles to regress and change from WinAppSDK 1.1. For more information, check out issue 7666 on GitHub.

Other limitations and known issues (1.2.0-preview2)

> **Important**
>
> When you reference WinAppSDK 1.2 from a project you might see an error similar to: "*Detected package downgrade: Microsoft.Windows.SDK.BuildTools from 10.0.22621.1 to 10.0.22000.194.*", which is caused by incompatible references to the package from the app project and the WinAppSDK package. To resolve this you can

update the reference in the project to a more recent and compatible version of Microsoft.Windows.SDK.BuildTools, or simply remove the reference from your project. If you remove it from your project, a compatible version will be implicitly referenced by the WinAppSDK package.

- Building with Arm64 Visual Studio is not currently supported.
- Bootstrapper and Undocked RegFree WinRT auto-initializer defaults is (now) only set for projects that produce an executable (OutputType=Exe or WinExe). This prevents adding auto-initializers into class library DLLs and other non-executables by default.
  - If you need an auto-initializer in a non-executable (e.g. a test DLL loaded by a generic executable that doesn't initialize the Bootstrapper) you can explicitly enable an auto-initializer in your project via `<WindowsAppSdkBootstrapInitialize>true</WindowsAppSdkBootstrapInitialize>` or `<WindowsAppSdkUndockedRegFreeWinRTInitialize>true</WindowsAppSdkUndockedRegFreeWinRTInitialize>`.
- The version information APIs (ReleaseInfo and RuntimeInfo) can be called but return version 0 (not the actual version information).

Version 1.2 Preview 1 (1.2.0-preview1)

In an existing Windows App SDK 1.1 (from the stable channel) app, you can update your Nuget package to 1.2.0-preview1 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager).

For the updated runtime and MSIX, see Latest Windows App SDK downloads.

WinUI 3

WinUI 3 apps can play audio and video with the **MediaPlayerElement** and **MediaTransportControls** media playback controls. For more info on how and when to use media controls, see Media players.

WinUI 3 has been updated with the latest controls, styles, and behaviors from WinUI 2.8. These updates include the addition of the **InfoBadge** control, improvements to accessibility and high contrast mode, as well as bug fixes across controls. For more details, see the release notes for WinUI 2.7 and WinUI 2.8.

Known issue

ListView styles regressed and changed from WinAppSDK 1.1.

Notifications

AppNotificationBuilder introduced as an alternative to XML payload for creating and defining App Notifications.

For usage information, see the AppNotificationBuilder spec on GitHub.

Also see Quickstart: App notifications in the Windows App SDK for an example of how to create a desktop Windows application that sends and receives local app notifications.

Breaking change

For push notifications, when making a channel request call, apps will need to use the Azure Object ID instead of the Azure App ID. See Quickstart: Push notification in the Windows App SDK for details on finding your Azure Object ID.

Fixed issue

PushNotificationManager.IsSupported will perform a check for elevated mode. It will return `false` if the app is elevated.

Known limitations (Notifications)

- In **AppNotificationScenario**, `Urgent` is only supported for Windows builds 19041 and later. You can use **AppNotificationBuilder.IsUrgentScenarioSupported** to check whether the feature is available at runtime.
- In **AppNotificationButton**, `hint-toolTip` and `hint-buttonStyle` are only supported for builds 19041 and later. You can use **IsButtonStyleSupported** and **IsToolTipSupported** to check whether the feature is available at runtime.
- In **MediaPlayerElement**, when used in XAML markup for an unpackaged app, the **Source** property cannot be set with an ms-appx or ms-resource URI. As an alternative, set the Source using a file URI, or set from code.

Windowing

Full title bar customization is now available on Windows 10, version 1809 and later through the **AppWindowTitleBar** class. You can set **AppWindowTitleBar.ExtendsContentIntoTitleBar** to `true` to extend content into the

title bar area, and **SetDragRectangles** to define drag regions (in addition to other customization options).

If you've been using the **AppWindowTitleBar.IsCustomizationSupported** property to check whether you can call the **AppWindowTitleBar** APIs, it now returns `true` on supported Windows App SDK Windows 10 versions (1809 and later).

Known limitations (Windowing)

Simple title bar customizations are not supported on Windows 10. These include **BackgroundColor**, **InactiveBackgroundColor**, **ForegroundColor**, **InactiveForegroundColor** and **IconShowOptions**. If you call these properties, they will be ignored silently. All other **AppWindowTitleBar** APIs work in Windows 10, version 1809 and later. For the caption button color APIs (among others) and **Height**, **ExtendsContentIntoTitleBar** must be set to `true`, otherwise they will also be ignored silently.

Access control

Introduced security.accesscontrol.h with the **GetSecurityDescriptorForAppContainerNames** function to ease and streamline named object sharing between packaged processes and general Win32 APIs. This method takes a list of Package Family Names (PFNs) and access masks, and returns a security descriptor. For more information, see the GetSecurityDescriptorForAppContainerNames spec on GitHub.

Other limitations and known issues (1.2.0-preview1)

- .NET PublishSingleFile isn't supported.

Related topics

- Latest stable channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Preview channel release notes for the Windows App SDK 1.1

Article • 05/23/2024

> **ⓘ Important**
>
> The preview channel is **not supported** for use in production environments, and apps that use the preview releases cannot be published to the Microsoft Store.

The preview channel includes releases of the Windows App SDK with preview channel features in late stages of development. Preview releases do not include experimental features and APIs but may still be subject to breaking changes before the next stable release.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on preview releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest preview channel release:

- Latest preview channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.1 Preview 3 (1.1.0-preview3)

This is the latest release of the preview channel for version 1.1. It supports all preview channel features (see Features available by release channel).

In an existing app using Windows App SDK 1.0, you can update your Nuget package to 1.1.0-preview3 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager). Additionally, see Latest Windows App SDK downloads for the updated runtime and MSIX.

> **Note**
>
> For C# developers, one of the following .NET SDK versions (or later) is required: 6.0.202, 6.0.104, 5.0.407, 5.0.213. To update your .NET SDK version, visit .NET Downloads or update to the latest version of Visual Studio. Without the required .NET SDK version, when updating your NuGet package you will see an error like: "*This version of WindowsAppSDK requires WinRT.Runtime.dll version 1.6 or greater.*".

In addition to all of the Preview 2 features, the following sections describe new and updated features, limitations, and known issues for this release.

WinUI 3 (1.1.0-preview3)

Mica and Background Acrylic are now available for WinUI 3 applications.

For more information about these materials, check out Materials in Windows 11. Check out our sample code for applying Mica in C++ applications at Apply Mica or Acrylic materials in desktop apps for Windows 11 and in C# applications on GitHub as part of the WinUI Controls Gallery.

Notifications (1.1.0-preview3)

Fixed issues:

- In 1.1.0-preview1 and 1.1.0-preview2, some unpackaged apps will have seen their app icons incorrectly copied to AppData\LocalMicrosoftWindowsAppSDK. For this release, they will be copied to AppData\Local\Microsoft\WindowsAppSDK instead. To avoid leaking icons, you should manually delete the app icon at the incorrect path after updating to 1.1.0-preview3.
- App icon and app display name retrieval for app notifications via Shortcuts is now supported. This app icon will be prioritized over any icon specified in resource files.
- Support for push notifications for unpackaged apps has been restored (see **Limitations** for noted exception). We've introduced the PushNotificationManager::IsSupported API to check if your app supports push notifications.

Limitations:

- Notifications for an *elevated* unpackaged app is not supported. PushNotificationManager::IsSupported will not perform a check for elevated mode. However, we are working on supporting this in a future release.

MSIX packaging

We've enhanced MSIX adding new and extending existing functionality via the extension categories:

- windows.appExecutionAlias
- windows.customDesktopEventLog
- windows.dataShortcuts
- windows.fileTypeAssociation
- windows.fileTypeAssociation.iconHandler
- windows.folder
- windows.shortcut

These require the Windows App SDK framework package to be installed. See Latest Windows App SDK downloads to install the runtime.

Environment manager (1.1.0-preview3)

API set that allows developers to add, remove, and modify environment variables without having to directly use the registry API.

Clarification from 1.1 Preview 1: Automatic removal of any environment variable changes when an app that used environment manager is uninstalled is only available for packaged apps. Additionally, reverting environment variable changes requires installation of the Windows App SDK framework package, see Latest Windows App SDK downloads for the runtime.

Other known limitations

Regressions from 1.1 Preview 2:

- For .NET apps using MRT Core APIs and WinUI apps that *don't* deploy with single-project MSIX:
  - RESW and image files that were added to the project as Existing Items and previously automatically included to the PRIResource and Content ItemGroups, respectively, won't be included in those ItemGroups. As a result, these resources won't be indexed during PRI generation, so they won't be available during runtime.
    - Workaround: Manually include the resources in the project file and remove them from the None ItemGroup.
    - Alternative workaround: When available, upgrade your apps' .NET SDK to 6.0.300. See Version requirements for .NET SDK for additional information.

- For .NET apps that *don't* deploy with single-project MSIX:
  - If a file is added to the Content ItemGroup twice or more, then there will be a build error.
    - Workaround: Delete the duplicate inclusion/s or set EnableDefaultContentItems to false in the project file.

Both regressions will be restored in the next stable release.

Version 1.1 Preview 2 (1.1.0-preview2)

This is the second release of the preview channel for version 1.1. It supports all preview channel features (see Features available by release channel).

In an existing app using Windows App SDK 1.0, you can update your Nuget package to 1.1.0-preview2 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager). Additionally, see Latest Windows App SDK downloads for the updated runtime and MSIX.

> **Note**
>
> For C# developers, one of the following .NET SDK versions (or later) is required: 6.0.202, 6.0.104, 5.0.407, 5.0.213. To update your .NET SDK version, visit .NET Downloads or update to the latest version of Visual Studio. Without the required .NET SDK version, when updating your NuGet package you will see an error like: "*This version of WindowsAppSDK requires WinRT.Runtime.dll version 1.6 or greater.*".

In addition to all of the Preview 1 features, the following sections describe new and updated features, limitations, and known issues for this release.

Notifications (1.1.0-preview2)

Fixed issues:

- An app without package identity sending notifications will now see its app icon in the notification if the icon is a part of the app's resource. If the app resource has no icon, the Windows default app icon is used.
- A WinUI 3 app that's not running can now be background-activated via a notification.

Regression from 1.1 Preview 1: Push notifications support for unpackaged apps. Expected to be restored in the next release.

Known limitations:

- We've introduced the PushNotificationManager::IsSupported API to check if self-contained apps support push notifications. However, this API is not yet working as intended, so keep an eye out in the next preview release for full support of the IsSupported API.
- Some unpackaged apps will see their app icons incorrectly copied to AppData\LocalMicrosoftWindowsAppSDK. For the next release, they will be copied to AppData\Local\Microsoft\WindowsAppSDK instead. To avoid leaking icons, the developer should manually delete their app icon at the incorrect path after upgrading to the next release.
- App icon and app display name retrieval for notifications via Shortcuts is not supported. But we're working on supporting that in a future release.

Deployment

New features:

- Packaged apps can now force deploy the Windows App SDK runtime packages using the **DeploymentManager.Initialize** API.
- The Bootstrapper API now includes new options for improved usability and troubleshooting. For more details, see Use the Windows App SDK runtime for apps packaged with external location or unpackaged and Rich information on Bootstrap initialization failure.

Known limitations:

- Self-contained deployment is supported only on Windows 10, 1903 and later.

Windowing

For easier programming access to functionality that's implemented in `USER32.dll` (see Windows and messages), this release surfaces more of that functionality in **AppWindow** itself.

New features:

- Apps with existing windows have more control over how a window is shown, by calling **AppWindow.ShowOnceWithRequestedStartupState**—the equivalent of `ShowWindow(SW_SHOWDEFAULT)`.
- Apps can show, minimize, or restore a window and specify whether the window should be activated or not at the time the call is made.

- Apps can now set a window's client area size in Win32 coordinates.
- We've added APIs to support z-order management of windows.
- Apps drawing custom titlebars with **AppWindowTitleBar.ExtendsContentIntoTitleBar** can set a *PreferredTitleBarHeight* option. You have a choice of a *standard height* titlebar, or a *tall* titlebar that provides more room for interactive content. See Title bar in the Fluent design guidelines for advice about when to use a tall titlebar.

Known limitations:

- Tall titlebar support is available only on Windows 11. We are working to bring this downlevel along with other custom titlebar APIs.

WinUI 3 (1.1.0-preview2)

Fixed issues:

- Fixed issue causing C# apps with WebView2 to crash on launch when the C/C++ Runtime (CRT) isn't installed by upgrading the WebView2 SDK from 1020.46 to 1185.39.
- Fixed issue causing some rounded corners to show a gradient when they should be a solid color. For more information see issue 6076 & issue 6194 on GitHub.
- Fixed issue where updated styles were missing from generic.xaml.
- Fixed layout cycle issue causing an app to crash when scrolling to the end of a ListView. For more information see issue 6218 on GitHub.

Performance

C# applications have several performance improvements. For more details, see the C#/WinRT 1.6.1 release notes.

Version 1.1 Preview 1 (1.1.0-preview1)

This is the first release of the preview channel for version 1.1. It supports all preview channel features (see Features available by release channel).

In an existing app using Windows App SDK 1.0, you can update your Nuget package to 1.1.0-preview1 (see the **Update a package** section in Install and manage packages in Visual Studio using the NuGet Package Manager). Additionally, see Latest Windows App SDK downloads for the updated runtime and MSIX.

The following sections describe new and updated features, limitations, and known issues for this release.

WinUI 3 (1.1.0-preview1)

Known issue: Users are unable to drop an element when drag-and-drop is enabled.

Elevated (admin) support

Using Windows App SDK 1.1 Preview 1, apps (including WinUI 3) will be able to run with elevated privilege.

Important limitations:

- Currently available only on Windows 11. But we're evaluating bringing this support downlevel in a later release.

Known issues:

- WinUI 3 apps crash when dragging an element during a drag-and-drop interaction.

Self-contained deployment

Windows App SDK 1.1 will introduce support for self-contained deployment. Our Windows App SDK deployment overview details the differences between framework-dependent and self-contained deployment, and how to get started.

Known issues:

- A C++ app that's packaged needs to add the below to the bottom of its project file to work around a bug in the self-contained `.targets` file that removes framework references to VCLibs:

```
XML
<PropertyGroup>

<IncludeGetResolvedSDKReferences>true</IncludeGetResolvedSDKReferences>
</PropertyGroup>

<Target Name="_RemoveFrameworkReferences"

BeforeTargets="_ConvertItems;_CalculateInputsForGenerateCurrentProjectA
ppxManifest">
```

```
    <ItemGroup>
        <FrameworkSdkReference Remove="@(FrameworkSdkReference)"
Condition="'%(FrameworkSdkReference.SDKName)' ==
'Microsoft.WindowsAppRuntime.1.1-preview1'" />
    </ItemGroup>
</Target>
```

- Supported only on Windows 10, 1903 and later.

Notifications (1.1.0-preview1)

Developers of packaged (including packaged with external location) and unpackaged apps can now send Windows notifications.

New features:

- Support for app notifications for packaged and unpackaged apps. Full details on GitHub
  - Developers can send app notifications, also known as toast notifications, locally or from their own cloud service.
- Support for push notification for packaged and unpackaged apps. Full details on GitHub
  - Developers can send raw notifications or app notifications from their own cloud service.

Limitations:

- Apps published as self-contained may not have push notifications support. Keep an eye out in the next preview release for an **IsSupported** API to check for push notifications support.
- Apps that are unpackaged sending app notifications will not see their app icon in the app notification unless they are console applications. Console apps that are unpackaged should follow the patterns shown in the ToastNotificationsDemoApp sample.
- Windows App SDK runtime must be installed to support push notifications, see Latest Windows App SDK downloads for the installer.
- A WinUI 3 app that's not running can't be background-activated via a notification. But we're working on supporting that in a future release.

Environment manager (1.1.0-preview1)

API set that allows developers to add, remove, and modify environment variables without having to directly use the registry API.

New features:

- Provides automatic removal of any environment variables changes when an app that used environment manager is uninstalled.

Limitations:

- Currently unavailable in C# apps. But we're evaluating bringing this feature to C# apps in a later release.

Other limitations and known issues

- If you're using C# with 1.1.0 Preview 1, then you must use one of the following .NET SDK versions at a minimum: .NET SDK 6.0.201, 6.0.103, 5.0.212, or 5.0.406. To upgrade your .NET SDK, you can update to the latest version of Visual Studio, or visit Download .NET.

Related topics

- Latest stable channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Preview channel release notes for the Windows App SDK 1.0

Article • 05/23/2024

> **Important**
>
> The preview channel is **not supported** for use in production environments, and apps that use the preview releases cannot be published to the Microsoft Store.

The preview channel includes releases of the Windows App SDK with preview channel features in late stages of development. Preview releases do not include experimental features and APIs but may still be subject to breaking changes before the next stable release.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on preview releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest preview channel release:

- Latest preview channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.0 Preview 3 (1.0.0-preview3)

Preview 3 is the latest release of the preview channel for version 1.0 of the Windows App SDK. Preview 3 supports all preview channel features.

Download 1.0 Preview 3 Visual Studio extensions (VSIX)

> **Note**

If you have Windows App SDK Visual Studio extensions (VSIX) already installed, then uninstall them before installing a new version. For directions, see Manage extensions for Visual Studio.

From the table below you can download the Visual Studio extensions (VSIX) for the 1.0 Preview 3 release. For all versions, see Latest Windows App SDK downloads. If you haven't done so already, start by configuring your development environment, using the steps in Install tools for the Windows App SDK.

The extensions below are tailored for your programming language and version of Visual Studio.

Expand table

1.0 Preview 3 downloads	Description
C# Visual Studio 2019 extension	Build C# apps with the Windows App SDK Visual Studio 2019 extension.
C++ Visual Studio 2019 extension	Build C++ apps with the Windows App SDK Visual Studio 2019 extension.
C# Visual Studio 2022 extension	Build C# apps with the Windows App SDK Visual Studio 2022 extension.
C++ Visual Studio 2022 extension	Build C++ apps with the Windows App SDK Visual Studio 2022 extension.
The .exe installer, and MSIX packages	Deploy the Windows App SDK with your app using the `.exe` installer, and MSIX packages.

The following sections describe new and updated features, limitations, and known issues for 1.0 Preview 3.

WinUI 3 (1.0.0-preview3)

We now support deploying WinUI 3 apps without MSIX packaging. See Create your first WinUI 3 (Windows App SDK) project to configure your WinUI 3 application to support unpackaged deployment.

Important limitations:

- Unpackaged WinUI 3 applications are **supported only on Windows versions 1909 and later**.

- Unpackaged WinUI 3 applications are **supported on x86 and x64**; arm64 support will be added in the next stable release.
- **Single-project MSIX Packaging Tools** for Visual Studio 2019 or Visual Studio 2022 is required for unpackaged apps.
- In an unpackaged app, you might receive a prompt to install .NET 3.5; if you do, then you can ignore it.
- Some APIs are not currently supported in unpackaged apps. We're aiming to fix this in the next stable release. A few examples:
  - ApplicationData
  - StorageFile.GetFileFromApplicationUriAsync
  - ApiInformation (not supported on Windows 10)
  - Package.Current
- ListView, CalendarView, and GridView controls are using the incorrect styles, and we're aiming to fix this in the next stable release.

For more info, or to get started developing with WinUI 3, see:

- WinUI
- Create your first WinUI 3 (Windows App SDK) project

Other limitations and known issues:

- **Unpackaged apps are not supported on Windows 10 version 1809**. We're aiming to fix this in the next release in the stable channel.

- **C# Single-project MSIX app doesn't compile if C++ UWP Tools aren't installed**. If you have a C# Single-project MSIX project, then you'll need to install the **C++ (v14x) Universal Windows Platform Tools** optional component.

- This release introduces the **Blank App, Packaged (WinUI 3 in Desktop)** project templates for C# and C++. These templates enable you to build your app into an MSIX package without the use of a separate packaging project (see Package your app using single-project MSIX). These templates have some known issues in this release:

  - **Missing Publish menu item until you restart Visual Studio**. When creating a new app in both Visual Studio 2019 and Visual Studio 2022 using the **Blank App, Packaged (WinUI 3 in Desktop)** project template, the command to publish the project doesn't appear in the menu until you close and re-open Visual Studio.

  - **Error when adding C++ static/dynamic library project references to C++ apps using Single-project MSIX Packaging**. Visual Studio displays an error that the

project can't be added as a reference because the project types are not compatible.

  - **Error when referencing a custom user control in a class library project**. The application will crash with the error that the system can't find the path specified.

  - **C# or C++ template for Visual Studio 2019**. When you try to build the project, you'll encounter the error "The project doesn't know how to run the profile *project name*". To resolve this issue, install the Single-project MSIX Packaging Tools extension.

  - **C# template for Visual Studio 2019 and Visual Studio 2022**. In Visual Studio when you **Start Debugging** or **Start Without Debugging**, if your app doesn't deploy and run (and there's no feedback from Visual Studio), then click on the project node in **Solution Explorer** to select it, and try again.

  - **C# template for Visual Studio 2019 and Visual Studio 2022**. You will encounter the following error when you try to run or debug your project on your development computer: "The project needs to be deployed before we can debug. Please enable Deploy in the Configuration Manager." To resolve this issue, enable deployment for your project in **Configuration Manager**. For detailed instructions, see the Create your first WinUI 3 (Windows App SDK) project.

  - **C++ template for Visual Studio 2022 version 17.0 releases up to Preview 4**. You will encounter the following error the first time you try to run your project: "There were deployment errors". To resolve this issue, run or deploy your project a second time. This issue will be fixed in Visual Studio 2022 version 17.0 Preview 7.

- **No support for Any CPU build configuration**: When adding the Windows App SDK to an existing .NET application or component that supports **Any CPU**, you must specify the desired architecture: `x86`, `x64` or `arm64`.

- **C# projects using 1.0 Preview 3 must use the following .NET SDK**: .NET 6 SDK or later (see Download .NET and .NET 5 will reach End of Support on May 10, 2022).

- An alternative to **DispatcherQueue.TryEnqueue** (for resuming execution on the dispatcher queue thread) is to use the resume_foreground helper function in the Windows Implementation Library (WIL):

  1. Add a reference to your project to the Microsoft.Windows.ImplementationLibrary NuGet package.

2. Add `#include <wil/cppwinrt_helpers.h>` to your `pch.h`.
3. Add `#include <winrt/Microsoft.UI.Dispatching.h>` to your `pch.h`.
4. Now `co_await wil::resume_foreground(your_dispatcherqueue);`.

Important issue impacting 1.0 Preview 1 and Preview 2

Version 1.0 Preview 1 and Preview 2 of the Windows App SDK includes a mechanism to clean up any environment variable changes made by a packaged app when that app is uninstalled. This feature is in an experimental state, and the first release includes a known bug that can corrupt the system **PATH** environment variable.

Preview 1 and Preview 2 corrupts any **PATH** environment variable that contains the expansion character `%`. This happens whenever any packaged app is uninstalled, regardless of whether that app uses the Windows App SDK.

Also see PATH environment variable corruption issue.

Details

The system **PATH** entry is stored in the **Path** value in the following key in the Windows Registry:

`Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment`

If you launch the Registry Editor (`regedit.exe`), then you can copy and paste the path above into the breadcrumb bar (immediately below the menu bar), and press Enter to locate the key.

The **Path** value of that key *should* be of type **REG_EXPAND_SZ**, but the bug changes it to **REG_SZ**. And that makes the system **PATH** environment variable unusable if it contains the variable expansion character `%`.

Affected releases

- Windows App SDK 1.0 Preview 1 (1.0.0-preview1)
- Windows App SDK 1.0 Preview 2 (1.0.0-preview2)

Mitigation

To get your machine back into a good state, take the following steps:

1. Check whether the **PATH** in the Registry is corrupt and, if so, reset it by running the script below.

   You can accomplish step 1 with the following Windows PowerShell script (PowerShell Core won't work). Run it elevated.

   Powershell

   ```
   # This script must be run from an elevated Windows PowerShell
   # window (right-click Windows PowerShell in the Start menu,
   # and select Run as Administrator).

   # If the PATH in the Registry has been set to REG_SZ, then delete
   # it, and recreate it as REG_EXPAND_SZ.

   $EnvPath = 'Registry::HKLM\System\CurrentControlSet\Control\Session
   Manager\Environment'
   $Environment=Get-Item $EnvPath
   $PathKind = $Environment.GetValueKind('Path')

   if ($PathKind -ne 'ExpandString') {
     $Path = $Environment.GetValue('Path')
     Remove-ItemProperty $EnvPath -Name Path
     New-ItemProperty $EnvPath -Name Path -PropertyType ExpandString -
   Value $Path
   }
   ```

2. Uninstall all apps that use the Windows App SDK 1.0 Preview1 or Preview2 (see the script below).

3. Uninstall the Windows App SDK 1.0 Preview1/Preview2 packages, including the package that contains the bug (see the script below).

   You can accomplish steps 2 and 3 with the following Windows PowerShell script (PowerShell Core won't work). Run it elevated.

   Powershell

   ```
   # This script must be run from an elevated Windows PowerShell
   # window (right-click Windows PowerShell in the Start menu,
   # and select Run as Administrator).

   # Remove the Windows App SDK 1.0 Preview1/2, and all apps that use it.

   $winappsdk = "Microsoft.WindowsAppRuntime.1.0-preview*"
   Get-AppxPackage | Where-Object { $_.Dependencies -like $winappsdk } |
   ```

```
Remove-AppxPackage
Get-AppxPackage $winappsdk | Remove-AppxPackage
```

Fix in Windows App SDK 1.0 Preview 3

The feature causing the **PATH** environment variable to be corrupted will be removed in the upcoming Windows App SDK 1.0 Preview 3 release. It might be reintroduced at a later date, when all bugs have been fixed and thoroughly tested.

We recommend using version 1.0 Preview 3.

Version 1.0 Preview 2 (1.0.0-preview2)

> **ⓘ Important**
>
> Version 1.0 Preview 1 and Preview 2 contain a critical bug. If you've already installed one of these previews, see **how to resolve the issue**. We recommend using version **1.0 Preview 3** instead.

This is the latest release of the preview channel for version 1.0. It supports all preview channel features.

The following sections describe new and updated features, limitations, and known issues for this release.

WinUI 3 (1.0.0-preview2)

New updates:

- Controls have been updated to reflect the latest Windows styles from WinUI 2.6.
- Single-project MSIX is supported.
- WinUI 3 package can now target build 17763 and later. See issue #921 for more info.
- In-app toolbar is supported. However, the in-app toolbar and existing Hot Reload/Live Visual Tree support require the upcoming Visual Studio 17.0 Preview 5 release, available later in October.

Bug fixed: WebView2Runtime text is now localized.

For more info or to get started developing with WinUI 3, see:

- WinUI

- Create your first WinUI 3 (Windows App SDK) project

Windowing (1.0.0-preview2)

This release introduces updates to the AppWindow class. There are no major new features added in this release, but there are changes to method names, properties, and some return values have been removed. See the documentation and samples for detailed updates. If you worked with AppWindow in the 1.0 Experimental or 1.0 Preview 1 releases, expect some changes to your code.

New updates:

- The **AppWindowConfiguration** class has been removed. The properties of this class is now available on the AppWindow itself, or on the **Presenter** classes.
- Most `bool` return values for the WinRT API methods in this space has been removed and are now `void` since these methods would always succeed.
- The C# ImportDll calls are no longer needed for GetWindowIdFromWindow and GetWindowFromWindowId. Use the .NET wrapper methods available in the **Microsoft.UI.Win32Interop** class instead.

Important limitations:

- The Windows App SDK does not currently provide methods for attaching UI framework content to an AppWindow; you're limited to using the HWND interop access methods.
- Window title bar customization works only on Windows 11. Use the IsCustomizationSupported method to check for title bar customization feature support. We intend to bring this functionality down-level.

For more info, see Manage app windows (Windows App SDK).

Input (1.0.0-preview2)

New updates:

- Improved support for precision touchpad input.

Important limitations:

- All PointerPoint static factory functions have been removed: **GetCurrentPoint**, **GetCurrentPointTransformed**, **GetIntermediatePoints**, and **GetIntermediatePointsTransformed**.

- The Windows App SDK does not support retrieving **PointerPoint** objects with pointer IDs. Instead, you can use the **PointerPoint** member function **GetTransformedPoint** to retrieve a transformed version of an existing **PointerPoint** object. For intermediate points, you can use the **PointerEventArgs** member functions GetIntermediatePoints and **GetTransformedIntermediatePoints**. See the documentation for additional details.

MRT Core (1.0.0-preview2)

New updates:

- App developers can now opt out an image file or a RESW file from being indexed in the PRI file in .NET projects. See issue 980 for more info.

Important limitations:

- In .NET projects, resource files copy-pasted into the project folder aren't indexed on F5 if the app was already built. As a workaround, rebuild the app. See issue 1503 for more info].
- In .NET projects, existing resource files added from an external folder aren't indexed without manual setting of the Build Action. To work around this issue, set the Build Action in Visual Studio: **Content** for image files and **PRIResource** for RESW files. See issue 1504 for more info.

Deployment for unpackaged apps

New features:

- Windows App SDK 1.0 Preview 2 introduces a .NET wrapper for the bootstrapper API (see Use the Windows App SDK runtime for apps packaged with external location or unpackaged). The bootstrapper API is a set of native C/C++ functions that unpackaged apps must use to dynamically take a dependency on the Windows App SDK framework package at run time. The .NET wrapper provides an easier way to call the bootstrapper API from .NET apps, including Windows Forms and WPF apps. The .NET wrapper for the bootstrapper API is available in the Microsoft.WindowsAppRuntime.Bootstrap.Net.dll assembly, which is local to your app project. For more info about the .NET wrapper, see .NET wrapper library.
- Packaged apps can now use the deployment API to get the main and singleton MSIX packages installed on the machine. The main and singleton packages are part of the framework package that is installed with the app, but due to a limitation with the Windows application model, packaged apps will need to take this additional step in order to get those packages installed. For more info about

how the deployment API works, see Windows App SDK deployment guide for framework-dependent packaged apps.

Important limitations:

- The .NET wrapper for the bootstrapper API only is only intended for use by unpackaged .NET applications to simplify access to the Windows App SDK.
- Only MSIX packaged apps that are full trust or have the packageManagement restricted capability have the permission to use the deployment API to install the main and singleton package dependencies. Support for partial-trust packaged apps will be coming in later releases.
- When F5 testing an x86 app which uses the DeploymentManager.Initialize method on an x64 system, ensure that the x64 framework is first installed by running the WindowsAppRuntimeInstall.exe. Otherwise, you will encounter a **NOT_FOUND** error due to Visual Studio not deploying the x64 framework, which normally occurs through Store deployment or sideloading.

App lifecycle

Most of the App Lifecycle features already exist in the UWP platform, and have been brought into the Windows App SDK for use by desktop app types, especially unpackaged Console apps, Win32 apps, Windows Forms apps, and WPF apps. The Windows App SDK implementation of these features cannot be used in UWP apps, since there are equivalent features in the UWP platform itself.

Non-UWP apps can also be packaged into MSIX packages. While these apps can use some of the Windows App SDK App Lifecycle features, they must use the manifest approach where this is available. For example, they cannot use the Windows App SDK **RegisterForXXXActivation** APIs and must instead register for rich activation via the manifest.

All the constraints for packaged apps also apply to WinUI 3 apps that are packaged; and there are additional considerations as described below.

Important considerations:

- Rich activation: GetActivatedEventArgs
  - *Unpackaged apps*: Fully usable.
  - *Packaged apps*: Usable, but these apps can also use the platform `GetActivatedEventArgs`. Note that the platform defines Windows.ApplicationModel.AppInstance whereas the Windows App SDK defines Microsoft.Windows.AppLifecycle.AppInstance. And while UWP apps can use the `ActivatedEventArgs` classes, such as `FileActivatedEventArgs` and

`LaunchActivatedEventArgs`, apps that use the Windows App SDK AppLifecycle feature must use the interfaces not the classes (e.g, `IFileActivatedEventArgs`, `ILaunchActivatedEventArgs`, and so on).
    - *WinUI 3 apps*: WinUI 3's App.OnLaunched is given a Microsoft.UI.Xaml.LaunchActivatedEventArgs, whereas the platform `GetActivatedEventArgs` returns a Windows.ApplicationModel.IActivatedEventArgs, and the WindowsAppSDK `GetActivatedEventArgs` returns a Microsoft.Windows.AppLifecycle.AppActivationArguments object which can represent a platform `LaunchActivatedEventArgs`.
    - For more info, see Rich activation with the app lifecycle API.

- Register/Unregister for rich activation
    - *Unpackaged apps*: Fully usable.
    - *Packaged apps*: Not usable use the app's MSIX manifest instead.
    - For more info, see Rich activation with the app lifecycle API.

- Single/Multi-instancing
    - *Unpackaged apps*: Fully usable.
    - *Packaged apps*: Fully usable.
    - *WinUI 3 apps*: If an app wants to detect other instances and redirect an activation, it must do so as early as possible, and before initializing any windows, etc. To enable this, the app must define DISABLE_XAML_GENERATED_MAIN, and write a custom Main (C#) or WinMain (C++) where it can do the detection and redirection.
    - RedirectActivationToAsync is an async call, and you should not wait on an async call if your app is running in an STA. For Windows Forms and C# WinUI 3 apps, you can declare Main to be async, if necessary. For C++ WinUI 3 and C# WPF apps, you cannot declare Main to be async, so instead you need to move the redirect call to another thread to ensure you don't block the STA.
    - For more info, see App instancing with the app lifecycle API.

- Power/State notifications
    - *Unpackaged apps*: Fully usable.
    - *Packaged apps*: Fully usable.
    - For more info, see Power management with the app lifecycle API.

Known issue:

File Type associations incorrectly encode %1 to be %251 when setting the Verb handler's command line template, which crashes unpackaged Win32 apps. You can manually edit

the Registry value to be %1 instead as a partial workaround. If the target file path has a space in it, then it will still fail and there is no workaround for that scenario.

Other limitations and known issues:

- Version 1.0 Preview 1 and Preview 2 contain a critical bug. If you've already installed one of these previews, see how to resolve the issue. We recommend using version 1.0 Preview 3 instead.
- This release introduces the **Blank App, Packaged (WinUI 3 in Desktop)** templates for C# and C++ projects. These templates enable you to build your app into an MSIX package without the use of a separate packaging project. These templates have some known issues in this release:
  - **C# template for Visual Studio 2019.** You will encounter the error when you try to build the project: "The project doesn't know how to run the profile *project name*". To resolve this issue, install the Single-project MSIX Packaging Tools extension.
  - **C# template for Visual Studio 2019 and Visual Studio 2022.** You will encounter the following error when you try to run or debug your project on your development computer: "The project needs to be deployed before we can debug. Please enable Deploy in the Configuration Manager." To resolve this issue, enable deployment for your project in **Configuration Manager**. For detailed instructions, see the Create your first WinUI 3 (Windows App SDK) project.
  - **C++ template for Visual Studio 2019 and Visual Studio 2022.** In this release, these projects are restricted to calling the subset of Win32 APIs that can be called by UWP apps. The **Blank App, Packaged with WAP (WinUI 3 in Desktop)** template is not affected by this issue.
  - **C++ template for Visual Studio 2022 version 17.0 releases up to Preview 4.** You will encounter the following error the first time you try to run your project: "There were deployment errors". To resolve this issue, run or deploy your project a second time. This issue will be fixed in Visual Studio 2022 version 17.0 Preview 5.
- **Push notifications API (Microsoft.Windows.PushNotifications namespace) incorrectly included in the 1.0 Preview 2 release.** This is still an experimental feature, and to you use it you must install the 1.0 Experimental release instead. This feature will be removed from the upcoming 1.0 release.

- **App lifecycle API (Microsoft.Windows.AppLifecycle namespace) incorrectly includes the Experimental attribute in the 1.0 Preview 2 release.** The Experimental attribute will be removed from this API in the next release.
- **No support for Any CPU build configuration**: When adding the Windows App SDK to an existing .NET application or component that supports **Any CPU**, you must specify the desired architecture: `x86`, `x64` or `arm64`.
- **C# projects using 1.0 Preview 2 must use the following .NET SDK**: .NET 6 SDK or later (see Download .NET and .NET 5 will reach End of Support on May 10, 2022).
- An alternative to **DispatcherQueue.TryEnqueue** (for resuming execution on the dispatcher queue thread) is to use the resume_foreground helper function in the Windows Implementation Library (WIL):

    1. Add a reference to your project to the Microsoft.Windows.ImplementationLibrary NuGet package.
    2. Add `#include <wil/cppwinrt_helpers.h>` to your `pch.h`.
    3. Add `#include <winrt/Microsoft.UI.Dispatching.h>` to your `pch.h`.
    4. Now `co_await wil::resume_foreground(your_dispatcherqueue);`.

Version 1.0 Preview 1 (1.0.0-preview1)

> **Important**
>
> Version 1.0 Preview 1 and Preview 2 contain a critical bug. If you've already installed one of these previews, see **how to resolve the issue**. We recommend using version **1.0 Preview 3** instead.

This is the first release of the preview channel for version 1.0. It supports all preview channel features.

The following sections describe new and updated features, limitations, and known issues for this release.

WinUI 3 (1.0.0-preview1)

This release of WinUI 3 is focused on building towards 1.0 with bug fixes.

- **New features**: No new features in Preview 1.

- **Fixed issues**: For the full list of issues addressed in this release, see our GitHub repo.

For more info or to get started developing with WinUI 3, see:

- WinUI
- Create your first WinUI 3 (Windows App SDK) project

Windowing (1.0.0-preview1)

This release brings the Windowing API we introduced in Experimental 1 to a Preview state. There are no major new features areas in this release as it is focused on bugfixes, stability, and adjustments to the API signature. The noteworthy changes and additions are called out below.

New features:

- DisplayAreaWatcher has been added to the Windowing APIs. This allows a developer to observe changes in the display topology and enumerate DisplayAreas currently defined in the system.
- AppWindow now supports setting the window icon via the SetIcon method, and AppWindowTitleBar now supports selecting whether to show/hide the window icon along with the system menu via the IconShowOptions property.

Important limitations:

- This release of AppWindow is currently available only to Win32 apps (both packaged and unpackaged).
- The Windows App SDK does not currently provide methods for attaching UI framework content to an AppWindow; you're limited to using the HWND interop access methods.
- Window title bar customization works only on Windows 11. Use the IsCustomizationSupported method to check for title bar customization feature support. We intend to bring this functionality down-level.

For more info, see Manage app windows (Windows App SDK).

Input (1.0.0-preview1)

This release brings some new features to the Input API. The noteworthy changes and additions are called out below.

New features and updates:

- PointerPredictor gives input latency sensitive applications such inking applications the ability to predict input point locations up to 15ms in the future to achieve better latency and smooth animation.
- PenDeviceInterop enables you to acquire a reference to the Windows.Devices.Input.PenDevice by using the FromPointerPoint method.
- InputCursor provides an explicit distinction between preset system cursor types and custom cursor types by removing the "Custom" type present in `CoreCursor`, and splitting the `CoreCursor` object into separate objects.
- Updates to InputCursor APIs.
- GestureRecognizer moved out of experimental to Microsoft.UI.Input.
- PointerPoint moved out of experimental to Microsoft.UI.Input.
- Mouse, touch, and pen input fully supported for WinUI 3 drag and drop.

Important limitations:

- This release of Input APIs has known issues with Windows version 1809.
- MRT Core is not yet supported by any subtype of InputCursor.
- Direct use of the platform SDK API Windows.UI.Core.CoreDragOperation will not work with WinUI 3 applications.
- PointerPoint properties RawPosition and ContactRectRaw were removed because they referred to non-predicted values, which were the same as the normal values in the OS. Use Position and ContactRect instead. Pointer prediction is now handled with the Microsoft.UI.Input.PointerPredictor API object.

MRT Core (1.0.0-preview1)

Starting in version 1.0 Preview 1, MRT Core APIs have moved from the Microsoft.ApplicationModel.Resources namespace to the Microsoft.Windows.ApplicationModel.Resources namespace.

Other limitations and known issues:

- Version 1.0 Preview 1 and Preview 2 contain a critical bug. If you've already installed one of these previews, see how to resolve the issue. We recommend using version 1.0 Preview 3 instead.

- Projects created by using the C++ **Blank App, Packaged with WAP (WinUI 3 in Desktop)** project template encounter the following build error by default: `fatal error C1083: Cannot open include file: 'winrt/microsoft.ui.dispatching.co_await.h': No such file or directory`. To resolve this issue, remove the following line of code from the **pch.h** file. This issue will be fixed in the next release.

```
C++

#include <winrt/microsoft.ui.dispatching.co_await.h>
```

- An alternative to **DispatcherQueue.TryEnqueue** (for resuming execution on the dispatcher queue thread) is to use the resume_foreground helper function in the Windows Implementation Library (WIL):

    1. Add a reference to your project to the Microsoft.Windows.ImplementationLibrary NuGet package.
    2. Add `#include <wil/cppwinrt_helpers.h>` to your `pch.h`.
    3. Add `#include <winrt/Microsoft.UI.Dispatching.h>` to your `pch.h`.
    4. Now `co_await wil::resume_foreground(your_dispatcherqueue);`.

- **No support for Any CPU build configuration**: The Windows App SDK is written in native code and thus does not support **Any CPU** build configurations. The WinUI 3 templates in Visual Studio only allow architecture-specific builds. When adding the Windows App SDK to an existing .NET application or component that supports **Any CPU**, you must specify the desired architecture: `x86`, `x64` or `arm64`.

- **.NET apps must target build 18362 or later**: Your TFM must be set to `net6.0-windows10.0.18362` or later, and your packaging project's `<TargetPlatformVersion>` must be set to 18362 or later. For more info, see the known issue on GitHub.

- **C# projects using 1.0 Preview 1 must use the following .NET SDK**: .NET 6 SDK or later (see Download .NET and .NET 5 will reach End of Support on May 10, 2022).

- **Unpackaged apps not supported on Windows 10 version 1809**: This should be resolved in the next release.

Related topics

- Latest stable channel release notes for the Windows App SDK
- Latest experimental channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Experimental channel release notes for the Windows App SDK 1.7

Article • 04/16/2025

> **ⓘ Important**
>
> The experimental channel is **not supported** for use in production environments, and apps that use the experimental releases cannot be published to the Microsoft Store.

The experimental channel includes releases of the Windows App SDK with experimental channel features in early stages of development. APIs for experimental features have the Experimental attribute. If you call an experimental API in your code, you will receive a build-time warning. All APIs in the experimental channel are subject to extensive revisions and breaking changes. Experimental features and APIs may be removed from subsequent releases at any time.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on experimental releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest experimental channel release:

- Latest experimental channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.7 Experimental (1.7.0-experimental3)

Use on-device AI with Windows Copilot Runtime APIs

> **ⓘ Important**
>
> The underlying ML models required for these APIs currently require your device to be running the latest Windows 11 Insider Preview Build on the Dev Channel. Additionally,

these APIs require your device to be a Copilot+ PC. See **Copilot+ PCs Developer Guide** to learn more about these devices. APIs will throw an exception when called on devices lacking the necessary support.

The Windows Copilot Runtime (WCR) offers several AI-powered features and APIs for you to easily, efficiently, and responsibly use on-device AI models in your Windows apps. In this release we are making available several scenario focused APIs for you to leverage powerful capabilities without the need to find, run, or optimize your own Machine Learning (ML) models.

Learn more about responsible development practices used during WCR API development that you can also apply as you create AI-assisted features in the Developing Responsible Generative AI Applications and Features on Windows guidance.

Phi Silica

With **Phi Silica**, Microsoft's most powerful NPU-tuned local language model, you can generate text responses to broad user prompts with built in content moderation. You can also specify it to perform common tasks like summarizing a piece of text, rewriting a piece of text for clarity, and converting text to a table format. Phi Silica is optimized for efficiency and performance on Windows Copilot+ PCs devices while still offering many of the capabilities found in Large Language Models (LLMs).

See Get started with Phi Silica in the Windows App SDK and API ref for Phi Silica in the Windows App SDK for more information.

Text Recognition (OCR)

Text recognition, also known as optical character recognition (OCR), APIs in Windows Copilot Runtime (WCR) can detect and extract text within images and convert it into machine readable character streams. These APIs can identify characters, words, lines, polygonal text boundaries, and provide confidence levels for each match. The set of WCR AI-assisted APIs benefit from NPU-assisted acceleration to perform faster and more accurately than the legacy Windows.Media.Ocr.OcrEngine APIs.

See Get Started with Text Recognition (OCR) in the Windows App SDK and API ref for AI-backed Text Recognition (OCR) in the Windows App SDK for more information.

Image Super Resolution

Using the 'ImageScaler' APIs you can increase the sharpness and clarity of an image and upscale the image up to 8x its original resolution.

See What can I do with Image Super Resolution? and API ref for AI imaging features in the Windows App SDK to get started.

Image Description

The Image Description APIs can be used to generate a text description of an image. The APIs are configurable to specify the length and type of the text description. Image descriptions may include a short caption or a long description for users with accessibility needs.

> **Note**
>
> When calling ImageDescriptionGenerator.DescribeAsync() in a Debug build, an error may occur that can be mitigated by continuing the build in Visual Studio.

See What can I do with Image Description? and API ref for AI imaging features in the Windows App SDK to learn more.

Image Segmentation

Using Image Segmentation APIs you can identify specific objects within an image. The model takes both an image and a "hints" object and returns a mask of the identified object.

See What can I do with Image Segmentation? and API ref for AI imaging features in the Windows App SDK to get started.

New APIs for Windowing

New `AppWindow` APIs make it easier to control your app windows and create a great experience. New capabilities include using `EnablePlacementPersistence` to automatically remember the size and position of your windows, using `SetTaskBarIcon` and `SetTitleBarIcon` to independently set the taskbar and titlebar icons, using `AppWindowTitleBar.PreferredTheme` to set the light/dark theme of the titlebar, and using `OverlappedPresenter.PreferredMinimum/MaximumSize` to set a minimum or maximum size for the window.

Other notable changes

- The missing C# projections for the new `BadgeNotifications` have been added so these APIs are now usable from C#.
- A class registration issue which prevented using the new `AppNotificationConferencingConfig` API has been fixed. Note that this enhanced user

experience for video or audio calling in notifications is only available on the latest Windows Insider releases of Windows.

New APIs

This release includes the following new and modified experimental APIs compared to 1.7-experimental2:

```
Microsoft.Graphics.Imaging

    ImageBuffer
    ImageBufferContract
    ImageObjectExtractor
    ImageObjectExtractorContract
    ImageObjectExtractorHint
    ImageScaler
    ImageScalerContract
    PixelFormat
```

```
Microsoft.UI.Content

    ChildSiteLink
        AutomationOption
        ProcessKeyboardInput
        ProcessPointerInput

    ContentAutomationOptions
    ContentIsland
        AutomationOption
        ProcessKeyboardInput
        ProcessPointerInput

    ContentSite
        ProcessKeyboardInput
        ProcessPointerInput

    ContentSiteView
        AutomationOption
        ProcessKeyboardInput
        ProcessPointerInput

    DesktopPopupSiteBridge
        AutomationOption

    IContentSiteAutomation
        AutomationOption
```

```
    IContentSiteInput
    ReadOnlyDesktopSiteBridge
        ProcessKeyboardInput
        ProcessPointerInput
```

```
Microsoft.UI.Windowing

    AppWindow
        EnablePlacementPersistence
        EnablePlacementPersistence
        GetCurrentPlacement
        SaveCurrentPlacement
        SetPlacement
        SetTaskBarIcon
        SetTaskBarIcon
        SetTitleBarIcon
        SetTitleBarIcon

    AppWindowTitleBar
        PreferredTheme

    OverlappedPresenter
        PreferredMaximumSize
        PreferredMinimumSize
        SetPreferredBounds

    PlacementPersistenceBehaviorFlags
    TitleBarTheme
```

```
Microsoft.Windows.AI.ContentModeration

    ContentFilterOptions
    ContentFilterOptionsContract
    ImageContentFilterSeverity
    SeverityLevel
    TextContentFilterSeverity
```

```
Microsoft.Windows.AI.Generative

    ImageDescriptionContract
    ImageDescriptionGenerator
    ImageDescriptionScenario
    LanguageModel
    LanguageModelContext
```

```
    LanguageModelContract
    LanguageModelOptions
    LanguageModelResponse
    LanguageModelResponseStatus
    LanguageModelSkill
```

```
Microsoft.Windows.SemanticSearch

    EmbeddingVector
    SemanticSearchContract
```

```
Microsoft.Windows.Vision

    BoundingBox
    DetectedLineStyle
    OrientationDetectionOptions
    RecognizedLine
    RecognizedLineStyle
    RecognizedText
    RecognizedWord
    TextRecognitionContract
    TextRecognizer
    TextRecognizerOptions
```

```
Microsoft.Windows.Workloads

    WorkloadPriority
    WorkloadsContract
```

Version 1.7 Experimental (1.7.0-experimental2)

Background Task Registration

A new `BackgroundTaskBuilder` API enables registering background tasks for Windows App SDK apps. For more info, see GitHub #4831.

New Notifications Features

New badge notification support allows showing a number or glyph badge on your app in the taskbar. For more info, see GitHub #4926.

> **Important**
>
> In this release, the C# projections are missing for the new `BadgeNotifications` APIs, which prevents using them from C#. The APIs are available in C++.

Video or audio calling can have an enhanced user experience in notifications. For more info, see GitHub #4783.

> **Important**
>
> This functionality is only available on the latest Windows Insider releases of Windows.

Other notable changes

- `RichEditBox` now supports math mode, via `RichEditTextDocument.SetMathMode` and `RichEditTextDocument.SetMath`.
- New `CompatibilityOptions` support will allow more control over how servicing changes affect apps. For more info, see GitHub #4976.

New APIs

This release includes the following new and modified experimental APIs:

```
Microsoft.Security.Authentication.OAuth

    AuthFailure
    AuthRequestParams
    AuthRequestResult
    AuthResponse
    ClientAuthentication
    CodeChallengeMethodKind
    OAuth2Manager
    OAuthContract
    TokenFailure
    TokenFailureKind
    TokenRequestParams
    TokenRequestResult
    TokenResponse
```

```
Microsoft.UI.Composition

    CompositionNotificationDeferral
    CompositionProjectedShadow
        MaxOpacity
        MinOpacity
        OpacityFalloff

    CompositionProjectedShadowCaster
        AncestorClip
        Mask

    CompositionProjectedShadowDrawOrder
    CompositionProjectedShadowReceiver
        DrawOrder
        Mask
```

```
Microsoft.UI.Composition.Experimental

    ExpCompositionVisualSurface
    ExpExpressionNotificationProperty
    IExpCompositionPropertyChanged
    IExpCompositionPropertyChangedListener
    IExpCompositor
    IExpVisual
```

```
Microsoft.UI.Content

    AutomationTreeOptions
    ChildSiteLink
    ContentAppWindowBridge
    ContentDisplayOrientations
    ContentEnvironmentStateChangedEventArgs
        DidDisplayScaleChange

    ContentExternalBackdropLink
    ContentExternalOutputLink
    ContentIsland
        AutomationTreeOption
        Children
        Connected
        ConnectionInfo
        ConnectRemoteEndpoint
        CreateForSystemVisual
        Disconnected
```

```
        FindAllForSystemCompositor
        FragmentRootAutomationProvider
        GetBySystemVisual
        InputCapabilities
        IsRemoteEndpointConnected
        LocalToClientTransformMatrix
        LocalToParentTransformMatrix
        NextSiblingAutomationProvider
        ParentAutomationProvider
        Popups
        PreviousSiblingAutomationProvider
        Root

    ContentIslandEnvironment
        CurrentOrientation
        DisplayScale
        NativeOrientation
        ThemeChanged

    ContentIslandStateChangedEventArgs
        DidLocalToClientTransformMatrixChange
        DidLocalToParentTransformMatrixChange

    ContentSite
        InputCapabilities
        LocalToClientTransformMatrix
        LocalToParentTransformMatrix
        SetContentNodeParent
        TryGetAutomationProvider

    ContentSiteAutomationProviderRequestedEventArgs
    ContentSiteEnvironment
        CurrentOrientation
        DisplayScale
        NativeOrientation
        NotifyThemeChanged

    ContentSiteEnvironmentView
        DisplayScale

    ContentSiteView
        AutomationTreeOption
        InputCapabilities
        LocalToClientTransformMatrix
        LocalToParentTransformMatrix

    CoreWindowSiteBridge
    CoreWindowTopLevelWindowBridge
    DesktopChildSiteBridge
        AcceptRemoteEndpoint
        ConnectionInfo
        CreateWithDispatcherQueue
        IsRemoteEndpointConnected
        RemoteEndpointConnecting
        RemoteEndpointDisconnected
```

```
        RemoteEndpointRequestedStateChanged

    DesktopPopupSiteBridge
    DesktopSiteBridge
        TryCreatePopupSiteBridge

    EndpointConnectionEventArgs
    EndpointRequestedStateChangedEventArgs
    IContentIslandEndpointConnectionPrivate
    IContentNodeOwner
    IContentSiteAutomation
    IContentSiteBridgeEndpointConnectionPrivate
    IContentSiteInput
    IContentSiteLink
    IContentSiteLink2
    InputCapabilities
    PopupWindowSiteBridge
    ProcessStarter
    ReadOnlyDesktopSiteBridge
    SystemVisualSiteBridge
```

```
Microsoft.UI.Input

    InputFocusNavigationHost
        GetForSiteLink

    InputKeyboardSource
        GetForWindowId

    InputLayoutPolicy
    InputLightDismissAction
        GetForIsland

    InputPointerActivationBehavior
    InputPointerSource
        ActivationBehavior
        DirectManipulationHitTest
        GetForVisual
        GetForWindowId
        RemoveForVisual
        TouchHitTesting
        TrySetDeviceKinds

    ProximityEvaluation
    TouchHitTestingEventArgs
```

```
Microsoft.UI.Input.Experimental
```

```
    ExpInputSite
    ExpPointerPoint
```

```

Microsoft.UI.Text

    RichEditTextDocument
        GetMath
        SetMath
        SetMathMode
```

```

Microsoft.UI.Windowing

    AppWindow
        DefaultTitleBarShouldMatchAppModeTheme

    DisplayArea
        GetMetricsFromWindowId
```

```

Microsoft.UI.Xaml

    XamlIsland
    XamlRoot
        TryGetContentIsland
```

```

Microsoft.UI.Xaml.Automation.Peers

    AutomationEvents
        Notification

    InkCanvasAutomationPeer
    PagerControlAutomationPeer
```

```

Microsoft.UI.Xaml.Controls

    ContentDialogPlacement
        UnconstrainedPopup

    DoInkPresenterWork
```

```
    ElementFactory
    FlowLayout
    FlowLayoutAnchorInfo
    FlowLayoutLineAlignment
    FlowLayoutState
    IApplicationViewSpanningRects
    IndexPath
    InkCanvas
    ISelfPlayingAnimatedVisual
    ItemContainer
        CanUserInvoke
        CanUserInvokeProperty
        CanUserSelect
        CanUserSelectProperty
        ItemInvoked
        MultiSelectMode
        MultiSelectModeProperty

    ItemContainerInteractionTrigger
    ItemContainerInvokedEventArgs
    ItemContainerMultiSelectMode
    ItemContainerUserInvokeMode
    ItemContainerUserSelectMode
    LayoutPanel
    NumberBox
        InputScope
        InputScopeProperty
        TextAlignment
        TextAlignmentProperty

    PagerControl
    PagerControlButtonVisibility
    PagerControlDisplayMode
    PagerControlSelectedIndexChangedEventArgs
    PagerControlTemplateSettings
    ProgressRing
        DeterminateSource
        DeterminateSourceProperty
        IndeterminateSource
        IndeterminateSourceProperty

    RecyclePool
    RecyclingElementFactory
    ScrollingScrollStartingEventArgs
    ScrollingZoomStartingEventArgs
    ScrollView
        ScrollStarting
        ZoomStarting

    SelectionModel
    SelectionModelChildrenRequestedEventArgs
    SelectionModelSelectionChangedEventArgs
    SelectTemplateEventArgs
    StackLayout
        IsVirtualizationEnabled
```

```
        IsVirtualizationEnabledProperty

    StackLayoutState
    TitleBar
    TitleBarAutomationPeer
    TitleBarTemplateSettings
    UniformGridLayoutState
```

```
Microsoft.UI.Xaml.Controls.Primitives

    ScrollPresenter
        ScrollStarting
        ZoomStarting
```

```
Microsoft.Windows.ApplicationModel.Background

    BackgroundTaskBuilder
    BackgroundTaskContract
```

```
Microsoft.Windows.ApplicationModel.Background.UniversalBGTask

    Task
```

```
Microsoft.Windows.ApplicationModel.WindowsAppRuntime

    CompatibilityChange
    CompatibilityContract
    CompatibilityOptions
    DeploymentManager
        Repair

    DeploymentStatus
        PackageRepairFailed

    ReleaseInfo
    RuntimeInfo
    VersionInfoContract
    WindowsAppRuntimeVersion
```

```
Microsoft.Windows.AppNotifications

    AppNotification
        ConferencingConfig

    AppNotificationConferencingConfig
```

```
Microsoft.Windows.AppNotifications.Builder

    AppNotificationBuilder
        AddCameraPreview

    AppNotificationButton
        SetSettingStyle

    AppNotificationButtonSettingStyle
```

```
Microsoft.Windows.BadgeNotifications

    BadgeNotificationGlyph
    BadgeNotificationManager
    BadgeNotificationsContract
```

```
Microsoft.Windows.Media.Capture

    CameraCaptureUI
    CameraCaptureUIContract
    CameraCaptureUIMaxPhotoResolution
    CameraCaptureUIMaxVideoResolution
    CameraCaptureUIMode
    CameraCaptureUIPhotoCaptureSettings
    CameraCaptureUIPhotoFormat
    CameraCaptureUIVideoCaptureSettings
    CameraCaptureUIVideoFormat
```

```
Microsoft.Windows.Storage

    ApplicationData
        GetForUnpackaged
```

Version 1.7 Experimental (1.7.0-experimental1)

This is the latest release of the experimental channel.

To download, retarget your WinAppSDK NuGet version to `1.7.241114004-experimental1`.

New CameraCaptureUI API

A new CameraCaptureUI API makes it easier to capture photos and videos in your WinAppSDK app. For more info, see GitHub issue #4721.

New Authentication API

A new `OAuth2Manager` API provides a streamlined solution for web authentication, offering OAuth 2.0 capabilities with full feature parity across all Windows platforms supported by WinAppSDK. For more info, see GitHub issue #4772.

New Background Task support

A new `BackgroundTaskBuilder` API brings integrated support for background task registration to your WinAppSDK apps. For more info, see GitHub issue #4822.

New APIs for 1.7-experimental1

This release includes the following new and modified experimental APIs:

C#

```
Microsoft.Security.Authentication.OAuth

    AuthFailure
    AuthRequestParams
    AuthRequestResult
    AuthResponse
    ClientAuthentication
    CodeChallengeMethodKind
    OAuth2Manager
    OAuthContract
    TokenFailure
    TokenFailureKind
    TokenRequestParams
    TokenRequestResult
    TokenResponse
```

C#

```
Microsoft.UI.Composition

    CompositionNotificationDeferral
    CompositionProjectedShadow
        MaxOpacity
        MinOpacity
        OpacityFalloff

    CompositionProjectedShadowCaster
        AncestorClip
        Mask

    CompositionProjectedShadowDrawOrder
    CompositionProjectedShadowReceiver
        DrawOrder
        Mask
```

C#

```
Microsoft.UI.Composition.Experimental

    ExpCompositionVisualSurface
    ExpExpressionNotificationProperty
    IExpCompositionPropertyChanged
    IExpCompositionPropertyChangedListener
    IExpCompositor
    IExpVisual
```

C#

```
Microsoft.UI.Content

    AutomationOptions
    ChildContentLink
    ContentAppWindowBridge
    ContentDisplayOrientations
    ContentEnvironmentStateChangedEventArgs
        DidDisplayScaleChange

    ContentExternalBackdropLink
    ContentExternalOutputLink
    ContentIsland
        Children
        Compositor
        Connected
        ConnectionInfo
        ConnectRemoteEndpoint
        Create
        Disconnected
        FindAllForCompositor
```

```
        FragmentRootAutomationProvider
        GetByVisual
        IsRemoteEndpointConnected
        NextSiblingAutomationProvider
        ParentAutomationProvider
        PreviousSiblingAutomationProvider
        Root
        TransformMatrix

    ContentIslandEnvironment
        AutomationOption
        CurrentOrientation
        DisplayScale
        NativeOrientation
        ThemeChanged

    ContentSite
        Compositor
        SetContentNodeParent
        SetIsInputPassThrough
        SiteVisual
        TransformMatrix
        TryGetAutomationProvider

    ContentSiteAutomationProviderRequestedEventArgs
    ContentSiteEnvironment
        CurrentOrientation
        DisplayScale
        NativeOrientation
        NotifyThemeChanged

    ContentSiteView
        TransformMatrix

    CoreWindowSiteBridge
    CoreWindowTopLevelWindowBridge
    DesktopChildSiteBridge
        AcceptRemoteEndpoint
        ConnectionInfo
        IsRemoteEndpointConnected
        RemoteEndpointConnecting
        RemoteEndpointDisconnected
        RemoteEndpointRequestedStateChanged

    DesktopSiteBridge
        TryCreatePopupSiteBridge

    EndpointConnectionEventArgs
    EndpointRequestedStateChangedEventArgs
    IContentIslandEndpointConnectionPrivate
    IContentLink
    IContentNodeOwner
    IContentSiteBridge2
    IContentSiteBridgeAutomation
    IContentSiteBridgeEndpointConnectionPrivate
```

```
    PopupWindowSiteBridge
    ProcessStarter
    ReadOnlyDesktopSiteBridge
    SystemVisualSiteBridge
```

C#

```
Microsoft.UI.Input

    InputKeyboardSource
        GetForWindowId

    InputLayoutPolicy
    InputLightDismissAction
        GetForIsland

    InputPointerActivationBehavior
    InputPointerSource
        ActivationBehavior
        DirectManipulationHitTest
        GetForVisual
        GetForWindowId
        RemoveForVisual
        TouchHitTesting
        TrySetDeviceKinds

    ProximityEvaluation
    TouchHitTestingEventArgs
```

C#

```
Microsoft.UI.Input.Experimental

    ExpInputSite
    ExpPointerPoint
```

C#

```
Microsoft.UI.Windowing

    AppWindow
        DefaultTitleBarShouldMatchAppModeTheme

    DisplayArea
        GetMetricsFromWindowId
```

C#

```
Microsoft.UI.Xaml

    XamlIsland
    XamlRoot
        TryGetContentIsland
```

C#

```
Microsoft.UI.Xaml.Automation.Peers

    AutomationEvents
        Notification

    InkCanvasAutomationPeer
    PagerControlAutomationPeer
```

C#

```
Microsoft.UI.Xaml.Controls

    ContentDialogPlacement
        UnconstrainedPopup

    DoInkPresenterWork
    ElementFactory
    FlowLayout
    FlowLayoutAnchorInfo
    FlowLayoutLineAlignment
    FlowLayoutState
    IApplicationViewSpanningRects
    IndexPath
    InkCanvas
    ISelfPlayingAnimatedVisual
    ItemContainer
        CanUserInvoke
        CanUserInvokeProperty
        CanUserSelect
        CanUserSelectProperty
        ItemInvoked
        MultiSelectMode
        MultiSelectModeProperty

    ItemContainerInteractionTrigger
    ItemContainerInvokedEventArgs
    ItemContainerMultiSelectMode
    ItemContainerUserInvokeMode
    ItemContainerUserSelectMode
    LayoutPanel
    NumberBox
        InputScope
        InputScopeProperty
```

```
        TextAlignment
        TextAlignmentProperty

    PagerControl
    PagerControlButtonVisibility
    PagerControlDisplayMode
    PagerControlSelectedIndexChangedEventArgs
    PagerControlTemplateSettings
    ProgressRing
        DeterminateSource
        DeterminateSourceProperty
        IndeterminateSource
        IndeterminateSourceProperty

    RecyclePool
    RecyclingElementFactory
    ScrollingScrollStartingEventArgs
    ScrollingZoomStartingEventArgs
    ScrollView
        ScrollStarting
        ZoomStarting

    SelectionModel
    SelectionModelChildrenRequestedEventArgs
    SelectionModelSelectionChangedEventArgs
    SelectTemplateEventArgs
    StackLayout
        IsVirtualizationEnabled
        IsVirtualizationEnabledProperty

    StackLayoutState
    TitleBar
    TitleBarAutomationPeer
    TitleBarTemplateSettings
    UniformGridLayoutState
```

C#

```
Microsoft.UI.Xaml.Controls.Primitives

    ScrollPresenter
        ScrollStarting
        ZoomStarting
```

C#

```
Microsoft.Windows.ApplicationModel.Background

    BackgroundTaskBuilder
    BackgroundTaskContract
```

C#

```
Microsoft.Windows.ApplicationModel.Background.UniversalBGTask

    Task
```

C#

```
Microsoft.Windows.ApplicationModel.WindowsAppRuntime

    DeploymentManager
        Repair

    DeploymentStatus
        PackageRepairFailed

    ReleaseInfo
    RuntimeInfo
    VersionInfoContract
```

C#

```
Microsoft.Windows.Media.Capture

    CameraCaptureUI
    CameraCaptureUIContract
    CameraCaptureUIMaxPhotoResolution
    CameraCaptureUIMaxVideoResolution
    CameraCaptureUIMode
    CameraCaptureUIPhotoCaptureSettings
    CameraCaptureUIPhotoFormat
    CameraCaptureUIVideoCaptureSettings
    CameraCaptureUIVideoFormat
```

C#

```
Microsoft.Windows.Storage

    ApplicationData
        GetForUnpackaged
```

Bug fixes

This release includes the following bug fixes:

- Changed `SplitButton` so touch input now matches the behavior of mouse input. For more info, see GitHub issue #178.

- Changed cascading menus so sub menus now open immediately if clicked. For more info, see GitHub issue #939.
- Fixed an issue where opening a `ComboBox` which is in a flyout closes all flyouts. For more info, see GitHub issue #1467.
- Fixed an issue where `SwipeControl` would randomly crash in a `ListView`. For more info, see GitHub issue #2527.
- Fixed an issue where drag-and-drop only a `ListViewItem` would leave it in the wrong visual state. For more info, see GitHub issue #3458.
- Fixed an issue in `StackLayout` so that it respects the ItemsRepeater.HorizontalAlignment and ItemsRepeater.VerticalAlignment properties (when StackLayout.Orientation is Vertical and Horizontal respectively). The old layout behaved as if the ItemsRepeater alignment was Stretch. With the fix, the layout results in items aligned to the right when the Right alignment is used, for example. For more info, see GitHub issue #3842.
- Fixed an issue where deleting items in the `ItemsRepeater`'s source would not generate items which moved up into view. For more info, see GitHub issue #6661.
- Fixed an issue where the right Alt key would not show keytips for Access Keys. For more info, see GitHub issue #8447. **Note:** This may result in key events for the right Alt key no longer being delivered to handles in the app or controls.
- Fixed a crash where `UniformGridLayout` would sometimes pick a wrong layout anchor and cause infinite layout passes when scrolling backwards. For more info, see GitHub issue #9199.
- Fixed an issue where setting `NavigationFailedEventArgs.Handled` to True would still throw an exception. For more info, see GitHub issue #9632.
- Fixed an issue where `TabView` would not apply any specified `CornerRadius`. For more info, see GitHub issue #9846.
- Fixed a potential layout cycle crash in `StackLayout`. For more info, see GitHub issue #9852.
- Fixed a potential crash in `ItemsView` when removing items. For more info, see GitHub issue #9868.

Related topics

- Stable channel
- Preview channel
- Install tools for the Windows App SDK
- Create your first WinUI 3 project
- Use the Windows App SDK in an existing project
- Deploy apps that use the Windows App SDK

Experimental channel release notes for the Windows App SDK 1.6

Article • 11/21/2024

> **ⓘ Important**
>
> The experimental channel is **not supported** for use in production environments, and apps that use the experimental releases cannot be published to the Microsoft Store.

The experimental channel provides releases of the Windows App SDK that include experimental channel features that are in early stages of development. APIs for experimental features have the Experimental attribute. If you call an experimental API in your code, you will receive a build-time warning. All APIs in the experimental channel might have breaking changes in future releases, but experimental APIs are especially subject to change. Experimental features may be removed from the next release, or may never be released.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on experimental releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest experimental channel release:

- Latest experimental channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.6 Experimental (1.6.0-experimental2)

This is the latest release of the experimental channel.

To download, retarget your WinAppSDK NuGet version to `1.6.240701003-experimental2`.

> **Note**
>
> Phi Silica and OCR APIs are not included in this release. These will be coming in a future 1.6 release.

Native AOT support updates

In 1.6-experimental1, the XAML compiler was generating `XamlTypeInfo.g.cs` with code that wasn't safe for AOT/Trimming. This relates to GitHub issue #9675, though it does not fully fix that issue.

Changed Edge WebView2 SDK Integration

The Windows App SDK now consumes the Edge WebView2 SDK as a NuGet reference rather than embedding a hardcoded version of the Edge WebView2 SDK. The new model allows apps to choose a newer version of the `Microsoft.Web.WebView2` package instead of being limited to the version with which the Windows App SDK was built. The new model also allows apps to reference NuGet packages which also reference the Edge WebView2 SDK. For more info, see GitHub issue #5689.

New Package Deployment APIs

The Package Management API has received several enhancements including Is*ReadyOrNewerAvailable*(), EnsureReadyOptions.RegisterNewerIfAvailable, Is*Provisioned*(), IsPackageRegistrationPending(), and several bug fixes. See PackageManagement.md and Pull Request #4453 for more details.

Other notable changes

- Starting with 1.6-experimental2, the latest WinUI 3 source will now publish to the main branch in the microsoft-ui-xaml GitHub repo, which will enable source searching in that repo.
- We added a new `ColorHelper.ToDisplayName()` API, filling that gap from UWP.
  - **Known issue:** Some language translations have character encoding issues. This will be fixed in the next 1.6 release.
- Added a new `Microsoft.Windows.Globalization.ApplicationLanguages` class, which notably includes a new `PrimaryLanguageOverride` feature. For more info, see GitHub issue #4523.

- New extensions enable Widget Providers to provide Widgets with web content and announcements for Widgets.

New APIs for 1.6-experimental2

1.6-experimental2 includes the following new APIs. These APIs are not experimental, but are not yet included in a stable release version of the WinAppSDK.

```
C#

Microsoft.UI.Xaml.Controls

    PipsPager
        WrapMode
        WrapModeProperty

    PipsPagerWrapMode
```

```
C#

Microsoft.Windows.Globalization

    ApplicationLanguages
```

```
C#

Microsoft.Windows.Management.Deployment

    EnsureReadyOptions
        RegisterNewerIfAvailable

    PackageDeploymentFeature
    PackageDeploymentManager
        IsPackageDeploymentFeatureSupported
        IsPackageProvisioned
        IsPackageProvisionedByUri
        IsPackageReadyOrNewerAvailable
        IsPackageReadyOrNewerAvailableByUri
        IsPackageSetProvisioned
        IsPackageSetReadyOrNewerAvailable

    PackageReadyOrNewerAvailableStatus
```

Additional 1.6-experimental2 APIs

This release includes the following new and modified experimental APIs:

C#

```
Microsoft.UI

    ColorHelper
        ToDisplayName
```

C#

```
Microsoft.UI.Composition

    CompositionNotificationDeferral
```

C#

```
Microsoft.UI.Composition.Experimental

    ExpCompositionVisualSurface
    ExpExpressionNotificationProperty
    IExpCompositionPropertyChanged
    IExpCompositionPropertyChangedListener
    IExpCompositor
    IExpVisual
```

C#

```
Microsoft.UI.Content

    AutomationOptions
    ChildContentLink
    ContentAppWindowBridge
    ContentDisplayOrientations
    ContentExternalBackdropLink
    ContentExternalOutputLink
    ContentIsland
        Children
        Compositor
        Connected
        ConnectionInfo
        ConnectRemoteEndpoint
        Create
        Disconnected
        FindAllForCompositor
        FragmentRootAutomationProvider
        GetByVisual
        IsRemoteEndpointConnected
        NextSiblingAutomationProvider
        Offset
        ParentAutomationProvider
        PreviousSiblingAutomationProvider
```

```
        Root
        RotationAngleInDegrees

    ContentIslandEnvironment
        AutomationOption
        CurrentOrientation
        DisplayScale
        NativeOrientation
        ThemeChanged

    ContentSite
        Compositor
        Offset
        RotationAngleInDegrees
        SetContentNodeParent
        SetIsInputPassThrough
        SiteVisual
        TryGetAutomationProvider

    ContentSiteAutomationProviderRequestedEventArgs
    ContentSiteEnvironment
        CurrentOrientation
        DisplayScale
        NativeOrientation
        NotifyThemeChanged

    ContentSiteView
        Offset
        RotationAngleInDegrees

    CoreWindowSiteBridge
    CoreWindowTopLevelWindowBridge
    DesktopChildSiteBridge
        AcceptRemoteEndpoint
        ConnectionInfo
        IsRemoteEndpointConnected
        RemoteEndpointConnecting
        RemoteEndpointDisconnected
        RemoteEndpointRequestedStateChanged

    DesktopSiteBridge
        TryCreatePopupSiteBridge

    EndpointConnectionEventArgs
    EndpointRequestedStateChangedEventArgs
    IContentIslandEndpointConnectionPrivate
    IContentLink
    IContentNodeOwner
    IContentSiteBridge2
    IContentSiteBridgeAutomation
    IContentSiteBridgeEndpointConnectionPrivate
    PopupWindowSiteBridge
    ProcessStarter
    ReadOnlyDesktopSiteBridge
    SystemVisualSiteBridge
```

C#

```
Microsoft.UI.Input

    EnteredMoveSizeEventArgs
    EnteringMoveSizeEventArgs
    ExitedMoveSizeEventArgs
    InputKeyboardSource
        GetForWindowId

    InputLayoutPolicy
    InputLightDismissAction
        GetForIsland

    InputNonClientPointerSource
        EnteredMoveSize
        EnteringMoveSize
        ExitedMoveSize
        WindowRectChanged
        WindowRectChanging

    InputPointerActivationBehavior
    InputPointerSource
        ActivationBehavior
        DirectManipulationHitTest
        GetForVisual
        GetForWindowId
        RemoveForVisual
        TouchHitTesting
        TrySetDeviceKinds

    MoveSizeOperation
    ProximityEvaluation
    TouchHitTestingEventArgs
    WindowRectChangedEventArgs
    WindowRectChangingEventArgs
```

C#

```
Microsoft.UI.Input.Experimental

    ExpInputSite
    ExpPointerPoint
```

C#

```
Microsoft.UI.Windowing

    AppWindow
        DefaultTitleBarShouldMatchAppModeTheme
```

```
    DisplayArea
        GetMetricsFromWindowId
```

C#

```
Microsoft.UI.Xaml

    XamlIsland
    XamlRoot
        CoordinateConverter
        TryGetContentIsland
```

C#

```
Microsoft.UI.Xaml.Automation.Peers

    PagerControlAutomationPeer
    ScrollPresenterAutomationPeer
```

C#

```
Microsoft.UI.Xaml.Controls

    ContentDialogPlacement
        UnconstrainedPopup

    ElementFactory
    FlowLayout
    FlowLayoutAnchorInfo
    FlowLayoutLineAlignment
    FlowLayoutState
    IApplicationViewSpanningRects
    IndexPath
    ISelfPlayingAnimatedVisual
    ItemContainer
        CanUserInvoke
        CanUserInvokeProperty
        CanUserSelect
        CanUserSelectProperty
        ItemInvoked
        MultiSelectMode
        MultiSelectModeProperty

    ItemContainerInteractionTrigger
    ItemContainerInvokedEventArgs
    ItemContainerMultiSelectMode
    ItemContainerUserInvokeMode
    ItemContainerUserSelectMode
    LayoutPanel
    NumberBox
```

```
        InputScope
        InputScopeProperty
        TextAlignment
        TextAlignmentProperty

    PagerControl
    PagerControlButtonVisibility
    PagerControlDisplayMode
    PagerControlSelectedIndexChangedEventArgs
    PagerControlTemplateSettings
    ProgressRing
        DeterminateSource
        DeterminateSourceProperty
        IndeterminateSource
        IndeterminateSourceProperty

    RecyclePool
    RecyclingElementFactory
    ScrollingViewChangingEventArgs
    ScrollView
        ViewChanging

    SelectionModel
    SelectionModelChildrenRequestedEventArgs
    SelectionModelSelectionChangedEventArgs
    SelectTemplateEventArgs
    StackLayout
        IsVirtualizationEnabled
        IsVirtualizationEnabledProperty

    StackLayoutState
    TabView
        CanTearOutTabs
        CanTearOutTabsProperty
        ExternalTornOutTabsDropped
        ExternalTornOutTabsDropping
        TabTearOutRequested
        TabTearOutWindowRequested

    TabViewExternalTornOutTabsDroppedEventArgs
    TabViewExternalTornOutTabsDroppingEventArgs
    TabViewTabTearOutRequestedEventArgs
    TabViewTabTearOutWindowRequestedEventArgs
    TitleBar
    TitleBarAutomationPeer
    TitleBarTemplateSettings
    UniformGridLayoutState
```

C#

```
Microsoft.UI.Xaml.Controls.Primitives
```

```
    ScrollPresenter
        ViewChanging
```

C#

```
Microsoft.Windows.ApplicationModel.WindowsAppRuntime

    DeploymentManager
        Repair

    DeploymentStatus
        PackageRepairFailed

    ReleaseInfo
    RuntimeInfo
    VersionInfoContract
```

C#

```
Microsoft.Windows.Widgets.Feeds.Providers

    FeedManager
        TryRemoveAnnouncementById

    IFeedManager3
```

C#

```
Microsoft.Windows.Widgets.Notifications

    WidgetAnnouncement
    WidgetAnnouncementInvokedArgs
```

C#

```
Microsoft.Windows.Widgets.Providers

    IWidgetAnnouncementInvokedTarget
    IWidgetManager2
    IWidgetManager3
    IWidgetProviderMessage
    IWidgetResourceProvider
    WidgetManager
        SendMessageToContent
        TryRemoveAnnouncementById
        TryShowAnnouncement

    WidgetMessageReceivedArgs
    WidgetResourceRequest
```

```
    WidgetResourceRequestedArgs
    WidgetResourceResponse
```

Known issues

- For TabView tab tear-out, pointer input behavior for *CanTearOutTabs* is incorrect on monitors with scale factor different from 100%. This will be fixed in the next 1.6 release.

Bug fixes

- Fixed an issue from 1.6-experimental1 where `NumberBox` wasn't using the correct foreground and background colors. For more info, see GitHub issue #9714.
- Fixed an issue where duplicate `KeyUp` events were raised for arrow and tab keys. For more info, see GitHub issue #9399.
- Fixed an issue where the `PowerManager.SystemSuspendStatusChanged` event was unusable to get the `SystemSuspendStatus`. For more info, see GitHub issue #2833.
- Fixed an issue where initial keyboard focus was not correctly given to a `WebView2` when that was the only control in the window.
- Fixed an issue when using `ExtendsContentIntoTitleBar=true` where the Min/Max/Close buttons did not correctly appear in the UI Automation, which prevented Voice Access from showing numbers for those buttons.
- Fixed an issue where an app might crash in a lock check due to unexpected reentrancy.
- Fixed an issue from 1.6-experimental1 where `TitleBar` only showed the Icon and Title because some elements did not show up on load.
- Fixed an issue where `Hyperlink` colors did not correctly update when switching into a high contrast theme.
- Fixed an issue where changing the collection of a `ListView` in a background window may incorrectly move that window to the foreground and take focus.
- Fixed an issue from 1.6-experimental1 where setting `AcrylicBrush.TintLuminosityOpacity` in .xaml in a class library project would crash with a type conversion error.
- Fixed an issue where calling `ItemsRepeater.StartBringIntoView` could sometimes cause items to disappear.
- Fixed an issue where touching and dragging on a `Button` in a `ScrollViewer` would leave it in a pressed state.

- Updated IntelliSense, which was missing information for many newer types and members.

Version 1.6 Experimental (1.6.0-experimental1)

This is the latest release of the experimental channel.

To download, retarget your WinAppSDK NuGet version to `1.6.240531000-experimental1`.

Required C# project changes for 1.6-experimental1

In 1.6-experimental1, Windows App SDK managed apps require Microsoft.Windows.SDK.NET.Ref `*.*.*.35-preview` (or later), which can be specified via WindowsSdkPackageVersion in your `csproj` file. For example:

XML

```
<Project Sdk="Microsoft.NET.Sdk">
   <PropertyGroup>
       <OutputType>WinExe</OutputType>
       <TargetFramework>net8.0-windows10.0.22621.0</TargetFramework>
       <TargetPlatformMinVersion>10.0.17763.0</TargetPlatformMinVersion>
       <WindowsSdkPackageVersion>10.0.22621.35-
preview</WindowsSdkPackageVersion>
   <PropertyGroup>
   ...
```

In addition, Windows App SDK managed apps using C#/WinRT should update to Microsoft.Windows.CsWinRT `2.1.0-prerelease.240602.1` (or later).

Native AOT support

> ⓘ **Note**
>
> For Windows App SDK 1.6.0 stable, the following guidance is obsolete. Projects should instead simply set `PublishAot` to **true** unconditionally.

The .NET `PublishAot` project property is now supported for native Ahead-Of-Time compilation. For details, see Native AOT Deployment. Because AOT builds on Trimming support, much of the following trimming-related guidance applies to AOT as well.

For `PublishAot` support, in addition to the C# project changes described in the previous section you'll also need a package reference to Microsoft.Windows.CsWinRT `2.1.0-prerelease.240602.1` (or later) to enable the source generator from that package.

Because the Windows App SDK invokes publishing targets when F5 deploying, we recommend enabling `PublishAot` at NuGet restore time by adding this to your `csproj` file:

XML

```
<PublishAot
Condition="'$(ExcludeRestorePackageImports)'=='true'">true</PublishAot>
```

In addition, we recommend conditionally enabling `PublishAot` when publishing release configurations, either in publish profiles or the project:

XML

```
<PublishAot Condition="'$(Configuration)'=='Release'">true</PublishAot>
```

Resolving AOT Issues

In this release, the developer is responsible for ensuring that all types are properly rooted to avoid trimming (such as with reflection-based `{Binding}` targets). Later releases will enhance both C#/WinRT and the XAML Compiler to automate rooting where possible, alert developers to trimming risks, and provide mechanisms to resolve.

Partial Classes

C#/WinRT also includes `PublishAot` support in version 2.1.0-prerelease.240602.1. To enable a class for AOT publishing with C#/WinRT, it must first be marked `partial`. This allows the C#/WinRT AOT source analyzer to attribute the classes for static analysis. Only classes (which contain methods, the targets of trimming) require this attribute.

Reflection-Free Techniques

To enable AOT compatibility, reflection-based techniques should be replaced with statically typed serialization, AppContext.BaseDirectory, typeof(), etc. For details, see Introduction to trim warnings.

Rooting Types

Until full support for `{Binding}` is implemented, types may be preserved from trimming as follows: Given project `P` consuming assembly `A` with type `T` in namespace `N`, which is only dynamically referenced (so normally trimmed), `T` can be preserved via:

`P.csproj`:

XML

```xml
<ItemGroup>
    <TrimmerRootDescriptor Include="ILLink.Descriptors.xml" />
</ItemGroup>
```

`ILLink.Descriptors.xml`:

XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<linker>
    <assembly fullname="A">
        <type fullname="N.T" preserve="all" />
    </assembly>
</linker>
```

For complete root descriptor XML expression syntax, see Root Descriptors.

> ⓘ **Note**
>
> Dependency packages that have not yet adopted AOT support may exhibit runtime issues.

Improved TabView tab tear-out

`TabView` supports a new `CanTearOutTabs` mode which provides an enhanced experience for dragging tabs and dragging out to a new window. When this new option is enabled, tab dragging is very much like the tab drag experience in Edge and Chrome, where a new window is immediately created during the drag, allowing the user to drag it to the edge of the screen to maximize or snap the window in one smooth motion. This implementation also doesn't use drag-and-drop APIs, so it isn't impacted by any limitations in those APIs. Notably, tab tear-out is supported in processes running elevated as Administrator.

Known issue: In this release, pointer input behavior for `CanTearOutTabs` is incorrect on monitors with scale factor different than 100%. This will be fixed in the next 1.6 release.

New TitleBar control

A new `TitleBar` control makes it easy to create a great, customizable titlebar for your app with the following features:

- Configurable Icon, Title, and Subtitle properties
- An integrated back button
- The ability to add a custom control like a search box
- Automatic hiding and showing of elements based on window width
- Affordances for showing active or inactive window state
- Support for default titlebar features including draggable regions in empty areas, theme responsiveness, default caption (min/max/close) buttons, and built-in accessibility support

The `TitleBar` control is designed to support various combinations of titlebars, making it flexible to create the experience you want without having to write a lot of custom code. We took feedback from the community toolkit titlebar prototype and look forward to additional feedback!

Known issue: In this release, the `TitleBar` only shows the Icon and Title due to an issue where some elements don't show up on load. To work around this, use the following code to load the other elements (Subtitle, Header, Content, and Footer):

C#

```
public MainWindow()
  {
      this.InitializeComponent();
      this.ExtendsContentIntoTitleBar = true;
      this.SetTitleBar(MyTitleBar);

      MyTitleBar.Loaded += MyTitleBar_Loaded;
  }

  private void MyTitleBar_Loaded(object sender, RoutedEventArgs e)
  {
      // Parts get delay loaded. If you have the parts, make them visible.
      VisualStateManager.GoToState(MyTitleBar, "SubtitleTextVisible",
false);
      VisualStateManager.GoToState(MyTitleBar, "HeaderVisible", false);
      VisualStateManager.GoToState(MyTitleBar, "ContentVisible", false);
      VisualStateManager.GoToState(MyTitleBar, "FooterVisible", false);

      // Run layout so we re-calculate the drag regions.
      MyTitleBar.InvalidateMeasure();
  }
```

This issue will be fixed in the next 1.6 release.

Other notable changes

- Unsealed `ItemsWrapGrid`. This should be a backward-compatible change.
- `PipsPager` supports a new mode where it can wrap between the first and list items.
- `RatingControl` is now more customizable, by moving some hard-coded style properties to theme resources. This allows apps to override these values to better customize the appearance of RatingControl.

New APIs for 1.6-experimental1

1.6-experimental1 includes the following new APIs. These APIs are not experimental, but are not yet included in a stable release version of the WinAppSDK.

```
C#

Microsoft.UI.Xaml.Controls

    PipsPager
        WrapMode
        WrapModeProperty

    PipsPagerWrapMode
        None
        Wrap
```

Additional 1.6-experimental1 APIs

This release includes the following new and modified experimental APIs:

```
C#

Microsoft.UI.Content

    ChildContentLink
    ContentExternalOutputLink
        IsAboveContent

    ContentIsland
        Children
        Create
        FindAllForCompositor
        GetByVisual
        Offset
        RotationAngleInDegrees
```

```
    ContentSite
        Offset
        RotationAngleInDegrees

    ContentSiteView
        Offset
        RotationAngleInDegrees

    IContentLink
    IContentSiteBridge2
    ReadOnlyDesktopSiteBridge
```

C#

```
Microsoft.UI.Input

    EnteredMoveSizeEventArgs
    EnteringMoveSizeEventArgs
    ExitedMoveSizeEventArgs
    InputNonClientPointerSource
        EnteredMoveSize
        EnteringMoveSize
        ExitedMoveSize
        WindowRectChanged
        WindowRectChanging

    MoveSizeOperation
    WindowRectChangedEventArgs
    WindowRectChangingEventArgs
```

C#

```
Microsoft.UI.Windowing

    AppWindow
        DefaultTitleBarShouldMatchAppModeTheme
```

C#

```
Microsoft.UI.Xaml

    XamlRoot
        CoordinateConverter
        TryGetContentIsland
```

C#

```
Microsoft.UI.Xaml.Controls

    ScrollingViewChangingEventArgs
    ScrollView
        ViewChanging

    StackLayout
        IsVirtualizationEnabled
        IsVirtualizationEnabledProperty

    TabView
        CanTearOutTabs
        CanTearOutTabsProperty
        ExternalTornOutTabsDropped
        ExternalTornOutTabsDropping
        TabTearOutRequested
        TabTearOutWindowRequested

    TabViewExternalTornOutTabsDroppedEventArgs
    TabViewExternalTornOutTabsDroppingEventArgs
    TabViewTabTearOutRequestedEventArgs
    TabViewTabTearOutWindowRequestedEventArgs
    TitleBar
    TitleBarAutomationPeer
    TitleBarTemplateSettings
```

C#

```
Microsoft.UI.Xaml.Controls.Primitives

    ScrollPresenter
        ViewChanging
```

Other known issues

- Non-XAML applications that use `Microsoft.UI.Content.ContentIslands` and do not handle the *ContentIsland.AutomationProviderRequested* event (or return *nullptr* as the automation provider) will crash if any accessibility or UI automation tool is enabled such as Voice Access, Narrator, Accessibility Insights, Inspect.exe, etc.

Bug fixes

This release includes the following bug fixes:

- Fixed an issue where clicking in an empty area of a `ScrollViewer` would always move focus to the first focusable control in the `ScrollViewer` and scroll that

control into view. For more info, see GitHub issue #597.

- Fixed an issue where the `Window.Activated` event sometimes fired multiple times. For more info, see GitHub issue #7343.
- Fixed an issue setting the `NavigationViewItem.IsSelected` property to `true` prevents its children from showing when expanded. For more info, see GitHub issue #7930.
- Fixed an issue where `MediaPlayerElement` would not properly display captions with `None` or `DropShadow` edge effects. For more info, see GitHub issue #7981.
- Fixed an issue where the `Flyout.ShowMode` property was not used when showing the flyout. For more info, see GitHub issue #7987.
- Fixed an issue where `NumberBox` would sometimes have rounding errors. For more info, see GitHub issue #8780.
- Fixed an issue where using a library compiled against an older version of WinAppSDK can hit a trying to find a type or property. For more info, see GitHub issue #8810.
- Fixed an issue where initial keyboard focus is not set when launching a window. For more info, see GitHub issue #8816.
- Fixed an issue where `FlyoutShowMode.TransientWithDismissOnPointerMoveAway` didn't work after the first time it is shown. For more info, see GitHub issue #8896.
- Fixed an issue where some controls did not correctly template bind `Foreground` and `Background` properties. For more info, see GitHub issue #7070, #9020, #9029, #9083 and #9102.
- Fixed an issue where `ThemeResource`s used in `VisualStateManager` setters wouldn't update on theme change. This commonly affected controls in flyouts. For more info, see GitHub issue #9198.
- Fixed an issue where `WebView` would lose key focus, resulting in extra blur/focus events and other issues. For more info, see GitHub issue #9288.
- Fixed an issue where `NavigationView` can show a binding error in debug output. For more info, see GitHub issue #9384.
- Fixed an issue where SVG files defining a negative viewbox no longer rendered. For more info, see GitHub issue #9415.
- Fixed an issue where changing `ItemsView.Layout` orientation caused an item to be removed. For more info, see GitHub issue #9422.
- Fixed an issue where scrolling a `ScrollView` generated a lot of debug output. For more info, see GitHub issue #9434.
- Fixed an issue where `MapContorl.InteractiveControlsVisible` does not work properly. For more info, see GitHub issue #9486.

- Fixed an issue where `MapControl.MapElementClick` event doesn't properly fire. For more info, see GitHub issue #9487.
- Fixed an issue where x:Bind doesn't check for null before using a weak reference, which can result in a crash. For more info, see GitHub issue #9551.
- Fixed an issue where changing the `TeachingTip.Target` property doesn't correctly update its position. For more info, see GitHub issue #9553.
- Fixed an issue where dropdowns did not respond in WebView2. For more info, see GitHub issue #9566.
- Fixed a memory leak when using `GeometryGroup`. For more info, see GitHub issue #9578.
- Fixed an issue where scrolling through a very large number of items from an `ItemRepeater` in a `ScrollView` can cause blank render frames. For more info, see GitHub issue #9643.
- Fixed an issue where `SceneVisual` wasn't working.

Related topics

- Stable channel
- Preview channel
- Install tools for the Windows App SDK
- Create your first WinUI 3 project
- Use the Windows App SDK in an existing project
- Deploy apps that use the Windows App SDK

Experimental channel release notes for the Windows App SDK 1.5

Article • 06/04/2024

> **ⓘ Important**
>
> The experimental channel is **not supported** for use in production environments, and apps that use the experimental releases cannot be published to the Microsoft Store.

The experimental channel provides releases of the Windows App SDK that include experimental channel features that are in early stages of development. APIs for experimental features have the Experimental attribute. If you call an experimental API in your code, you will receive a build-time warning. All APIs in the experimental channel might have breaking changes in future releases, but experimental APIs are especially subject to change. Experimental features may be removed from the next release, or may never be released.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on experimental releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest experimental channel release:

- Latest experimental channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.5 Experimental (1.5.0-experimental2)

This is the latest release of the experimental channel. To download, retarget your WinAppSDK NuGet version to `1.5.240124002-experimental2`.

Bug fixes

This release includes the following bug fixes:

- Fixed an issue causing apps to crash on startup when using a custom `NavigationViewItem`. For more info, see GitHub issue #8814.
- Fixed a `NavigationView` issue where the ellipsis button would incorrectly generate an error. For more info, see GitHub issue #8380.
- Fixed an issue where a `SystemBackdrop` would not render properly in a multi-window app. For more info, see GitHub issue #8423.
- Fixed a duplication issue when inserting into the beginning of an `ObservableCollection`. For more info, see GitHub issue #8370.

New APIs for 1.5-experimental2

1.5-experimental2 includes the following new APIs. These APIs are not experimental, but are not yet included in a stable release version of the WinAppSDK.

```
C#

Microsoft.Graphics.DirectX

    DirectXPixelFormat
        A4B4G4R4
```

```
C#

Microsoft.UI.Xaml

    DebugSettings
        LayoutCycleDebugBreakLevel
        LayoutCycleTracingLevel

    LayoutCycleDebugBreakLevel
    LayoutCycleTracingLevel
```

```
C#

Microsoft.UI.Xaml.Automation.Peers

    SelectorBarItemAutomationPeer
```

```
C#

Microsoft.UI.Xaml.Controls

    MapControl
```

```
    MapControlMapServiceErrorOccurredEventArgs
    MapElement
    MapElementClickEventArgs
    MapElementsLayer
    MapIcon
    MapLayer
    SelectorBar
    SelectorBarItem
    SelectorBarSelectionChangedEventArgs
    WebView2
        EnsureCoreWebView2Async
        EnsureCoreWebView2Async
```

C#

```
Microsoft.Windows.Management.Deployment

    EnsureReadyOptions
    PackageDeploymentManager
        AddPackageSetAsync
        DeprovisionPackageAsync
        DeprovisionPackageByUriAsync
        DeprovisionPackageSetAsync
        EnsurePackageReadyAsync
        EnsurePackageReadyByUriAsync
        EnsurePackageSetReadyAsync
        IsPackageReady
        IsPackageReadyByUri
        IsPackageRegistrationPending
        IsPackageRegistrationPendingForUser
        ProvisionPackageAsync
        ProvisionPackageByUriAsync
        ProvisionPackageSetAsync
        RegisterPackageAsync
        RegisterPackageByUriAsync
        RegisterPackageSetAsync
        RemovePackageByUriAsync
        RepairPackageAsync
        RepairPackageByUriAsync
        RepairPackageSetAsync
        ResetPackageAsync
        ResetPackageByUriAsync
        ResetPackageSetAsync
        StagePackageAsync
        StagePackageByUriAsync
        StagePackageSetAsync

    PackageDeploymentProgress
    PackageDeploymentResult
        Error
        ErrorText

    PackageSet
```

```
        Items
        PackageUri

    PackageVolume
        FindPackageVolumeByName
        FindPackageVolumeByPath
        FindPackageVolumes
        IsRepairNeeded
        Repair

    ProvisionPackageOptions
    RegisterPackageOptions
        DependencyPackageFamilyNames

    RemovePackageOptions
        FailIfNotFound
```

Additional 1.5-experimental2 APIs

This release includes the following new and modified experimental APIs:

C#

```
Microsoft.UI.Xaml

    Application
        DispatcherShutdownMode

    DispatcherShutdownMode
    XamlIsland
        SystemBackdrop
```

C#

```
Microsoft.UI.Xaml.Hosting

    WindowsXamlManager
        IsXamlRunningOnCurrentThread
        XamlShutdownCompletedOnThread

    XamlShutdownCompletedOnThreadEventArgs
```

C#

```
Microsoft.Windows.System.Workloads

    IWorkloadHandler
    Workload
    WorkloadManager
```

```
    WorkloadProgress
    WorkloadProgressStatus
    WorkloadResult
    WorkloadsContract
    WorkloadStatus
```

Version 1.5 Experimental (1.5.0-experimental1)

This is the latest release of the experimental channel. To download, retarget your WinAppSDK NuGet version to `1.5.231202003-experimental1`.

New APIs for 1.5-experimental1

1.5-experimental1 includes the following new APIs. These APIs are not experimental, but are not yet included in a stable release version of the WinAppSDK.

C#

```
Microsoft.Windows.Widgets.Feeds.Providers
    CustomQueryParametersRequestedArgs
    CustomQueryParametersUpdateOptions
    FeedDisabledArgs
    FeedEnabledArgs
    FeedManager
    FeedProviderDisabledArgs
    FeedProviderEnabledArgs
    FeedProviderInfo
    IFeedManager
    IFeedProvider
```

Additional 1.5-experimental1 APIs

This release includes the following new and modified experimental APIs:

C#

```
Microsoft.UI.Content

    ContentIsland
        ConnectionInfo
        ConnectRemoteEndpoint
        IsRemoteEndpointConnected

    ContentIslandEnvironment
        CurrentOrientation
        DisplayScale
```

```
        NativeOrientation
        ThemeChanged

    ContentSiteEnvironment
        CurrentOrientation
        DisplayScale
        NativeOrientation
        NotifyThemeChanged

    DesktopChildSiteBridge
        AcceptRemoteEndpoint
        ConnectionInfo
        IsRemoteEndpointConnected
        RemoteEndpointConnecting
        RemoteEndpointDisconnected
        RemoteEndpointRequestedStateChanged

    EndpointConnectionEventArgs
    EndpointRequestedStateChangedEventArgs
    IContentIslandEndpointConnectionPrivate
    IContentSiteBridgeEndpointConnectionPrivate
    ProcessStarter
    SystemVisualSiteBridge
        IsClosed
```

C#

```
Microsoft.UI.Input

    FocusNavigationReason
    FocusNavigationRequest
    FocusNavigationRequestEventArgs
    FocusNavigationResult
    InputFocusController
        DepartFocus
        NavigateFocusRequested

    InputFocusNavigationHost
```

C#

```
Microsoft.UI.Xaml

    DebugSettings
        LayoutCycleDebugBreaks
        LayoutCycleTracing

    LayoutCycleDebugBreakLevel
    LayoutCycleTracingLevel
    XamlIsland
```

C#

```
Microsoft.UI.Xaml.Controls

    SelectionModel
        SelectAllFlat
```

C#

```
Microsoft.UI.Xaml.Core.Direct

    XamlPropertyIndex
        FlyoutBase_SystemBackdrop
        Popup_SystemBackdrop
```

C#

```
Microsoft.Windows.Management.Deployment

    AddPackageOptions
    AddPackageSetOptions
    DeploymentPriority
    DeploymentProcessingModel
    EnsureIsReadyOptions
    FindPackageSetOptions
    PackageDeploymentContract
    PackageDeploymentManager
    PackageDeploymentProgress
    PackageDeploymentProgressStatus
    PackageDeploymentResult
    PackageDeploymentStatus
    PackageRuntimeManager
    PackageSet
    PackageSetItem
    PackageSetItemRuntimeDisposition
    PackageSetManager
    PackageSetRuntimeDisposition
    PackageVolume
    PackageVolumeManager
    PackageVolumeStatus
    RegisterPackageOptions
    RemovePackageOptions
    StagePackageOptions
    StubPackageOption
```

Related topics

- Stable channel
- Preview channel

- Install tools for the Windows App SDK
- Create your first WinUI 3 project
- Use the Windows App SDK in an existing project
- Deploy apps that use the Windows App SDK

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Experimental channel release notes for the Windows App SDK 1.4

Article • 05/23/2024

> **ⓘ Important**
>
> The experimental channel is **not supported** for use in production environments, and apps that use the experimental releases cannot be published to the Microsoft Store.

The experimental channel provides releases of the Windows App SDK that include experimental channel features that are in early stages of development. APIs for experimental features have the Experimental attribute. If you call an experimental API in your code, you will receive a build-time warning. All APIs in the experimental channel might have breaking changes in future releases, but experimental APIs are especially subject to change. Experimental features may be removed from the next release, or may never be released.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on experimental releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest experimental channel release:

- Latest experimental channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.4 Experimental (1.4.0-experimental1)

This is the latest release of the experimental channel. To download, retarget your WinAppSDK NuGet version to `1.4.230518007-experimental1`.

Bug fixes

This release contains miscellaneous fixes, including the following:

- Fixed an issue where enabling the debug settings framerate counter in a new WinUI desktop application caused an access violation. For more information, see issue 2835 on GitHub.
- Fixed an issue where horizontal scrolling on a touchpad did not work in a WebView2 web page. For more information, see issue 7772 on GitHub.

Additional Experimental APIs

This release includes the following new and modified experimental APIs:

```
C#

Microsoft.UI

   IClosableNotifier
```

```
C#

Microsoft.UI.Composition.SystemBackdrops

   DesktopAcrylicController
       Closed
       FrameworkClosed
       Kind

   DesktopAcrylicKind
   MicaController
       Closed
       FrameworkClosed
```

```
C#

Microsoft.UI.Content

   ContentAppWindowBridge
       SettingChanged

   ContentEnvironmentSettingChangedEventArgs
   ContentExternalBackdropLink
   ContentExternalOutputLink
       ExternalOutputBorderMode

   ContentIsland
       FrameworkClosed

   ContentIslandWindow
```

```
        SettingChanged

    ContentSite
        FrameworkClosed

    ContentSiteWindow
        NotifySettingChanged
        SettingChanged

    CoreWindowTopLevelWindowBridge
        SettingChanged

    DesktopSiteBridge
        Closed
        FrameworkClosed

    IContentWindow
        SettingChanged

    SystemVisualSiteBridge
        Closed
        FrameworkClosed
```

C#

```
Microsoft.UI.Input

    InputLayoutPolicy
    InputNonClientPointerSource
        ConfigurationChanged
        GetForWindowId

    NonClientRegionConfigurationChangedEventArgs
```

C#

```
Microsoft.UI.System

    ThemeSettings
```

C#

```
Microsoft.UI.Windowing

    DisplayArea
        GetMetricsFromWindowId
```

C#

```
Microsoft.UI.Xaml

   XamlRoot
       ContentWindow
```

C#

```
Microsoft.UI.Xaml.Controls

   ItemContainer
       Child
       ChildProperty

   ItemContainerMultiSelectMode
       Extended

   ItemsView
       Animator
       AnimatorProperty

   MenuFlyoutPresenter
       SystemBackdrop
       SystemBackdropProperty

   RiverFlowLayout
       InvalidateItemsInfo
       RequestedRangeCount
       RequestedRangeStartIndex

   RiverFlowLayoutElementAnimator
```

C#

```
Microsoft.UI.Xaml.Controls.Primitives

   CommandBarFlyoutCommandBar
       SystemBackdrop
       SystemBackdropProperty
```

C#

```
Microsoft.UI.Xaml.Input

   AccessKeyManager
       EnterDisplayMode
```

Related topics

- Latest stable channel release notes for the Windows App SDK
- Latest preview channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Experimental channel release notes for the Windows App SDK 1.3

Article • 05/23/2024

> **ⓘ Important**
>
> The experimental channel is **not supported** for use in production environments, and apps that use the experimental releases cannot be published to the Microsoft Store.

The experimental channel provides releases of the Windows App SDK that include experimental channel features that are in early stages of development. APIs for experimental features have the Experimental attribute. If you call an experimental API in your code, you will receive a build-time warning. All APIs in the experimental channel might have breaking changes in future releases, but experimental APIs are especially subject to change. Experimental features may be removed from the next release, or may never be released.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on experimental releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest experimental channel release:

- Latest experimental channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.3 Experimental (1.3.0-experimental1)

This is the latest release of the experimental channel. To download, retarget your WinAppSDK NuGet version to `1.3.230202101-experimental1`.

XAML Backdrop APIs

With properties built in to the XAML Window, Mica & Background Acrylic backdrops are now easier to use in your WinUI 3 app.

See the Xaml Backdrop API spec on GitHub for more information about the **Window.SystemBackdrop** property.

Of note in this release, you're able to set the backdrop only in code-behind, as below. Setting `<Window.SystemBackdrop>` in markup results in a compile error.

Additionally, the Xaml Backdrop APIs are currently missing an 'experimental' tag as they are under active development.

C#

```
public MainWindow()
{
    this.InitializeComponent();

    this.SystemBackdrop = new MicaBackdrop();
}
```

Window.AppWindow

Replacing several lines of boilerplate code, you're now able to use AppWindow APIs directly from a **Window** through `Window.AppWindow`. See the Window.AppWindow API spec on GitHub for additional background and usage information.

New features from across WinAppSDK

- `ApplicationModel.DynamicDependency`: `PackageDependency.PackageGraphRevisionId` that replaces the deprecated MddGetGenerationId.
- Environment Manager: `EnvironmentManager.AreChangesTracked` to inform you whether changes to the environment manager are able to be tracked in your application. See the Environment Manager API spec on GitHub for more information.
- MRT Core: A new event, `Application.ResourceManagerInitializing` allows your app to provide its own implementation of the `IResourceManager` interface, and gives you access to the ResourceManager that WinUI uses to resolve resource URIs.
- With the latest experimental VSIX, you're now able to convert your app between unpackaged and packaged through the Visual Studio menu instead of in your project file.

- A new event, `DebugSettings.XamlResourceReferenceFailed` is now raised when a referenced Static/ThemeResource lookup can't be resolved. This event gives access to a trace that details where the framework searched for that key in order to better enable you to debug Static & ThemeResource lookup failures. For more information, see issues 4972, 2350, and 6073 on GitHub.

Bug fixes

- Fixed issues with touch input causing the soft keyboard to not appear on text boxes. For more information, see issue 6291 on GitHub.
- Fixed issue causing an ItemsRepeater with an IElementFactory as its ItemTemplate to throw an ArgumentException. For more info, see issue 4705 on GitHub.

Additional Experimental APIs

This release also includes several APIs that are in early development.

The list below details the APIs introduced in this experimental release that we don't plan to ship in the 1.3.0 stable release.

```
C#

**Microsoft.UI.Content**

    DesktopSiteBridge
        GetInputEnabledToRoot
        GetVisibleToRoot
        InputEnabled
```

```
C#

**Microsoft.UI.Dispatching**

    DispatcherQueue
        FrameworkShutdownStarting
```

```
C#

**Microsoft.UI.Input**

    InputLightDismissAction
        GetForIsland

    InputNonClientPointerSource
    InputPointerActivationBehavior
```

```
    InputPointerSource
        ActivationBehavior

    NonClientRegionCaptionTappedEventArgs
    NonClientRegionHoverEventArgs
    NonClientRegionKind
```

C#

```
**Microsoft.UI.Input.DragDrop**

    DragDropManager
    DragDropModifiers
    DragInfo
    DragOperation
    DragUIContentMode
    DragUIOverride
    DropOperationTargetRequestedEventArgs
    IDropOperationTarget
```

C#

```
**Microsoft.UI.Xaml.Automation.Peers**

    ItemContainerAutomationPeer
    ItemsViewAutomationPeer
```

C#

```
**Microsoft.UI.Xaml.Controls**

    AnnotatedScrollBar
    AnnotatedScrollBarLabel
    AnnotatedScrollBarScrollEventArgs
    AnnotatedScrollBarScrollEventType
    AnnotatedScrollBarScrollOffsetRequestedEventArgs
    AnnotatedScrollBarSubLabelRequestedEventArgs
    AnnotatedScrollBarValueRequestedEventArgs
    ElementFactory
        GetElement
        GetElementCore
        RecycleElement
        RecycleElementCore

    IndexBasedLayoutOrientation
    ItemContainer
    ItemContainerInteractionTrigger
    ItemContainerInvokedEventArgs
    ItemContainerMultiSelectMode
    ItemContainerUserInvokeMode
    ItemContainerUserSelectMode
```

```
    ItemsView
    ItemsViewItemInvokedEventArgs
    ItemsViewItemInvokeMode
    ItemsViewSelectionMode
    Layout
        IndexBasedLayoutOrientation

    NonVirtualizingLayout
        IndexBasedLayoutOrientationCore

    RiverFlowLayout
    RiverFlowLayoutItemsInfoRequestedEventArgs
    RiverFlowLayoutItemsJustification
    RiverFlowLayoutItemsStretch
    VirtualizingLayout
        IndexBasedLayoutOrientationCore

    VirtualizingLayoutContext
        VisibleRect
        VisibleRectCore
```

C#

```
**Microsoft.Graphics.Display**

    DisplayInformation
        AngularOffsetFromNativeOrientation
        DpiChanged
        OrientationChanged
        RawDpi
        RawPixelsPerViewPixel

    DisplayOrientation
```

C#

```
**Microsoft.UI.Xaml.Hosting**

    DesktopWindowXamlSource
        CreateSiteBridge
        SiteBridge
        SystemBackdrop
```

Related topics

- Latest stable channel release notes for the Windows App SDK
- Latest preview channel release notes for the Windows App SDK
- Install tools for the Windows App SDK

- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Experimental channel release notes for the Windows App SDK 1.2

Article • 05/23/2024

> **ⓘ Important**
>
> The experimental channel is **not supported** for use in production environments, and apps that use the experimental releases cannot be published to the Microsoft Store.

The experimental channel provides releases of the Windows App SDK that include experimental channel features that are in early stages of development. APIs for experimental features have the Experimental attribute. If you call an experimental API in your code, you will receive a build-time warning. All APIs in the experimental channel might have breaking changes in future releases, but experimental APIs are especially subject to change. Experimental features may be removed from the next release, or may never be released.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on experimental releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest experimental channel release:

- Latest experimental channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.2 Experimental (1.2.0-experimental2)

This is the latest release of the experimental channel. It supports all experimental channel features and features from Version 1.2 Preview 1 (1.2.0-preview1).

To download, retarget your WinAppSDK NuGet version to `1.2.220909.2-experimental2`.

Fixed issue

In upcoming Windows Insider Preview builds, applications using Windows App SDK would fail to launch.

Version 1.2 Experimental (1.2.0-experimental1)

This is the latest release of the experimental channel. It supports all experimental channel features.

To download, retarget your WinAppSDK NuGet version to `1.2.220727.1-experimental1`.

Input & Composition

First introduced in Windows App SDK 0.8, there are several experimental classes in the Microsoft.UI.Input.Experimental & Microsoft.UI.Composition.Experimental namespaces.

New to this release:

- InputPointerSource has a new static factory, GetforWindowId.

Content

New to this release, the experimental classes in the Microsoft.UI.Content namespace provide the building blocks of interactive content. These are low level primitives that can be assembled into content to provide the interactive experience for an end user. The content defines the structure for: rendering output with animations, processing input on different targets, providing accessibility representation, and handling host state changes.

Notable APIs:

- `ContentIsland` - brings together Output, Input, and Accessibility and provides the abstraction for interactive content. A custom visual tree can be constructed and made interactive with these APIs.
- `DesktopChildSiteBridge` - enables a `ContentIsland` to be connected into an HWND-based hierarchy.

Check out the sample on GitHub for more information.

Dispatching

DispatcherQueue now dispatches as reentrant. Previously, no more than a single DispatcherQueueHandler callback could be active on a single thread at a time. Now, if a handler starts a nested message pump, additional callbacks dispatch as reentrant. This matches Win32 behavior around window messages and nested message pumps.

Notifications

Registering app display name and icon for app notification is now supported. Check out the spec on GitHub for additional information.

WinUI 3

- Controls and styles are up to date with the WinUI 2.8 release.
- UWP is no longer supported in the experimental releases.

Other limitations and known issues

- Apps need to be rebuilt after updating to Windows App SDK 1.2-experimental1 due to a breaking change introduced in the ABI.
- Apps that reference a package that depends on WebView2 (like Microsoft.Identity.Client) fail to build. This is caused by conflicting binaries at build time. See issue 2492 on GitHub for more information.
- Using `dotnet build` with a WinAppSDK C# class library project may see a build error "Microsoft.Build.Packaging.Pri.Tasks.ExpandPriContent task could not be loaded". To resolve this issue set `<EnableMsixTooling>true</EnableMsixTooling>` in your project file.
- The default WinAppSDK templates note that the MaxVersionTested="10.0.19041.0" when it should be "10.0.22000.0". For full support of some features, notably UnlockedDEHs, update the MaxVersionTested to "10.0.22000.0" in your project file.

Related topics

- Latest stable channel release notes for the Windows App SDK
- Latest preview channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Experimental channel release notes for the Windows App SDK 1.0

Article • 05/23/2024

> **ⓘ Important**
>
> The experimental channel is **not supported** for use in production environments, and apps that use the experimental releases cannot be published to the Microsoft Store.

The experimental channel provides releases of the Windows App SDK that include experimental channel features that are in early stages of development. APIs for experimental features have the Experimental attribute. If you call an experimental API in your code, you will receive a build-time warning. All APIs in the experimental channel might have breaking changes in future releases, but experimental APIs are especially subject to change. Experimental features may be removed from the next release, or may never be released.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on experimental releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest experimental channel release:

- Latest experimental channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 1.0 Experimental (1.0.0-experimental1)

This release supports all experimental channel features.

Download

The following sections describe new and updated features, limitations, and known issues for this release.

WinUI 3

This release of WinUI 3 is focused on building towards new features for 1.0 stable and fixing bugs.

- **New features**: Support for showing a ContentDialog per window rather than per thread.
- **Bugs**: For the full list of bugs addressed in this release, see our GitHub repo.
- **Samples**: To see WinUI 3 controls and features in action, you can clone and build the WinUI 3 Gallery app from GitHub, or download the app from the Microsoft Store.

For more information or to get started developing with WinUI, see:

- WinUI
- Create your first WinUI 3 (Windows App SDK) project

Push notifications (experimental feature)

This release introduces a push notifications API that can be used by packaged desktop apps with Azure app registration-based identities. To use this feature, you must sign up for our private preview.

Important limitations:

- Push notifications are only supported in MSIX packaged apps that are running on Windows 10 version 2004 (build 19041) or later releases.
- Microsoft reserves the right to disable or revoke apps from push notifications during the private preview.
- Microsoft does not guarantee the reliability or latency of push notifications.
- During the private preview, push notification volume is limited to 1 million per month.

For more information, see Push notifications overview.

Windowing

This release includes updates to the windowing APIs. These are a set of high-level windowing APIs, centered around the AppWindow class, which allows for easy-to-use

windowing scenarios that integrates well with the Windows user experience and other apps. This is similar to, but not the same as, the UWP AppWindow.

Important limitations:

- This release of `AppWindow` is currently available only to Win32 apps (both packaged and unpackaged).
- The Windows App SDK does not currently provide methods for attaching UI framework content to an `AppWindow`; you're limited to using the `HWND` interop access methods.
- The Windowing API's will currently not work on Windows version 1809 and 1903 for AMD64.

For more information, see Manage app windows (Windows App SDK).

Deployment for unpackaged apps

This release introduces updates to the *dynamic dependencies* feature, including the *bootstrapper API*.

Important limitations:

- The dynamic dependencies feature is only supported for unpackaged apps.
- Elevated callers aren't supported.

For more information, see the following articles:

- MSIX framework packages and dynamic dependencies
- Use the Windows App SDK runtime for apps packaged with external location or unpackaged

Other limitations and known issues

- **No support for Any CPU build configuration**: The Windows App SDK is written in native code and thus does not support **Any CPU** build configurations. The WinUI 3 templates in Visual Studio only allow architecture-specific builds. When adding the Windows App SDK to an existing .NET application or component that supports **Any CPU**, you must specify the desired architecture: `x86`, `x64` or `arm64`.
- **.NET apps must target build 18362 or later**: Your TFM must be set to `net6.0-windows10.0.18362` or later, and your packaging project's `<TargetPlatformVersion>` must be set to 18362 or later. For more info, see the known issue on GitHub.
- **C# apps using 1.0 Experimental must use one of the following .NET SDKs**:

- .NET 6 SDK or later (see Download .NET and .NET 5 will reach End of Support on May 10, 2022).

Related topics

- Latest stable channel release notes for the Windows App SDK
- Latest preview channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Experimental channel release notes for the Windows App SDK 0.8

Article • 05/23/2024

> **ⓘ Important**
>
> The experimental channel is **not supported** for use in production environments, and apps that use the experimental releases cannot be published to the Microsoft Store.

The experimental channel provides releases of the Windows App SDK that include experimental channel features that are in early stages of development. APIs for experimental features have the Experimental attribute. If you call an experimental API in your code, you will receive a build-time warning. All APIs in the experimental channel might have breaking changes in future releases, but experimental APIs are especially subject to change. Experimental features may be removed from the next release, or may never be released.

Important links:

- If you'd like to upgrade an existing app from an older version of the Windows App SDK to a newer version, see Update existing projects to the latest release of the Windows App SDK.
- For documentation on experimental releases, see Install tools for preview and experimental channels of the Windows App SDK.

Latest experimental channel release:

- Latest experimental channel release notes for the Windows App SDK

Latest stable channel release:

- Latest stable channel release notes for the Windows App SDK

Version 0.8 Preview (0.8.0-preview)

This release supports all experimental channel features.

Download

The following sections describe new and updated features, limitations, and known issues for this release.

WinUI 3

This release introduces critical bug fixes (including those fixed in 0.5 servicing releases) and other changes for WinUI. For a complete list, see the Overview and release notes: WinUI 3 - Project Reunion 0.8 Preview (May 2021).

App lifecycle (experimental feature)

This release introduces new experimental features related to managing the app lifecycle of your app.

- All apps (packaged and unpackaged) can use **GetActivatedEventArgs** (although packaged apps can already use the implementation of this in the platform).
- Only unpackaged apps can use the **RegisterForXXXActivation** functions.
- Packaged desktop apps can use app lifecycle instancing.

For more information, see App instancing with the app lifecycle API and Rich activation with the app lifecycle API.

Deployment for unpackaged apps (experimental feature)

This release introduces new experimental deployment features for unpackaged apps. Unpackaged apps can now dynamically take a dependency on the Windows App SDK runtime packages so you can continue using your existing MSI or setup program for app deployment. This is available through the following features:

- Standalone installer for Windows App SDK.
- MSIX package bundle that includes dynamic dependencies functionality.

For more info, see Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged.

DWriteCore

This release adds the following features:

- Text decorations (underline and strikethrough in the text layout API)
- Vertical text layout
- Font face kerning API

- Experimental support for unpackaged apps

For more information, see DirectWrite to DWriteCore migration.

MRT Core

This release adds the following features:

- The build action for resources is now automatically set, reducing the need for manual project configuration.
- Experimental support for unpackaged apps.

For more information, see Manage resources with MRT Core.

Limitations and known issues

- **No support for Any CPU build configuration**: The Windows App SDK is written in native code and thus does not support **Any CPU** build configurations. The WinUI 3 templates in Visual Studio only allow architecture-specific builds. When adding the Windows App SDK to an existing .NET application or component that supports **Any CPU**, you must specify the desired architecture: `x86`, `x64` or `arm64`.
- **.NET apps must target build 18362 or later**: Your TFM must be set to `net6.0-windows10.0.18362` or later, and your packaging project's `<TargetPlatformVersion>` must be set to 18362 or later. For more info, see the known issue on GitHub.

Samples

The Windows App SDK samples do not work with this release.

Related topics

- Latest stable channel release notes for the Windows App SDK
- Latest preview channel release notes for the Windows App SDK
- Install tools for the Windows App SDK
- Create your first WinUI 3 (Windows App SDK) project
- Use the Windows App SDK in an existing project
- Deployment overview

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Breaking changes - Project Reunion 0.8 to Windows App SDK 1.0

Article • 04/19/2024

If you're migrating an app to Windows App SDK 1.0 from Project Reunion 0.8, the breaking changes listed here might affect you. Changes are grouped by technology area, such as input and MRT Core.

Input

API changes from 0.8 to 1.0

- No access to experimental types | API change |
- Stable Microsoft.UI.Input.PointerPoint added. It replaces **Microsoft.UI.Input.Experimental.ExpPointerPoint**.
  - PointerPoint now references Microsoft.UI.Input enums and structs instead of **Windows.UI.*** enums and structs (for example, IPointerPointTransform, PointerUpdateKind, PointerDeviceType).
  - PointerPoint is now fully agile.
  - PointerPoint static functions **GetCurrentPoint**, **GetCurrentPointTransformed**, **GetIntermediatePoints**, and **GetIntermediatePointsTransformed** removed. They are replaced by member functions on PointerPoint and PointerEventArgs.
  - New GetTransformedPoint method.
  - Removed **ContactRectRaw** from PointerPointProperties.
  - Removed **RawPosition** properties.
- Stable Microsoft.UI.Input.PointerEventArgs added. It replaces **Microsoft.UI.Input.Internal.ExpPointerEventArgs)**.
  - New GetIntermediateTransformedPoints method.
- Stable Microsoft.UI.Input.GestureRecognizer added. It replaces **Microsoft.UI.Input.Experimental.ExpGestureRecognizer**. References Microsoft.UI.Input enums, structs, and classes instead of **Windows.UI.Input.*** and **Windows.UI.Core.*** types (for example, GestureSettings, ManipulationDelta, TappedEventArgs).
- Public stable InputCursor, InputSystemCursor, and InputDesktopResourceCursor added. This replaces previous references to **Windows.UI.Core.CoreCursor** in the 0.8 API surface.
- Public stable InputPointerSource added – this is returned from the XAML SwapChainPanel class instead of the previous experimental

Microsoft.UI.Input.Experimental.ExpIndependentPointerInputObserver class.
    - References **Microsoft.UI.Input.InputCursor** instead of **Windows.UI.Core.CoreCursor**.
    - References **Microsoft.UI.Input** structs, enums, and types instead of **Windows.UI.*** types, such as **PointerEventArgs** and **InputPointerSourceDeviceKinds**).
    - **Type** is no longer explicitly closeable.
- **Microsoft.UI.Input.KeyboardInput** renamed to **Microsoft.UI.Input.InputKeyboardSource**.

Behavior changes from 0.8 to 1.0

- Underlying input system infrastructure upgraded to use an independent message queue for processing.
    - Supports low-latency off-UI-thread input such as inking.
    - Fully supports lifted interaction tracker APIs such as **Microsoft.UI.Composition.VisualInteractionSource** (and others).
    - Fully supports hover input for off-thread input delivery (this was a limitation in 0.8).
    - System input messages, such as **WM_POINTERDOWN**, are no longer visible through Win32 APIs on the UI thread as they are routed to an independent message queue inside the infrastructure.
- **PointerPoint** is now agile and can be accessed on any thread.
- **PointerPoint** objects can no longer be constructed statically from a pointer ID.
- XAML-based drag and drop operations fully support mouse, touch, and pen input (0.8 used pen-to-mouse downleveling).
- Direct use of **Windows.ApplicationModel.DataTransfer.DragDrop.Core.CoreDragOperation** will no longer work on the UI thread. XAML drag and drop must be used instead.

MRT Core

MRT Core APIs have moved from the Microsoft.ApplicationModel.Resources namespace to the Microsoft.Windows.ApplicationModel.Resources namespace.

Collaborate with us on GitHub

Windows developer feedback

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Overview and release notes: WinUI 3 - Project Reunion 0.8 Preview (May 2021)

Article • 03/13/2023

WinUI 3 is a native user experience (UX) framework for building modern Windows apps. It ships independently from the Windows operating system as a part of Project Reunion (now called the **Windows App SDK**). The 0.8 Preview release provides WinUI 3 templates in Visual Studio to help you start building apps with a WinUI 3-based user interface.

WinUI 3 - Project Reunion 0.8 Preview is a pre-release version of WinUI 3 that includes bug fixes, general improvements, and experimental features - some of which will be stabilized for the Windows App SDK 0.8 stable release in June 2021.

> **ⓘ Important**
>
> This preview release is intended for early evaluation and to gather feedback from the developer community. It should **NOT** be used for production apps.
>
> If you're looking to ship a production app using Project Reunion and WinUI 3, see Overview and release notes: WinUI 3 - Project Reunion 0.5.
>
> Please use the WinUI GitHub repo to provide feedback and log suggestions and issues.

> **ⓘ Note**
>
> **Project Reunion** is the previous code name for the Windows App SDK. This documentation still uses **Project Reunion** when referring to previous releases that used this code name.

Install WinUI 3 - Project Reunion 0.8 Preview

This version of WinUI 3 is available as part of the Project Reunion 0.8 Preview. To install, see:

Install tools for preview and experimental channels of the Windows App SDK

Since WinUI ships as a part of Project Reunion, you'll download the Project Reunion Visual Studio Extension (VSIX) to get started, which includes a set of developer tools and

components. For more on the Project Reunion package, see Deploy apps that use the Windows App SDK. The Windows App SDK VSIX includes WinUI 3 templates in Visual Studio that you'll use to build your WinUI 3 app.

> **Note**
>
> To see WinUI 3 controls and features in action, you can clone and build the **WinUI 3 Controls Gallery** from GitHub, or download it from the **Microsoft Store**.

Once you've set up your development environment, see WinUI 3 templates in Visual Studio to familiarize yourself with the available Visual Studio project and item templates.

For more information about getting started with building a WinUI 3 app, see the following articles:

- Create your first WinUI 3 project
- Build a C# .NET app with WinUI 3 and Win32 interop

Aside from the limitations and known issues, building an app using the WinUI projects is similar to building a UWP app with XAML and WinUI 2. Therefore, most of the guidance documentation for UWP apps and the **Windows.UI** WinRT namespaces in the Windows SDK is applicable.

WinUI 3 API reference documentation is available here: WinUI 3 API Reference

WebView2

To use WebView2 with this WinUI 3 release, please download the Evergreen Bootstrapper or Evergreen Standalone Installer found on this page if you don't already have the WebView2 Runtime installed.

Windows Community Toolkit

If you're using the Windows Community Toolkit, download the latest version.

Visual Studio Support

In order to take advantage of the latest tooling features added into WinUI 3 like Hot Reload, Live Visual Tree, and Live Property Explorer, you must use a preview version of Visual Studio 2019 16.10. Please note that Visual Studio preview releases are pre-release products, so you may run into bugs and limitations when using preview versions of Visual Studio to build WinUI 3 apps.

The table below shows the compatibility of Visual Studio 2019 versions with WinUI 3 - Project Reunion 0.5.

Expand table

VS Version	WinUI 3 - Project Reunion 0.5
16.8	No
16.9	Yes, but with no Hot Reload, Live Visual Tree, or Live Property Explorer
16.10 Previews	Yes, with all WinUI 3 tooling (in preview)

Updating your existing WinUI 3 app

If you created an app with an earlier preview or stable version of WinUI 3, you can update the project to use WinUI 3 - Project Reunion 0.8 Preview. For instructions, see Update existing projects to the latest release of Project Reunion.

Major changes introduced in this release

- Pivot control is now available to use in this release.
- Most of the critical bug fixes from the Project Reunion v0.5.5, v0.5.6, and v0.5.7 servicing releases are included in this release as well. For servicing fixes that didn't make it into this preview, see Known Issues.

List of bugs fixed in WinUI 3 - Project Reunion 0.8 Preview

- x:Bind does not work in a custom MenuFlyoutItem (more generally, x:Bind in ControlTemplate doesn't work if the parent namescope also uses x:Bind)
  - Thanks to @lhak for filing this issue on GitHub!
- Hide windowed popups when the top-level window (or island) moves
- StandardUICommand page in WinUI 3 Controls Gallery not showing everything
- Mouse right-click in TextBox crashes the application
  - Thanks to @Herdubreid for filing this issue on GitHub!
- Context menu displayed at the wrong location for EditableComboBox

- Changing RichTextBlock Selection with touch makes selection indicator go away
- Microsoft C++ exception: winrt::hresult_error at memory location 0x... when NavigationView is being used
  - Thanks to @LeftTwixWand for filing this issue on GitHub!
- WinUI 3 error message needs rewording: "Cannot resolve 'Windows.metadata'. Please install the Windows Software Development Kit. The Windows SDK is installed with Visual Studio."
- VSM Setter quirk for raising exception has backwards logic
  - Thanks to @HppZ for filling this issue on GitHub!
- Move focus off the WebView2 (if necessary) when browser process crashes
- ProgressBar doesn't show difference between Paused and Error option
  - Thanks to @j0shuams for filing this issue on GitHub!
- PointerReplay constantly running
- NavigationView causes crash in UWP, Reunion 0.5 Preview
  - Thanks to @kalin-todorov for filing this issue on GitHub!
- AutoSuggestBox, ComboBox, and CommandBarFlyout aren't setting ShouldConstrainToRootBounds="false" on their popup
- WinUI 3 - Project Reunion 0.5 generating C++ exceptions for a C# managed app
  - Thanks to @Noemata for filing this issue on GitHub!
- WebView2 initial tab focus lost
- WebView2 crashes when DPI changes after Close()
- Change in Appearance of AppBarButton with Flyout on CommandBar in WinUI3
  - Thanks to @eleanorleffler for filing this issue on GitHub!

Features and capabilities introduced in past WinUI 3 Previews

The following features and capabilities were introduced in past WinUI 3 preview releases and continue to be supported in WinUI 3 - Project Reunion 0.8 Preview.

> **Note**

Some of the following features will continue to be a part of WinUI 3 previews, but may not be a part of the next supported release. These features are marked as experimental and will throw a warning when used in an app. APIs that are a part of the WinUI 2.6 pre-release are also marked as experimental in this release.

- Ability to create desktop apps with WinUI, including .NET for Win32 apps
- Preview-level support for building UWP apps
- RadialGradientBrush
- TabView updates
- Dark theme updates
- Improvements and updates to WebView2
  - Support for High DPI
  - Support for window resizing and moving
  - Updated to target more recent version of Edge
  - No longer necessary to reference a WebView2-specific Nuget package
- SwapChainPanel
- MRT Core Support
  - This makes apps faster and lighter on startup and provides quicker resource lookup.
- Arm64 Support
- Drag and drop inside and outside of apps
- RenderTargetBitmap (currently only XAML content - no SwapChainPanel content)
- Custom cursor support
- Off-thread input
- Improvements to our tooling/developer experience:
  - Live Visual Tree, Hot Reload, Live Property Explorer and similar tools
  - Intellisense for WinUI 3
- Improvements required for open source migration
- Custom titlebar capabilities: new Window.ExtendsContentIntoTitleBar and Window.SetTitleBar APIs that allow developers to create custom title bars in desktop apps.
- VirtualSurfaceImageSource support
- In-app acrylic
- Multi-window support in desktop apps
- Input validation

Provide feedback and suggestions

We welcome your feedback in the WinUI GitHub repo.

Limitations and known issues

The WinUI 3 - Project Reunion 0.8 Preview is just that, a preview. Please expect bugs, limitations, and other issues.

The following items are some of the known issues with WinUI 3 - Project Reunion 0.8. If you find an issue that isn't listed below, please let us know by contributing to an existing issue or filing a new issue through the WinUI GitHub repo.

Platform and OS support

WinUI 3 - Project Reunion 0.8 Preview is compatible with PCs running the Windows 10 October 2018 Update (version 1809 - build 17763) and newer.

Developer tools

- Only C# and C++/WinRT apps are supported.
- Desktop apps support .NET 6 (and later) and C# 9, and must be packaged in an MSIX app.
- No XAML Designer support.
- New C++/CX apps are not supported, however, your existing apps will continue to function (please move to C++/WinRT as soon as possible).
- Unpackaged desktop deployment is not supported.
- When running a desktop app using F5, make sure that you are running the packaging project. Hitting F5 on the app project will run an unpackaged app, which WinUI 3 does not yet support.

Missing platform features

- Xbox support
- HoloLens support
- Windowed popups
  - More specifically, the `ShouldConstrainToRootBounds` property always acts as if it's set to `true`, regardless of the property value.
- Inking support, including:
  - InkCanvas
  - HandwritingView
  - InkPresenter

- Background acrylic
- MediaElement and MediaPlayerElement
- MapControl
- SwapChainPanel does not support transparency
- AcrylicBrush and other effects using a CompositionBackdropBrush can't sample from a SwapChainPanel or WebView2.
- Global Reveal uses fallback behavior, a solid brush
- XAML Islands is not supported in this release
- Using WinUI 3 directly in an existing non-WinUI desktop app has the following limitation: The currently available path for migrating an existing app is to add a **new** WinUI 3 project to your solution, and adjust or refactor your logic as needed.
- Application.Suspending is not called in desktop apps. See API reference documentation on the Application.Suspending Event for more details.
- The UISettings.ColorValuesChanged Event and AccessibilitySettings.HighContrastChanged Event are no longer supported in desktop apps. This may cause issues if you are using it to detect changes in Windows themes.
- Previously, to get a CompositionCapabilities instance you would call CompositionCapabilites.GetForCurrentView(). However, the capabilities returned from this call were *not* dependent on the view. To address and reflect this, we've deleted the GetForCurrentView() static in this release, so now you can create a CompositionCapabilties object directly.
- CoreWindow, ApplicationView, CoreApplicationView, CoreDispatcher and their dependencies are not supported in desktop apps (see below)

CoreWindow, ApplicationView, CoreApplicationView, and CoreDispatcher in desktop apps

New in WinUI 3 Preview 4 and standard going forward, CoreWindow, ApplicationView, CoreApplicationView CoreDispatcher, and their dependencies are not available in desktop apps. For example, the Window.Dispatcher property is always **null**, but the **Window.DispatcherQueue** property can be used as an alternative.

These APIs only work in UWP apps. In past previews they've partially worked in desktop apps as well, but since Preview 4 they've been fully disabled. These APIs are designed for the UWP case where there is only one window per thread, and one of the features of WinUI 3 is to enable multiple in the future.

There are APIs that internally depend on existence of these APIs, which consequently aren't supported in a desktop app. These APIs generally have a static `GetForCurrentView` method. For example UIViewSettings.GetForCurrentView.

For more information on affected APIs as well as workarounds and replacements for these APIs, please see Windows Runtime APIs not supported in desktop apps.

Known issues

- You may receive a build error due to mismatched versions of the .NET SDK and the winrt.runtime.dll. As a workaround, you can try the following:

  Explicitly set your .NET SDK to the correct version. To determine the correct version for your app, locate the `<TargetFramework>` tag in your project file. Using the Windows SDK build number that your app is targeting in the `<TargetFramework>` tag (such as 18362 or 19041), add the following item to your project file, then save your project:

  XML

  ```
  <ItemGroup>
      <FrameworkReference Update="Microsoft.Windows.SDK.NET.Ref"
  RuntimeFrameworkVersion="10.0.{Target Windows SDK Build Number}.16" />
      <FrameworkReference Update="Microsoft.Windows.SDK.NET.Ref"
  TargetingPackVersion="10.0.{Target Windows SDK Build Number}.16" />
  </ItemGroup>
  ```

  Note this workaround is required for .NET SDK 5.0.203 and earlier, but will not be required for .NET SDK 5.0.204 or 5.0.300.

- When using Visual Studio 2019 16.10 Preview 2, Live Visual Tree may cause a crash. To avoid this, update to the latest Visual Studio 2019 16.10 Preview.

- Window caption buttons may be misplaced when SetTitleBar is not set or null

WinUI 3 Controls Gallery

Check out the WinUI 3 Controls Gallery (previously called *XAML Controls Gallery - WinUI 3 version*) for a sample app that includes all controls and features that are a part of WinUI 3 - Project Reunion 0.8 Preview.

Example of the WinUI 3 Controls Gallery app

The WinUI 3 Controls Gallery app is available through the Microsoft Store.

You can also download the sample by cloning the GitHub repo. To do this, clone the **winui3** branch using the following command:

> **Note**
>
> There's also a **winui3preview** branch in this GitHub repo that provides a version of the WinUI 3 Controls Gallery that's using WinUI 3 - Project Reunion 0.8 Preview.

```
git clone --single-branch --branch winui3 https://github.com/microsoft/Xaml-Controls-Gallery.git
```

After cloning, ensure that you switch to the **winui3** branch in your local Git environment:

```
git checkout winui3
```

See also

- Windows App SDK
- Stable release channel for the Windows App SDK
- Windows App SDK Samples

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Overview and release notes: WinUI 3 - Project Reunion 0.5 (March 2021)

Article • 03/13/2023

WinUI 3 is a native user experience (UX) framework for building modern Windows apps. It ships independently from the Windows operating system as a part of Project Reunion (now called the **Windows App SDK**). The Project Reunion 0.5 release provides Visual Studio project templates to help you start building apps with a WinUI 3-based user interface.

WinUI 3 - Project Reunion 0.5 is the first stable, supported version of WinUI 3 that can be used to create production apps that can be published to the Microsoft Store. This release consists of the stability updates and general improvements that allow WinUI 3 to be forward-compatible and production-ready.

> **Note**
>
> **Project Reunion** is the previous code name for the Windows App SDK. This documentation still uses **Project Reunion** when referring to previous releases that used this code name.

Install WinUI 3 - Project Reunion 0.5

This new version of WinUI 3 is available as part of Project Reunion 0.5. To install, see:

Install tools for preview and experimental channels of the Windows App SDK

Now that WinUI ships as a part of Project Reunion, you'll download the Project Reunion Visual Studio Extension (VSIX) to get started, which includes a set of developer tools and components. For more on the Project Reunion package, see Deploy apps that use the Windows App SDK. The Windows App SDK VSIX includes WinUI 3 templates in Visual Studio that you'll use to build your WinUI 3 app.

> **Note**
>
> To see WinUI 3 controls and features in action, you can clone and build the WinUI 3 version of the XAML Controls Gallery from GitHub.

Once you've set up your development environment, see WinUI 3 templates in Visual Studio to familiarize yourself with the available Visual Studio Project and Item templates.

For more information about getting started with building a WinUI 3 app, see the following articles:

- Create your first WinUI 3 project
- Build a C# .NET app with WinUI 3 and Win32 interop

Aside from the limitations and known issues, building an app using the WinUI projects is similar to building a UWP app with XAML and WinUI 2. Therefore, most of the guidance documentation for UWP apps and the **Windows.UI** WinRT namespaces in the Windows SDK is applicable.

WinUI 3 API reference documentation is available here: WinUI 3 API Reference

WebView2

To use WebView2 with this WinUI 3 release, please download the Evergreen Bootstrapper or Evergreen Standalone Installer found on this page if you don't already have the WebView2 Runtime installed.

Windows Community Toolkit

If you're using the Windows Community Toolkit, download the latest version.

Visual Studio Support

In order to take advantage of the latest tooling features added into WinUI 3 like Hot Reload, Live Visual Tree, and Live Property Explorer, you must use a preview version of Visual Studio 2019 16.10. Please note that Visual Studio preview releases are pre-release products, so you may run into bugs and limitations when using preview versions of Visual Studio to build WinUI 3 apps.

The table below shows the compatibility of Visual Studio 2019 versions with WinUI 3 - Project Reunion 0.5.

Expand table

VS Version	WinUI 3 - Project Reunion 0.5
16.8	No

VS Version	WinUI 3 - Project Reunion 0.5
16.9	Yes, but with no Hot Reload, Live Visual Tree, or Live Property Explorer
16.10 Previews	Yes, with all WinUI 3 tooling (in preview)

Update your existing WinUI 3 app

If you created an app with an earlier preview or release version of WinUI 3, you can update the project to use the latest release of WinUI 3 - Project Reunion 0.5. For instructions, see Update existing projects to the latest release of the Windows App SDK.

Major changes introduced in this release

Stable features

This release provides the stability and support to make WinUI 3 suitable for production apps that can ship to the Microsoft Store. It includes support and forward compatibility for most features introduced in past previews:

- Ability to create desktop apps with WinUI, including .NET for Win32 apps
- RadialGradientBrush
- TabView updates
- Dark theme updates
- Improvements and updates to WebView2
  - Support for High DPI
  - Support for window resizing and moving
  - Updated to target more recent version of Edge
  - No longer necessary to reference a WebView2-specific Nuget package
- SwapChainPanel
- MRT Core Support
  - This makes apps faster and lighter on startup and provides quicker resource lookup.
- Arm64 Support
- Drag and drop inside and outside of apps
- RenderTargetBitmap (currently only XAML content - no SwapChainPanel content)
- Custom cursor support
- Off-thread input
- Improvements to our tooling/developer experience:
  - Live Visual Tree, Hot Reload, Live Property Explorer and similar tools

    - Intellisense for WinUI 3
- Improvements required for open source migration
- Custom titlebar capabilities: new Window.ExtendsContentIntoTitleBar and Window.SetTitleBar APIs that allow developers to create custom title bars in desktop apps.
- VirtualSurfaceImageSource support
- In-app acrylic

Preview features

As this is a stable release of WinUI 3, preview features have been removed. You can still access preview features by using the latest preview version of WinUI 3 (see WinUI 3 - Project Reunion 0.8 Preview). Please note the following key features are still in preview, and work to stabilize them is ongoing:

- UWP support
    - This means you cannot build or run a UWP app using the WinUI 3 - Project Reunion 0.5 VSIX. You'll need to use the WinUI 3 - Project Reunion Preview VSIX, and follow the rest of the instructions in Create your first WinUI 3 project. See Overview and release notes: WinUI 3 - Project Reunion 0.8 Preview for more information.
- Multi-window support in desktop apps
- Input validation

Provide feedback and suggestions

We welcome your feedback in the WinUI GitHub repo.

Limitations and known issues

The following items are some of the known issues with WinUI 3 - Project Reunion 0.5. If you find an issue that isn't listed below, please let us know by contributing to an existing issue or filing a new issue through the WinUI GitHub repo.

Platform and OS support

WinUI 3 - Project Reunion 0.5 is compatible with PCs running the Windows 10 October 2018 Update (version 1809 - build 17763) and newer.

Developer tools

- Only C# and C++/WinRT apps are supported
- Desktop apps support .NET 6 (and later) and C# 9, and must be packaged in an MSIX app
- No XAML Designer support
- New C++/CX apps are not supported, however, your existing apps will continue to function (please move to C++/WinRT as soon as possible)
- Unpackaged desktop deployment is not supported
- When running a desktop app using F5, make sure that you are running the packaging project. Hitting F5 on the app project will run an unpackaged app, which WinUI 3 does not yet support.

Missing Platform Features

- Xbox support
- HoloLens support
- Windowed popups
  - More specifically, the `ShouldConstrainToRootBounds` property always acts as if it's set to `true`, regardless of the property value.
- Inking support, including:
  - InkCanvas
  - HandwritingView
  - InkPresenter
- Background acrylic
- MediaElement and MediaPlayerElement
- MapControl
- SwapChainPanel does not support transparency
- AcrylicBrush and other effects using a CompositionBackdropBrush can’t sample from a SwapChainPanel or WebView2.
- Global Reveal uses fallback behavior, a solid brush
- XAML Islands is not supported in this release
- Using WinUI 3 directly in an existing non-WinUI desktop app has the following limitation: The currently available path for migrating an existing app is to add a

new WinUI 3 project to your solution, and adjust or refactor your logic as needed.

- Application.Suspending is not called in desktop apps. See API reference documentation on the Application.Suspending Event for more details.

- CoreWindow, ApplicationView, CoreApplicationView, CoreDispatcher and their dependencies are not supported in desktop apps (see below)

CoreWindow, ApplicationView, CoreApplicationView, and CoreDispatcher in desktop apps

New in WinUI 3 Preview 4 and standard going forward, CoreWindow, ApplicationView, CoreApplicationView CoreDispatcher, and their dependencies are not available in desktop apps. For example, the Window.Dispatcher property is always **null**, but the **Window.DispatcherQueue** property can be used as an alternative.

These APIs only work in UWP apps. In past previews they've partially worked in desktop apps as well, but since Preview 4 they've been fully disabled. These APIs are designed for the UWP case where there is only one window per thread, and one of the features of WinUI 3 is to enable multiple in the future.

There are APIs that internally depend on existence of these APIs, which consequently aren't supported in a desktop app. These APIs generally have a static `GetForCurrentView` method. For example UIViewSettings.GetForCurrentView.

For more information on affected APIs as well as workarounds and replacements for these APIs, please see WinRT API changes for desktop apps

Known issues

- The UISettings.ColorValuesChanged Event and AccessibilitySettings.HighContrastChanged Event are no longer supported in desktop apps. This may cause issues if you are using it to detect changes in Windows themes.

- Previously, to get a CompositionCapabilities instance you would call CompositionCapabilites.GetForCurrentView(). However, the capabilities returned from this call were *not* dependent on the view. To address and reflect this, we've deleted the GetForCurrentView() static in this release, so now you can create a CompositionCapabilties object directly.

- You may receive a build error due to mismatched versions of the .NET SDK and the winrt.runtime.dll. As a workaround, you can try the following:

Explicitly set your .NET SDK to the correct version. To determine the correct version for your app, locate the `<TargetFramework>` tag in your project file. Using the Windows SDK build number that your app is targeting in the `<TargetFramework>` tag (such as 18362 or 19041), add the following item to your project file, then save your project:

XML

```
<ItemGroup>
    <FrameworkReference Update="Microsoft.Windows.SDK.NET.Ref"
RuntimeFrameworkVersion="10.0.{Target Windows SDK Build Number}.16" />
    <FrameworkReference Update="Microsoft.Windows.SDK.NET.Ref"
TargetingPackVersion="10.0.{Target Windows SDK Build Number}.16" />
</ItemGroup>
```

Note this workaround is required for .NET SDK 5.0.203 and earlier, but will not be required for .NET SDK 5.0.204 or 5.0.300.

- When using Visual Studio 2019 16.10 Preview 2, Live Visual Tree may cause a crash. To avoid this, update to the latest Visual Studio 2019 16.10 Preview.

WinUI 3 Controls Gallery

Check out the WinUI 3 Controls Gallery (previously called *XAML Controls Gallery - WinUI 3 version*) for a sample app that includes all controls and features that are a part of WinUI 3 - Project Reunion 0.5.

Example of the WinUI 3 Controls Gallery app

You can download the sample by cloning the GitHub repo. To do this, clone the **winui3** branch using the following command:

```dos
git clone --single-branch --branch winui3 https://github.com/microsoft/Xaml-Controls-Gallery.git
```

After cloning, ensure that you switch to the **winui3** branch in your local Git environment:

```dos
git checkout winui3
```

See also

- Windows App SDK
- Stable release channel for the Windows App SDK
- Windows App SDK Samples

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

WinUI 3 - Project Reunion 0.5 Preview (March 2021) release notes

Article • 03/13/2023

WinUI 3 is a native user experience (UX) platform for building modern Windows apps. This preview of WinUI 3 works with both desktop/Win32 and UWP apps, and includes Visual Studio project templates to help get started building apps with a WinUI-based user interface as well as a NuGet package that contains the WinUI libraries.

WinUI 3 - Project Reunion 0.5 Preview is the first release of WinUI 3 where it is provided as a part of the Project Reunion package (now called the **Windows App SDK**). Alongside that change, this preview release contains critical bug fixes, increased stability, and a few other general improvements (see **Capabilities introduced in WinUI 3 - Project Reunion 0.5 Preview**).

> **ⓘ Important**
>
> This WinUI 3 preview is intended for early evaluation and to gather feedback from the developer community. It should **NOT** be used for production apps.
>
> Please use the **WinUI GitHub repo** to provide feedback and log suggestions and issues.

> **ⓘ Note**
>
> **Project Reunion** is the previous code name for the Windows App SDK. This documentation still uses **Project Reunion** when referring to previous releases that used this code name.

Install WinUI 3 - Project Reunion 0.5 Preview

This version of WinUI 3 is available as part of the Project Reunion 0.5 Preview.

To install, follow the directions found in Install tools for preview and experimental channels of the Windows App SDK.

In contrast to past preview versions of WinUI 3, you'll download a Project Reunion VSIX package instead of a WinUI VSIX package. The Project Reunion VSIX includes WinUI 3 templates in Visual Studio that you'll use to build your WinUI 3 app. Once you've

completed your installation, the experience of developing a WinUI 3 app should not change.

> **Note**
>
> You can also clone and build the WinUI 3 Preview version of the XAML Controls Gallery.

> **Note**
>
> To use WinUI 3 tooling such as Live Visual Tree, Hot Reload, and Live Property Explorer, you must enable WinUI 3 tooling with Visual Studio Preview Features as described in the instructions here.

Once you've set up your development environment, see WinUI 3 templates in Visual Studio to familiarize yourself with the available Visual Studio Project and Item templates.

For more information about getting started with the WinUI project templates, see the following articles:

- Create your first WinUI 3 project

Aside from the limitations and known issues, building an app using the WinUI projects is similar to building a UWP app with XAML and WinUI 2. Therefore, most of the guidance documentation for UWP apps and the **Windows.UI** WinRT namespaces in the Windows SDK is applicable.

WinUI 3 API reference documentation is available here: WinUI 3 API Reference

If you created a project using WinUI 3 Preview 4, you can upgrade your project to use Project Reunion 0.5 Preview.

WebView2

To use WebView2 with this WinUI 3 preview, please download the Evergreen Bootstrapper or Evergreen Standalone Installer found on this page if you don't already have the WebView2 Runtime installed.

Windows Community Toolkit

If you're using the Windows Community Toolkit, download the latest version.

Visual Studio Support

In order to take advantage of the latest tooling features added into WinUI 3 like Hot Reload, Live Visual Tree, and Live Property Explorer, you must use the latest **preview** version of Visual Studio with the latest WinUI 3 preview and be sure to enable WinUI tooling in Visual Studio Preview Features, as described in the instructions here. The table below shows the compatibility of future versions with WinUI 3 - Project Reunion 0.5 Preview:

Expand table

VS Version	WinUI 3 - Project Reunion 0.5 Preview
16.8 RTM	No
16.9 Previews	Yes, with tooling
16.9 RTM	Yes, but no Hot Reload, Live Visual Tree, or Live Property Explorer
16.10 Previews	Yes, with tooling

Major changes introduced in this release

- WinUI 3 now ships as a part of the Project Reunion package, which will also be the mechanism for how our future supported releases will ship.
- In-app acrylic is now supported.
- The Pivot control is no longer supported, and has been deprecated in WinUI 3. We recommend using the NavigationView control for your in-app navigation scenarios.
- WinUI 3 and Project Reunion will only be supported down-level to Windows 10 version 1809 - it requires build 17763 or later.
- Preview features are now marked as experimental.
  - A preview feature is anything that will continue to be a part of WinUI 3 previews, but will not be a part of the next WinUI 3 supported release.
  - Preview features also include any experimental APIs that are a part of the WinUI 2.6 preview.
  - When building an app that uses a preview feature, your app will throw a warning.

List of bugs fixed in WinUI 3 - Project Reunion 0.5 Preview

Below is a list of user-facing bugs that the team has fixed since Preview 3. There has also been a lot of work going on surrounding stabilization and improving our testing.

- WinUI 3 error message needs rewording: "Cannot resolve 'Windows.metadata'. Please install the Windows Software Development Kit. The Windows SDK is installed with Visual Studio."
- App does not respond to theme changes in Windows when selecting Windows Default theme until restarted
- Exception on XamlDirect.CreateInstance call
  - Thanks to @BorzillaR for filing this issue on GitHub!
- ProgressBar doesn't show difference between Paused and Error option
- StandardUICommand list view item flyout shows in wrong position.
- Crash in desktop XamlControlsGallery when trying to reorder ListView items with touch
  - Thanks to @j0shuams for filing this issue on GitHub!
- Press and hold on RichTextBlock puts flyout in wrong place
- Left/Right arrows aren't moving focus through RadioButtons
  - Thanks to @vmadurga for filing this issue on Github!
- Narrator remains silent when user hits the down/up arrow keys to select the next/previous month/year/date in DatePicker
- NavigationView light-dismiss doesn't work in WinUI 3

New features and capabilities introduced in past WinUI 3 Previews

The following features and capabilities were introduced in WinUI 3 Previews 1-4 and continue to be supported in WinUI 3 - Project Reunion 0.5 Preview.

- Ability to create desktop apps with WinUI, including .NET for Win32 apps
- RadialGradientBrush
- TabView updates

- Dark theme updates
- Improvements and updates to WebView2
  - Support for High DPI
  - Support for window resizing and moving
  - Updated to target more recent version of Edge
  - No longer necessary to reference a WebView2-specific Nuget package
- SwapChainPanel
- MRT Core Support
  - This makes apps faster and lighter on startup and provides quicker resource lookup.
- Arm64 Support
- Drag and drop inside and outside of apps
- RenderTargetBitmap (currently only XAML content - no SwapChainPanel content)
- Custom cursor support
- Off-thread input
- Improvements to our tooling/developer experience:
  - Live Visual Tree, Hot Reload, Live Property Explorer and similar tools
  - Intellisense for WinUI 3
- Improvements required for open source migration
- Custom titlebar capabilities: new Window.ExtendsContentIntoTitleBar and Window.SetTitleBar APIs that allow developers to create custom title bars in desktop apps.
- VirtualSurfaceImageSource support

Provide feedback and suggestions

We welcome your feedback in the WinUI GitHub repo.

Limitations and known issues

The WinUI 3 - Project Reunion 0.5 Preview is just that, a preview. The scenarios around desktop apps are especially new. Please expect bugs, limitations, and other issues.

The following items are some of the known issues with WinUI 3 - Project Reunion 0.5 Preview. If you find an issue that isn't listed below, please let us know by contributing to an existing issue or filing a new issue through the WinUI GitHub repo.

Platform and OS support

WinUI 3 - Project Reunion 0.5 Preview is compatible with PCs running the Windows 10 October 2018 Update (version 1809 - build 17763) and newer.

Developer tools

- Only C# and C++/WinRT apps are supported
- Desktop apps support .NET 6 (and later) and C# 9, and must be packaged in an MSIX app
- UWP apps support .NET Native and C# 7.3
- Developer tools and Intellisense may not work properly in Visual Studio.
- No XAML Designer support
- New C++/CX apps are not supported, however, your existing apps will continue to function (please move to C++/WinRT as soon as possible)
- Support for multiple windows in desktop apps is in progress, but not yet complete and stable.
  - Please file a bug on our repo if you find new issues or regressions with multi-window behavior.
- Unpackaged desktop deployment is not supported
- When running a desktop app using F5, make sure that you are running the packaging project. Hitting F5 on the app project will run an unpackaged app, which WinUI 3 does not yet support.

Missing Platform Features

- Xbox support
- HoloLens support
- Windowed popups
  - More specifically, the `ShouldConstrainToRootBounds` property always acts as if it's set to `true`, regardless of the property value.
- Inking support
- Acrylic
- MediaElement and MediaPlayerElement
- MapControl
- RenderTargetBitmap for SwapChainPanel and non-XAML content
- SwapChainPanel does not support transparency
- Global Reveal uses fallback behavior, a solid brush
- XAML Islands is not supported in this release
- 3rd party ecosystem libraries will not fully function
- IMEs do not work

- CoreWindow, ApplicationView, CoreApplicationView, CoreDispatcher and their dependencies are not supported in desktop apps (see below)

CoreWindow, ApplicationView, CoreApplicationView, and CoreDispatcher in desktop apps

New in Preview 4 and standard going forward, CoreWindow, ApplicationView, CoreApplicationView CoreDispatcher, and their dependencies are not available in desktop apps. For example the Window.Dispatcher property is always null, but the Window.DispatcherQueue property can be used as an alternative.

These APIs only work in UWP apps. In past previews they've partially worked in desktop apps as well, but in Preview4 they've been fully disabled. These APIs are designed for the UWP case where there is only one window per thread, and one of the features of WinUI3 is to enable multiple.

There are APIs that internally depend on existence of these APIs, which consequently aren't supported in a desktop app. These APIs generally have a static `GetForCurrentView` method. For example UIViewSettings.GetForCurrentView.

For more information on affected APIs as well as workarounds and replacements for these APIs, please see WinRT API changes for desktop apps

Known issues

- There's an issue that's causing UWP apps to fail to launch on Windows 10 version 1809. The team is actively working on fixing this bug for the next preview release.
- Alt+F4 does not close desktop app windows.
- The UISettings.ColorValuesChanged Event and AccessibilitySettings.HighContrastChanged Event are no longer supported in desktop apps. This may cause issues if you are using it to detect changes in Windows themes.
- This release includes some experimental APIs, which will throw build warnings when used. These have not been thoroughly tested by the team and may have unknown issues. Please file a bug on our repo if you encounter any issues.
- Previously, to get a CompositionCapabilities instance you would call CompositionCapabilites.GetForCurrentView(). However, the capabilities returned from this call were *not* dependent on the view. To address and reflect this, we've

deleted the GetForCurrentView() static in this release, so now you can create a CompositionCapabilties object directly.

- For C# UWP apps:

  The WinUI 3 framework is a set of WinRT components which can be used from C++ (using C++/WinRT) or C#. When using C#, there are two versions of .NET, depending on the app model: when using WinUI 3 in a UWP app, you're using .NET Native; when using in a desktop app, you're using .NET 6 or later (and C#/WinRT).

  When using C# for a WinUI 3 app in UWP, there are a few API namespace differences compared to C# in a WinUI 3 desktop app or a C# WinUI 2 app: some types are in a `Microsoft` namespace rather than a `System` namespace. For example, rather than the `INotifyPropertyChanged` interface being in the `System.ComponentModel` namespace, it's in the `Microsoft.UI.Xaml.Data` namespace.

  This applies to:
  - `INotifyPropertyChanged` (and related types)
  - `INotifyCollectionChanged`
  - `ICommand`

  The `System` namespace versions still exist, but cannot be used with WinUI 3. This means that `ObservableCollection` doesn't work as-is in WinUI 3 C# UWP apps. For a workaround, see the CollectionsInterop sample in the XAML Controls Gallery sample.

XAML Controls Gallery (WinUI 3 Preview branch)

See the WinUI 3 Preview branch of the XAML Controls Gallery for a sample app that includes all controls and features that are a part of the WinUI 3 - Project Reunion 0.5 Preview.

Example of the WinUI 3 Preview XAML Controls Gallery app

To download the sample, clone the **winui3preview** branch using the following command:

```
git clone --single-branch --branch winui3preview
https://github.com/microsoft/Xaml-Controls-Gallery.git
```

After cloning, ensure that you switch to the **winui3preview** branch in your local Git environment:

```
git checkout winui3preview
```

See also

- Windows App SDK
- Stable release channel for the Windows App SDK
- Windows App SDK Samples

Collaborate with us on

Windows developer feedback

GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

System requirements for Windows app development

Article • 03/16/2023

To develop apps for Windows 10 and 11, you'll need *Visual Studio*, the *Windows SDK*, and the *Windows App SDK*. Before installing these tools, make sure your development computer meets the minimum system requirements.

Install the tools for the Windows App SDK to get started.

Visual Studio

Visual Studio is a comprehensive *Integrated Development Environment (IDE)* that's used to *edit*, *debug*, *build*, and *publish* apps.

For the minimum system requirements, see:

- Visual Studio 2022 system requirements
- Visual Studio 2019 system requirements

Windows SDK

The Windows SDK provides access to all of the APIs and development features exposed by the Windows OS. The Windows SDK is required for building Windows apps as well as other types of components (such as services and drivers). The latest Windows SDK is installed with *Visual Studio 2019* and *Visual Studio 2022* by default.

For the minimum system requirements, see Windows SDK.

Windows App SDK

The Windows App SDK is a set of developer tools that represent the next evolution in the Windows app development platform. It provides a unified set of APIs and tools that can be used in a consistent way by any desktop app on Windows 11 (and it's backward-compatible for Windows 10, version 1809).

> **Note**

The Windows App SDK was previously known by the code name *Project Reunion*. Some SDK assets (such as the VSIX extension and certain NuGet packages) still use this name, but these assets will be renamed in a future release. Some documentation still uses *Project Reunion* when referring to an existing asset or a specified earlier release.

The Windows App SDK has the following minimum system requirements:

- Windows 10, version 1809 (build 17763) or later.
- Visual Studio 2022, version 17.0 or later, with the required workloads and components.
- Visual Studio 2019, version 16.9 or later, with the required workloads and components.
- Windows SDK, version 2004 (build 19041) or later (included with Visual Studio 2019 and 2022 by default).
- If you plan to build .NET apps, you'll also need .NET 6 or later (see Download .NET).

Visual Studio support for WinUI 3 tools

You can build, run, and deploy apps built with stable versions of the Windows App SDK on Visual Studio 2019 versions *16.9*, *16.10*, and *16.11 Preview*. You can also use Visual Studio 2022 *17.0 Preview 2 and later* to build apps with the Windows App SDK *v0.8.2 and later*. However, in order to take advantage of the latest WinUI 3 tooling features such as *hot reload*, *live visual tree*, and *live property explorer*, you'll need the version of Visual Studio 2019 with a stable version of the Windows App SDK as shown in the following table:

Expand table

	Visual Studio 2019 16.9	**Visual Studio 2019 16.10**	**Visual Studio 2019 16.11 Preview**	**Visual Studio 2022 17.0 Preview**
Windows App SDK 0.5	Tools unavailable	Tools available	Tools unavailable	Tools unavailable
Windows App SDK 0.8	Tools unavailable	Tools unavailable	Tools available (starting with Visual Studio 2019 16.11 Preview).	Tools available (starting with Visual Studio 2022 17.0 Preview 2). *Requires Windows App SDK v0.8.2 or later.*

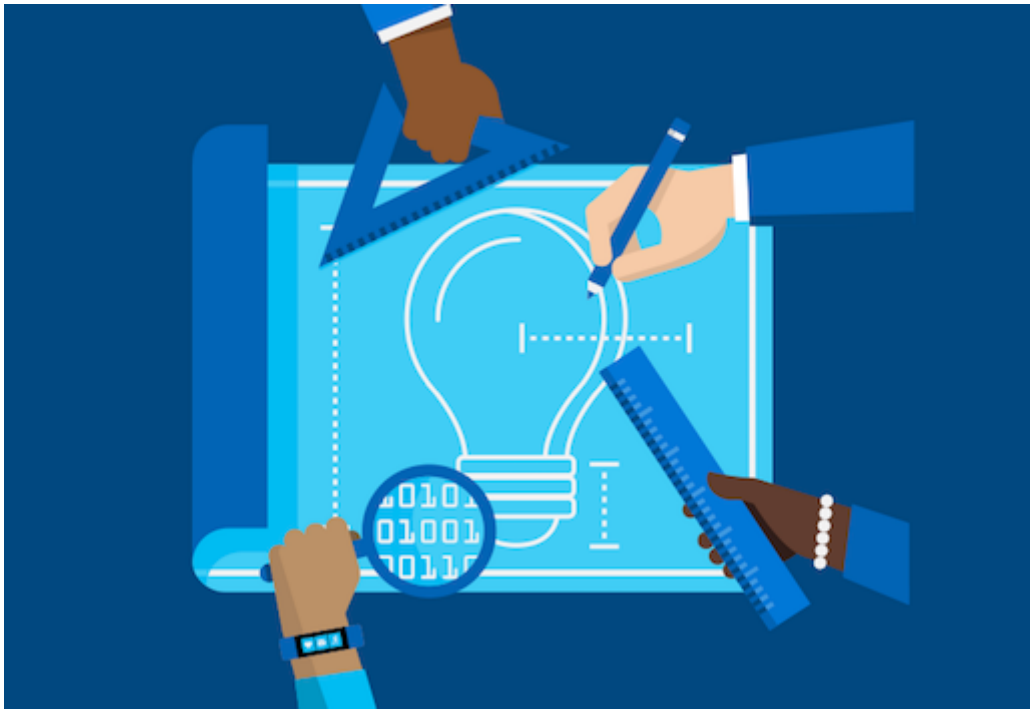
	Visual Studio 2019 16.9	Visual Studio 2019 16.10	Visual Studio 2019 16.11 Preview	Visual Studio 2022 17.0 Preview
Windows App SDK 1.0 Experimental	Tools unavailable	Tools unavailable	Tools available (starting with Visual Studio 2019 16.11 Preview 3).	Tools available (starting with Visual Studio 2022 17.0 Preview 2).
Windows App SDK 1.0 Preview 2	Tools unavailable	Tools unavailable	Tools available (starting with Visual Studio 2019 16.11 Preview 3).	Tools available (starting with Visual Studio 2022 17.0 Preview 2).

See Also

- Windows App SDK and supported Windows releases

Enable your device for development

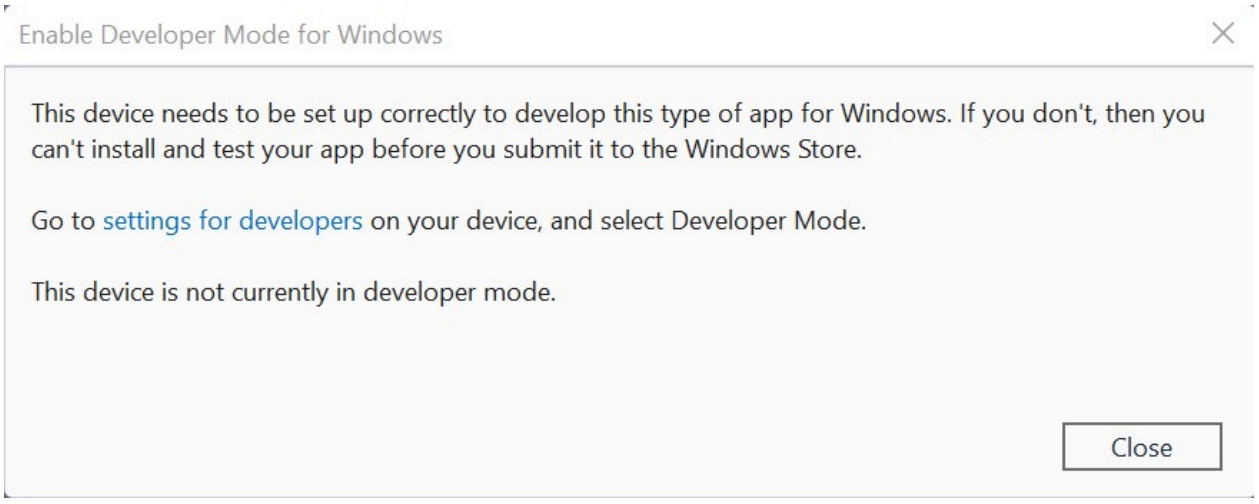
Article • 01/16/2025



If you're writing software with Visual Studio, you *will* need to enable Developer Mode on both the development PC and on any devices you'll use to test your code.

> **Note**
>
> If you're using your computer for ordinary day-to-day activities (such as gaming, web browsing, email, or Office apps), there is no need to activate Developer Mode. If you're trying to fix an issue with your computer, check out Windows help.

Opening a Windows project when Developer Mode *isn't* enabled will either open the **For developers** settings page, or cause the following dialog to appear in Visual Studio:



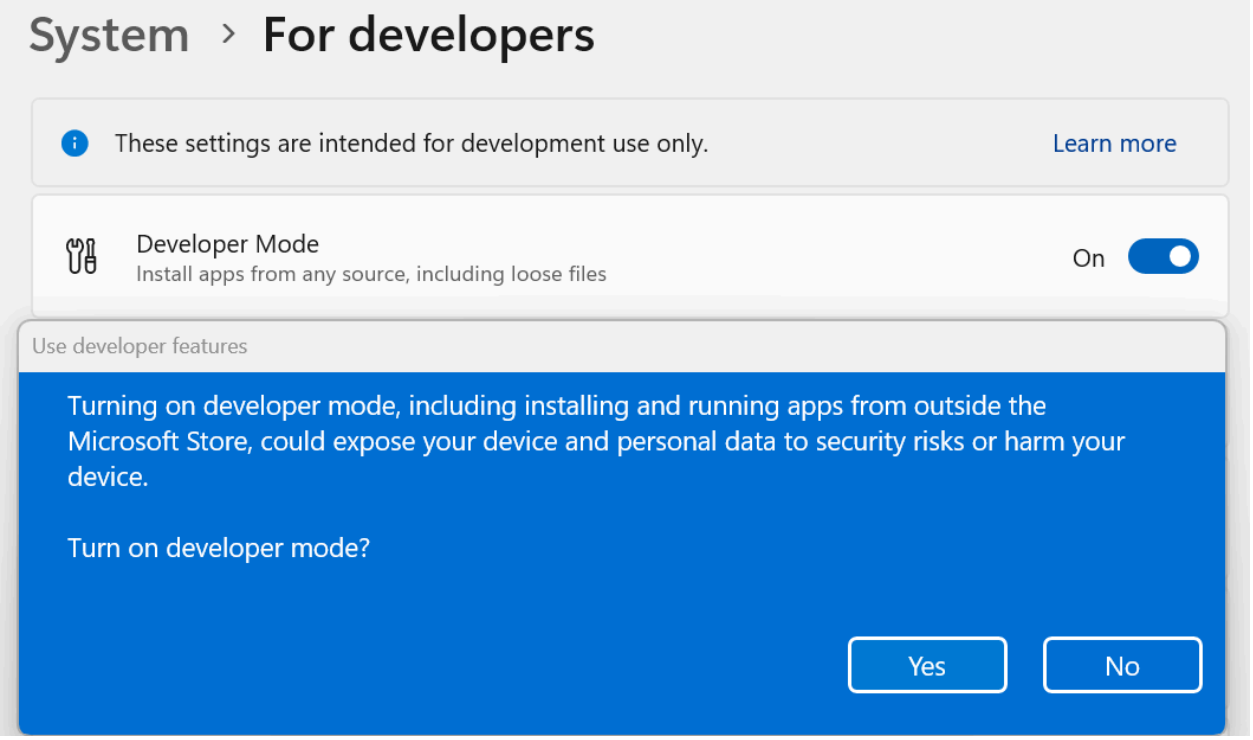
If you see this dialog, select **settings for developers** to open the **For developers** settings page.

> **Note**
>
> You can go to the **For developers** settings page at any time to *enable* or *disable* Developer Mode. Simply enter *for developers* into the search box in the taskbar.

Activate Developer Mode

To enable Developer Mode, or access other settings:

1. Open Windows Settings.
2. Search for **Developer settings** or Go to **Update & Settings** then **For developers**.
3. Toggle the Developer Mode setting, at the top of the **For developers** page
4. Read the disclaimer for the setting you choose. Click **Yes** to accept the change.



> **Note**
>
> Enabling Developer mode requires administrator access. If your device is owned by an organization, this option may be disabled.

Developer Mode features

Developer Mode replaces the requirements for a developer license. In addition to sideloading, the Developer Mode setting enables debugging and additional deployment options. This includes starting an SSH service to allow deployment to this device. In order to stop this service, you need to disable Developer Mode.

When you enable Developer Mode on desktop, a package of features is installed, including:

- **Windows Device Portal**: Device Portal is only enabled (and firewall rules are only configured for it) when the **Enable Device Portal** option is turned on.
- Installs and configures firewall rules for SSH services that allow remote installation of apps. Enabling **Device Discovery** will turn on the SSH server.

For more information on these features (or if you encounter difficulties in the installation process) check out Developer Mode features and debugging.

Developer Mode features and debugging

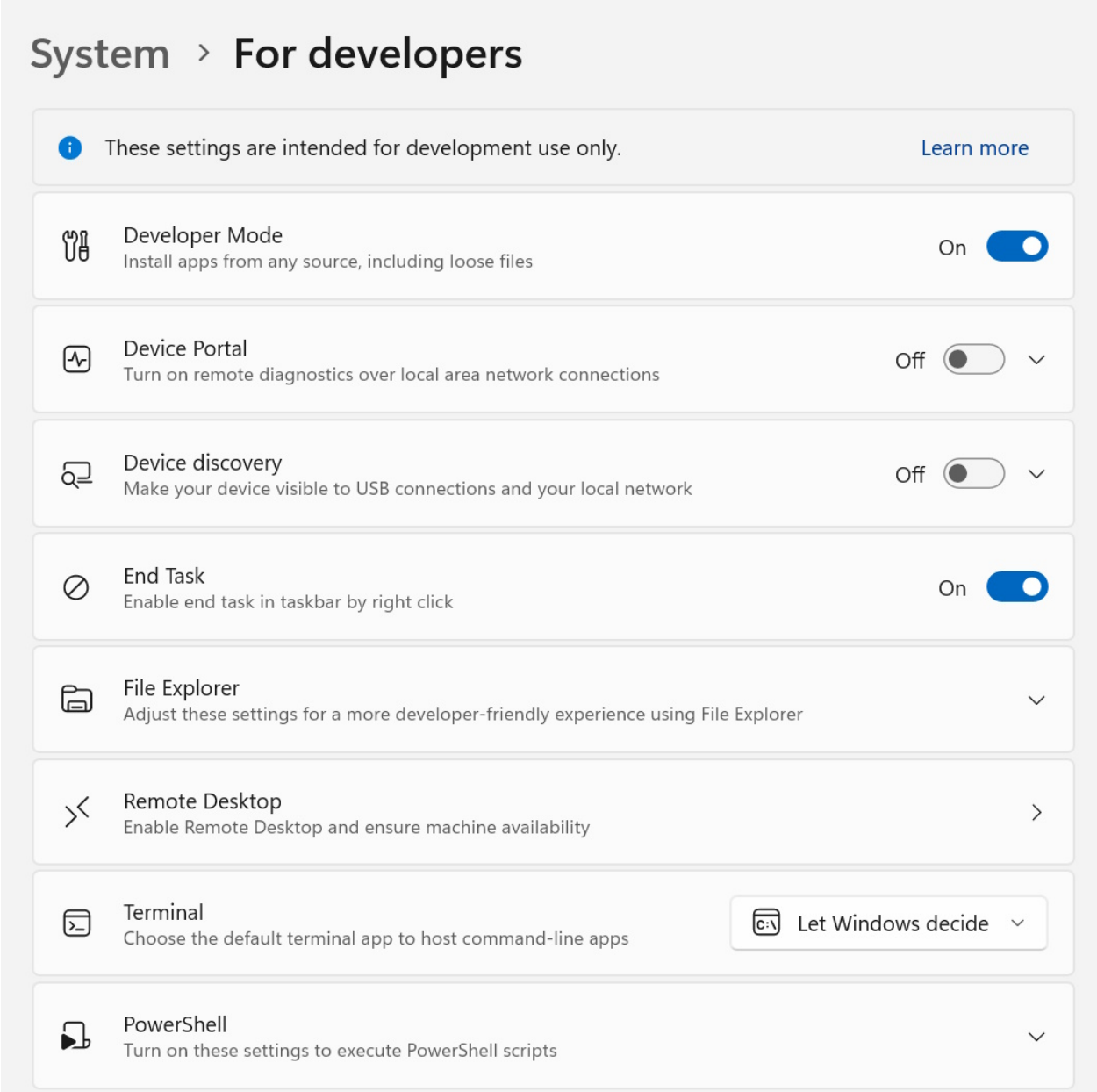
Article • 02/20/2024

If you're only interested in the basics of installing Developer Mode on your app, follow the instructions outlined in enable your device for development to get started. This article covers advanced features of Developer Mode, Developer Mode in previous versions of Windows 10, and debugging failures related to Developer Mode installations.

Additional Developer Mode features

For each device family, additional developer features may be available. These features are only available when Developer Mode is enabled on the device (and may vary depending on the version of your OS).

This image shows developer features for Windows:



Device Portal

To learn more about Device Portal, see Windows Device Portal overview.

For device-specific setup instructions, see:

- Device Portal for desktop
- Device Portal for HoloLens
- Device Portal for IoT
- Device Portal for Mobile
- Device Portal for Xbox

If you encounter problems enabling Developer Mode or Device Portal, see the Known Issues forum to find workarounds for these issues, or visit Failure to install the Developer Mode package to learn which WSUS KBs to allow in order to unblock the Developer Mode package.

SSH

SSH services are enabled when you enable Device Discovery on your device. This is used when your device is a remote deployment target for MSIX packaged applications. The names of the services are *SSH Server Broker* and *SSH Server Proxy*.

> **Note**
>
> This is not Microsoft's OpenSSH implementation, which you can find on **GitHub**.

In order to take advantage of the SSH services, you can enable Device Discovery to allow pin pairing. If you intend to run another SSH service, you can set this up on a different port or turn off the Developer Mode SSH services. To turn off the SSH services, turn off Device Discovery.

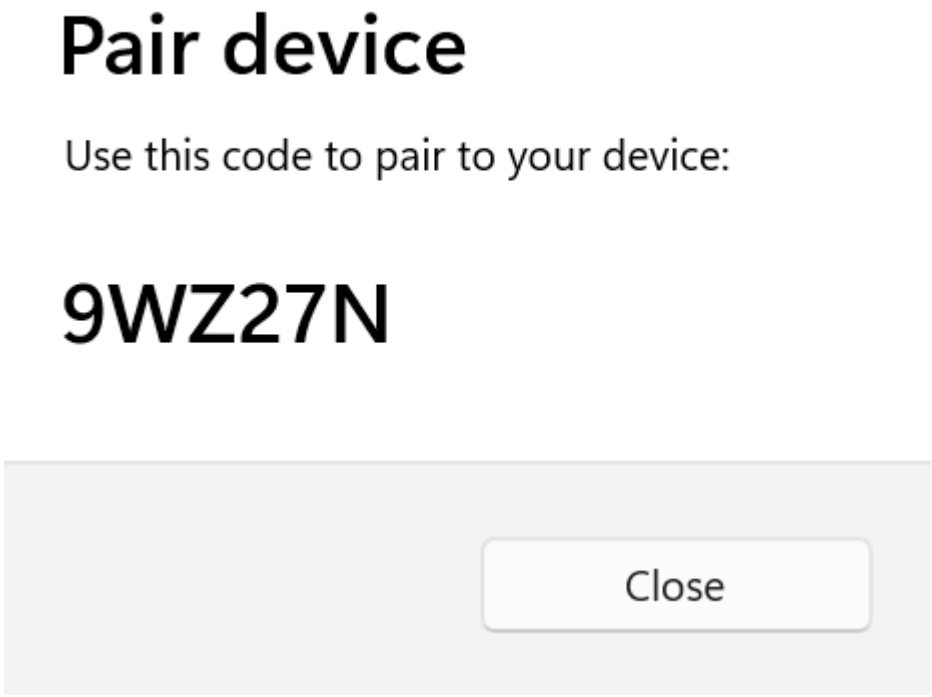
SSH login is done via the *DevToolsUser* account, which accepts a password for authentication. This password is the PIN displayed on the device after pressing the Device Discovery **Pair** button, and it's only valid while the PIN is displayed. A SFTP subsystem is also enabled for manual management of the `DevelopmentFiles` folder where loose file deployments are installed from Visual Studio.

Caveats for SSH usage

The existing SSH server used in Windows is not yet protocol compliant. Using an SFTP or SSH client may require special configuration. In particular, the SFTP subsystem runs at version 3 or less, so any connecting client should be configured to expect an old server. The SSH server on older devices uses `ssh-dss` for public key authentication (which OpenSSH has deprecated). To connect to such devices, the SSH client must be manually configured to accept `ssh-dss`.

Device Discovery

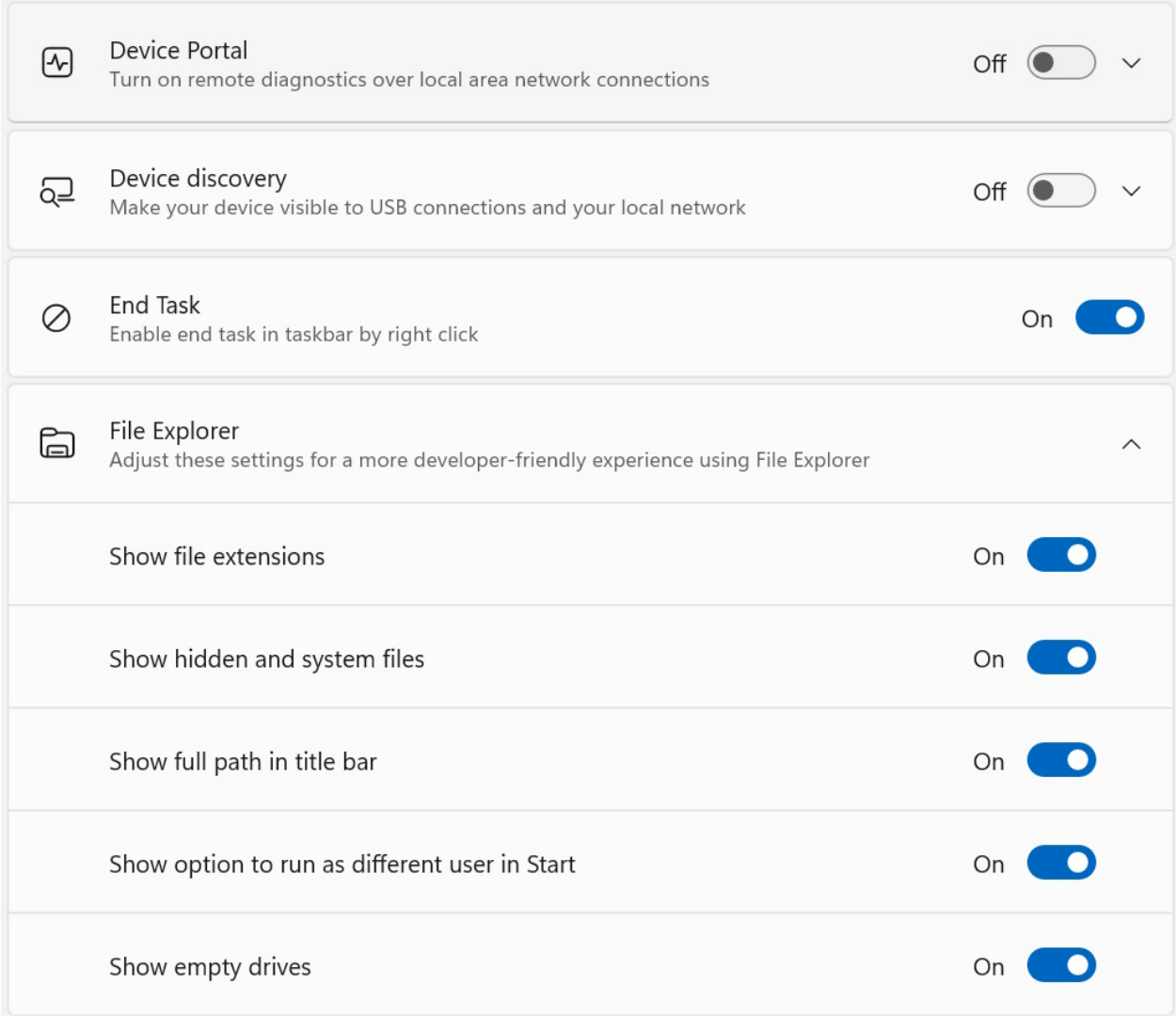
When you enable Device Discovery, you're allowing your device to be visible to other devices on the network through mDNS. This feature also allows you to get the SSH PIN for pairing to the device by pressing the **Pair** button exposed immediately after Device Discovery is enabled. This PIN prompt must be displayed on the screen in order to complete your first Visual Studio deployment targeting the device.



You should enable Device Discovery only if you intend to make the device a deployment target. For example, if you use Device Portal to deploy an app to a phone for testing, you need to enable Device Discovery on the phone, but not on your development PC.

Optimizations for Windows Explorer, Remote Desktop, and PowerShell (desktop only)

On the desktop device family, the **For developers** settings page has shortcuts to settings you can use to optimize your PC for development tasks. The sliders let you enable or disable settings easily from this single location.



Notes

In early versions of Windows 10 Mobile, a **Crash Dumps** option was present in the **Developer Settings** menu. This has been moved to Device Portal so that it can be used remotely rather than exclusively on USB.

There are several tools you can use to deploy an app from a Windows 10 PC to a Windows 10 device. Both devices must be connected to the same subnet of the network (by a wired or wireless connection) or they must be connected by USB. Both of these options only install the app package (`.appx` or `.appxbundle`). They don't install certificates.

- Use the Windows Application Deployment (`WinAppDeployCmd`) tool. Learn more about the WinAppDeployCmd tool.
- You can use Device Portal to deploy from your browser to a mobile device running Windows 10, Version 1511 or later. Use the Apps page in Device Portal to upload an app package (`.appx`) and install it on the device.

Failure to install Developer Mode package

Sometimes, due to network or administrative issues, Developer Mode won't install correctly. The Developer Mode package is required for remote deployment to this PC (using Device Portal from a browser or Device Discovery to enable SSH), but not for local development. Even if you encounter these issues, you can still deploy your app locally using Visual Studio (or from this device to another device).

See the Known Issues forum to find workarounds for these issues (and others).

> **Note**
>
> If Developer Mode doesn't install correctly, we encourage you to file a feedback request.
>
> 1. Install the **Feedback Hub app** (if you don't already have it) and open it.
> 2. Click **Add new feedback**.
> 3. Choose the **Developer Platform** category and the **Developer Mode** subcategory.
> 4. Fill out the fields (you may optionally attach a screenshot) and click **Submit**.
>
> Submitting feedback will help Microsoft resolve the issue you encountered.

Failed to locate the package

> Developer Mode package couldn’t be located in Windows Update. Error Code 0x80004005. Learn more.

This error may occur due to a network connectivity problem, Enterprise settings, or the package may be missing.

To fix this issue:

1. Ensure that your computer is connected to the internet.
2. If you're on a domain-joined computer, speak to your network administrator. The Developer Mode package (like all Features on Demand) is blocked by default in WSUS 2.1. In order to unblock the Developer Mode package in the current and previous releases, the following KBs should be allowed in WSUS:
    - 4016509

- 3180030
- 3197985

3. Check for Windows updates in **Settings** → **Updates and Security** → **Windows Updates**.

4. Verify that the Windows Developer Mode package is present in **Settings** → **System** → **Apps & Features** → **Manage optional features** → **Add a feature**. If it's missing, Windows can't find the correct package for your computer.

5. After performing the above steps, *disable* and then *re-enable* Developer Mode to verify the fix.

Failed to install the package

> Developer Mode package failed to install. Error code 0x80004005. Learn more.

This error may occur due to incompatibilities between your build of Windows and the Developer Mode package.

To fix this issue:

1. Check for Windows updates in the **Settings** → **Updates and Security** → **Windows Updates**.
2. Restart your computer to ensure all updates are applied.

Use group policies or registry keys to enable a device

For most developers, you'll want to use the settings app to enable your device for debugging. In certain scenarios (such as automated tests) you can use other ways to enable your Windows desktop device for development.

> **ⓘ Note**
>
> These steps will not enable the SSH server or allow the device to be targeted for remote deployment and debugging.

You can use `gpedit.msc` to set the group policies to enable your device, unless you have *Windows 10 Home* or *Windows 11 Home*. If you do, you'll need to use regedit or PowerShell commands to set the registry keys directly to enable your device.

Use gpedit to enable your device

1. Run `gpedit.msc`.
2. Go to **Local Computer Policy → Computer Configuration → Administrative Templates → Windows Components → App Package Deployment**.
3. Edit the following policies to enable sideloading:

   - Allow all trusted apps to install.

   OR

   Edit the following policies to enable both sideloading and Developer Mode:

   - Allow all trusted apps to install.
   - Allows development of UWP apps and installation from an *Integrated Development Environment (IDE)*.
   - Reboot your machine.

Use regedit to enable your device

1. Run `regedit`.
2. To enable sideloading, set the value of this `DWORD` to `1`:

   `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\AppModelUnlock\AllowAllTrustedApps`

   OR

   To enable developer mode, set the values of this `DWORD` to `1`:

   `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\AppModelUnlock\AllowDevelopmentWithoutDevLicense`

Use PowerShell to enable your device

1. Run PowerShell with administrator privileges.
2. To enable sideloading, run this command:

   ```
   PowerShell
   ```

```
PS C:\WINDOWS\system32> reg add
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\AppModelU
nlock" /t REG_DWORD /f /v "AllowAllTrustedApps" /d "1"
```

OR

To enable developer mode, run this command:

PowerShell

```
PS C:\WINDOWS\system32> reg add
"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\AppModelU
nlock" /t REG_DWORD /f /v "AllowDevelopmentWithoutDevLicense" /d "1"
```

Upgrade your device from Windows 8.1 to Windows 10 or 11

When you create or sideload apps on your Windows 8.1 device, you have to install a developer license. If you upgrade your device from Windows 8.1 to Windows 10 or 11, this information remains. Run the following command to remove this information from your upgraded Windows device.

> ⓘ **Note**
>
> This step is not required if you upgrade directly from Windows 8.1 to Windows 10, Version 1511 or later.

To unregister a developer license

1. Run PowerShell with administrator privileges.
2. Run this command:

PowerShell

```
unregister-windowsdeveloperlicense
```

After you unregister your license, you'll need to enable your device for development (as described in this topic) so that you can continue to develop on this device. If you don't,

you may get an error when you debug your app (or if you try to create a package for it). Here's an example of this error:

> Error : DEP0700 : Registration of the app failed.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Create a developer account

Article • 06/01/2023

Register now for a developer account so you can publish your apps to the Microsoft Store and participate in other Microsoft programs.

Opening your developer account

We offer individual and company accounts in locations around the world. Check out the overview of the sign-up process to see how it works.

Have a name for your app?

As soon as you open your developer account, you can create your app by reserving a name.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Install tools for the Windows App SDK

Article • 02/11/2025

Configure your development computer with the tools required to build Windows apps using the Windows App SDK (stable release channel) and WinUI.

Before installing any tools, see System requirements for Windows app development.

> **Important**
>
> This article applies only to the **stable release channel** of the Windows App SDK. For other release channels, see **Install tools for preview and experimental channels of the Windows App SDK**.

Install tools with winget

[Visual Studio 2022 and later] Install the required tools and workloads using the console and one of the following commands. These commands will open Visual Studio Installer with any missing workloads selected, for which you can select **Modify** to install the required workloads.

For C# developers

Visual Studio Community

```
winget install "Visual Studio Community 2022"  --override "--add
Microsoft.VisualStudio.Workload.ManagedDesktop
Microsoft.VisualStudio.ComponentGroup.WindowsAppSDK.Cs" -s msstore
```

For C++ developers

Visual Studio Community

```
winget install "Visual Studio Community 2022"  --override "--add
Microsoft.VisualStudio.Workload.NativeDesktop
Microsoft.VisualStudio.ComponentGroup.WindowsAppSDK.Cpp"  -s msstore
```

Install tools manually

The following sections describe how to install the required tools and workloads manually.

Install Visual Studio

Use the following links to install Visual Studio 2022 (recommended) or Visual Studio 2019. You can choose between the free Visual Studio Community Edition, Visual Studio Professional, or Visual Studio Enterprise. Before installing either, see System requirements for Windows app development.

> **Important**
>
> Visual Studio 2022 is recommended for developing apps using any version of the Windows App SDK (Visual Studio 2019 supports Windows App SDK 1.1 and earlier).

Download Visual Studio 2022

Download Visual Studio 2019

Required workloads and components

During Visual Studio installation, you have the option to install workloads and components (you can also open the Visual Studio Installer and select **Modify** to add workloads and components after installation). We recommend installing the following:

Visual Studio 2022 version 17.1 and later

From within the Visual Studio Installer app:

- On the **Workloads** tab:
  - For C# app development using the Windows App SDK, select **.NET Desktop Development**.
    - Then in the **Installation details** pane of the installation dialog box, select **Windows App SDK C# Templates** (at the bottom of the list).
  - For C++ app development using the Windows App SDK, select **Desktop development with C++**
    - Then in the **Installation details** pane of the installation dialog box, select **Windows App SDK C++ Templates** (at the bottom of the list).

- On the **Individual components** tab, in the **SDKs, libraries, and frameworks** section, make sure **Windows 10 SDK (10.0.19041.0)** is selected.

Visual Studio project and item templates

The Windows App SDK includes Visual Studio project and item templates for creating and developing apps that use the WinUI 3 library to implement the user interface.

Visual Studio 2022 version 17.1 and later

If you followed the instructions in Required workloads and components above, the templates should already be installed.

Select **C#** or **C++** as the language, **Windows** as the platform, and **WinUI** as the Project type to create a new Windows App SDK project.

Optionally, install Template Studio for WinUI (C#) to accelerate the creation of new .NET WinUI apps using a wizard-based UI. Select from a variety of project types and features to generate a project template customized for you.

For more versions of the Windows App SDK, see Downloads for the Windows App SDK.

Hybrid C/C++ runtime library linkage

In releases 1.0.3 and 1.1 Preview 2 and later, the Windows App SDK uses Hybrid C/C++ runtime library linkage (hybrid CRT linkage). This is a CRT linkage technique that simplifies deployment. Whether you're a C++ application developer or a C++ library developer, here are some resources for learning about hybrid CRT linkage:

- Hybrid CRT linkage coding guidelines on GitHub.
- The hybrid CRT linkage segment of the WinUI community call from April 20, 2022.

Next steps

To create your first WinUI 3 app that uses the Windows App SDK, see Create your first WinUI 3 project.

Also see Use the Windows App SDK in an existing project.

Related topics

- Windows App SDK
- Windows App SDK release channels
- Install tools for preview and experimental channels of the Windows App SDK
- System requirements for Windows app development
- Windows App SDK and supported Windows releases
- Downloads for the Windows App SDK
- Create your first WinUI 3 project
- Use the Windows App SDK in an existing project

Latest Windows App SDK downloads

Article • 03/19/2025

This page provides download links to the *latest* releases of the Windows App SDK in the three release channels (stable, preview, and experimental). To get started quickly, download the latest x64 installer below. Or, for older releases, see Archive of Windows App SDK downloads.

Download the latest x64 stable installer (1.7.0)

For all the latest Windows App SDK downloads, refer to the tables and links below. To learn more, see Windows App SDK release channels and the release notes provided below for each version.

Stable release

Windows App SDK 1.7

Expand table

Version	Runtime downloads
1.7.0 (1.7.250310001) 03/18/2025 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Preview release

Windows App SDK 1.7 Preview

Expand table

Version	Runtime downloads
1.7 Preview1 (1.7.250208002-preview1) 02/11/2025 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Experimental release

Windows App SDK 1.7 Experimental

Expand table

Version	Runtime downloads
1.7 Experimental3 (1.7.250127003-experimental3) 02/06/2025 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.7 Experimental2 (1.7.250109001-experimental2) 01/09/2025 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.7 Experimental1 (1.7.241114004-experimental1) 11/20/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Depending on your development scenario, you might require the following:

SDK downloads

Expand table

Tool	Description
NuGet package	The **Microsoft.WindowsAppSDK** NuGet package provides access to APIs provided by the Windows App SDK. The NuGet package is included with the Visual Studio extension project templates. If you have an existing project in which you want to use the Windows App SDK, you can install the latest version of the Windows App SDK NuGet package in your project directly from Visual Studio. For setup instructions, see Use the Windows App SDK in an existing project.

Runtime downloads

Apps packaged with external location, and unpackaged apps, that use the Windows App SDK can use the standalone `.exe` runtime installer or MSIX packages to deploy the Windows App SDK package dependencies with them. For setup instructions, see Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged.

Expand table

Tool	Description
Installer	The standalone .exe installer, **WindowsAppRuntimeInstall.exe**, is available as a separate download beginning with Windows App SDK 1.0.1. It installs the Windows App SDK Runtime which includes the Framework, Main, Singleton and DDLM packages.
Redistributable	The Windows App Runtime Redistributable (**Microsoft.WindowsAppRuntime.Redist**) is a zip file that includes the installer and MSIX packages for all architectures (x64, x86, and ARM64).

Archive of Windows App SDK downloads

Article • 11/21/2024

This page provides download links to *older* releases of the Windows App SDK for the three release channels (stable, preview, and experimental). For the *latest* releases, see Latest downloads for the Windows App SDK.

Depending on your development scenario, you might require the following:

SDK downloads:

Expand table

Tool	Description
Visual Studio extension	The Windows App SDK Visual Studio extension (VSIX) provides project and item templates to get started. You can choose from three versions of the extension: stable, preview, and experimental. See Set up your development environment for more details on how to install the extension.
NuGet package	The **Microsoft.WindowsAppSDK** NuGet package provides access to APIs provided by the Windows App SDK. The NuGet package is included with the Visual Studio extension project templates. If you have an existing project in which you want to use the Windows App SDK, you can install the latest version of the Windows App SDK NuGet package in your project directly from Visual Studio. For setup instructions, see Use the Windows App SDK in an existing project.

Runtime downloads:

Apps packaged with external location, and unpackaged apps, that use the Windows App SDK can use the standalone `.exe` runtime installer or MSIX packages to deploy the Windows App SDK package dependencies with them. For setup instructions, see Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged.

Expand table

Tool	Description
Installer	The standalone .exe installer, **WindowsAppRuntimeInstall.exe**, is available as a separate download beginning with Windows App SDK 1.0.1. It installs the

Tool	Description
	Windows App SDK Runtime which includes the Framework, Main, Singleton and DDLM packages.
Redistributable	The Windows App Runtime Redistributable (**Microsoft.WindowsAppRuntime.Redist**) is a zip file that includes the installer and MSIX packages for all architectures (x64, x86, and ARM64).

To learn more, see the Windows App SDK release channels and the release notes provided below for each version.

Stable releases

Windows App SDK 1.6

Expand table

Version	Runtime downloads
1.6.6 (1.6.250228001) 03/11/2025 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.6.5 (1.6.250205002) 02/11/2025 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.6.4 (1.6.250108002) 01/15/2025 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.6.3 (1.6.241114003) 11/18/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.6.1 (1.6.240923002) 10/01/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.6.0 (1.6.240829007) 09/04/2024	Installer (x64) Installer (x86)

Version	Runtime downloads
Release notes Source	Installer (arm64) Redistributable

Windows App SDK 1.5

Expand table

Version	Runtime downloads
1.5.9 (1.5.250108004) 11/12/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.5.8 (1.5.241107002) 11/12/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.5.7 (1.5.241001000) 10/08/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.5.6 (1.5.240802000) 08/13/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.5.5 (1.5.240627000) 07/09/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.5.4 (1.5.240607001) 06/11/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.5.3 (1.5.240428000) 05/01/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.5.2 (1.5.240404000) 04/09/2024	Installer (x64) Installer (x86)

Version	Runtime downloads
Release notes Source	Installer (arm64) Redistributable
1.5.1 (1.5.240311000) 03/12/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.5.0 (1.5.240227000) 02/29/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.4

Expand table

Version	Runtime downloads
1.4.7 (1.4.240802001) 08/13/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.4.6 (1.4.240512000) 05/14/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.4.5 (1.4.240211001) 02/13/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.4.4 (1.4.231219000) 01/09/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.4.3 (1.4.231115000) 11/16/2023 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.4.2 (1.4.231008000) 10/10/2023	Installer (x64) Installer (x86)

Version	Runtime downloads
Release notes Source	Installer (arm64) Redistributable
1.4.1 (1.4.230913002) 09/19/2023 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.4.0 (1.4.230822000) 08/29/2023 Release notes	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.3

Expand table

Version	Runtime downloads
1.3.3 (1.3.230724000) 07/25/2023 Release notes	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.3.2 (1.3.230602002) 06/13/2023 Release notes	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.3.1 (1.3.230502000) 05/09/2023 Release notes	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.3.0 (1.3.230331000) 04/12/2023 Release notes	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.2

Expand table

Version	SDK downloads	Runtime downloads
1.2.5 (1.2.230313.1) 03/15/2023 Release notes	Installer (x64) Installer (x86) Installer (arm64) Redistributable	
1.2.4 (1.2.230217.4) 02/22/2023 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.2.3 (1.2.230118.102) 01/25/2023 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.2.2 (1.2.221209.1) 12/14/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.2.0 11/10/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.1

Expand table

Version	SDK downloads	Runtime downloads
1.1.5 09/14/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.1.4 08/11/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.1.3 07/20/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Version	SDK downloads	Runtime downloads
1.1.2 06/28/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.1.1 06/14/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.1.0 05/24/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Preview releases

Windows App SDK 1.6 Preview

Expand table

Version	Runtime downloads
1.6 Preview2 (1.6.240821007-preview2) 08/22/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.6 Preview1 (1.6.240807006-preview1) 08/13/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.5 Preview

Expand table

Version	Runtime downloads
1.5 Preview1 (1.5.240205001-preview1) 02/06/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.4 Preview

Expand table

Version	Runtime downloads
Version 1.4 Preview 2 (1.4.0-preview2) 08/15/2023 Release notes	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.4 Preview1 (1.4.230628000-preview1) 07/06/2023 Release notes	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.3 Preview

Expand table

Version	Runtime downloads
Version 1.3 Preview 1 (1.3.0-preview1) 03/07/2023 Release notes	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.2 Preview

Expand table

Version	SDK downloads	Runtime downloads
1.2 Preview2 10/04/2022 Release notes	No new Visual Studio extensions.	Installer (x64 Preview2) Installer (x86 Preview2) Installer (arm64 Preview2) Redistributable (Preview2)
1.2 Preview1 09/06/2022 Release notes	No new Visual Studio extensions.	Installer (x64 Preview1) Installer (x86 Preview1) Installer (arm64 Preview1) Redistributable (Preview1)

Windows App SDK 1.1 Preview

Expand table

Version	SDK downloads	Runtime downloads
1.1.0 Preview 3 05/03/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.1.0 Preview 2 04/19/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.1.0 Preview 1 03/29/2022 Release notes	No new Visual Studio extensions.	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Experimental releases

Windows App SDK 1.6 Experimental

Expand table

Version	Runtime downloads
1.6 Experimental2 (1.6.240701003-experimental2) 07/02/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.6 Experimental1 (1.6.240531000-experimental1) 06/04/2024 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.5 Experimental

Expand table

Version	Runtime downloads
1.5 Experimental2 (1.5.240124002-experimental2) 01/25/2024	Installer (x64) Installer (x86)

Version	Runtime downloads
Release notes Source	Installer (arm64) Redistributable
1.5 Experimental1 (1.5.231202003-experimental1) 12/06/2023 Release notes Source	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.4 Experimental1

Expand table

Version	Runtime downloads
Version 1.4 Experimental (1.4.0-experimental1) 05/23/2023 Release notes	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.3 Experimental1

Expand table

Version	Runtime downloads
Version 1.3 Experimental (1.3.0-experimental1) 02/06/2023 Release notes	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Windows App SDK 1.2 Experimental

Expand table

Version	SDK downloads	Runtime downloads
Version 1.2 Experimental (1.2.0-experimental2) 09/21/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Version	SDK downloads	Runtime downloads
	C++ Visual Studio 2019 extension	
Version 1.2 Experimental (1.2.0-experimental1) 08/15/2022 Release notes	No new Visual Studio extensions.	Installer (x64) Installer (x86) Installer (arm64) Redistributable

Maintenance releases

None at present.

Out-of-support stable releases

Expand table

Version	SDK downloads	Runtime downloads
Version 1.0.4 06/14/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.0.3 04/18/2022 Release notes	No new Visual Studio extensions.	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.0.2 04/05/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.0.1 03/15/2022 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Installer (x64) Installer (x86) Installer (arm64) Redistributable
1.0.0 11/16/2021 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Redistributable
0.8 06/24/2021	Visual Studio extension	Redistributable

Version	SDK downloads	Runtime downloads
Release notes		
0.5 03/29/2021		
0.1 12/11/2020		

Out-of-support preview releases

Expand table

Version	SDK downloads	Runtime downloads
Version 1.0 Preview 3 (1.0.0-preview3) 10/27/2021 Release notes	C# Visual Studio 2022 extension C++ Visual Studio 2022 extension C# Visual Studio 2019 extension C++ Visual Studio 2019 extension	Redistributable
1.0 Preview 2 10/5/2021 Release notes		
1.0 Preview 1 09/17/2021 Release notes		

Out-of-support experimental releases

Expand table

Version	SDK downloads	Runtime downloads
1.0.0 Experimental 08/09/2021 Release notes		Redistributable
0.8 Experimental 05/27/2021 Release notes	Visual Studio extension	Redistributable

Visual Studio project and item templates for Windows apps

Article • 10/24/2022

Visual Studio 2019 (and later) provides many project and item templates that help you build apps for Windows 11 and Windows 10 devices by using C# or C++. This topic describes the templates and helps you choose one for your scenario.

- Project templates include project files, code files, and other assets that are configured to build an app or a component that can be loaded and used by an app.
- Item templates are project files that contain commonly used code and XAML that can be added to a project to reduce development time. For example, you can use an item template to add a new window, page, or control to your app.

For more information about installing and configuring Visual Studio to get access to these templates, see Start developing Windows apps.

WinUI templates

WinUI is the modern native user interface (UI) platform for Windows apps across desktop (.NET and native Win32) and UWP app platforms. WinUI 3 is the latest major version of WinUI, and it transforms WinUI into a full UX framework for desktop Windows apps.

WinUI 3 is available as part of the Windows App SDK. It includes a VSIX package for Visual Studio 2019 (and later) that provides project and item templates that help you get started building apps with a WinUI-based interface.

Template Studio for WinUI (C#) is a Visual Studio 2022 extension that accelerates the creation of new .NET WinUI apps using a wizard-based UI. Select from a variety of project types and features to generate a project template customized for you.

For more information about the available WinUI project and item templates, see WinUI 3 templates in Visual Studio.

UWP templates

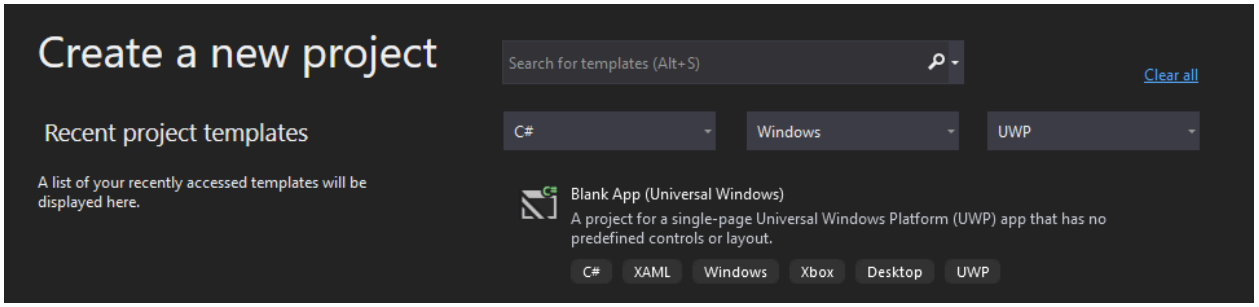
Visual Studio provides a variety of project templates for building UWP apps with C# or C++. To use these project templates, you must include the **Universal Windows Platform**

development workload when you install Visual Studio. For the C++ project templates, you must also include the **C++ (v142) Universal Windows Platforms tools** optional component for the **Universal Windows Platform development** workload.

Template Studio for UWP is a Visual Studio 2022 extension that accelerates the creation of new .NET UWP apps using a wizard-based UI. Select from a variety of project types and features to generate a project template customized for you.

Project templates for C# and UWP

To access the UWP C# project templates when you create a new project in Visual Studio, filter the language to **C#**, the platform to **Windows**, and the project type to **UWP**.



You can use these project templates to create C# UWP apps.

Expand table

Template	Description
Blank App (Universal Windows)	Creates a UWP app. The generated project includes a basic page that derives from the Windows.UI.Xaml.Controls.Page class that you can use to start building your UI.
Unit Test App (Universal Windows)	Creates a unit test project in C# for a UWP app. For more information, see Unit test C# code.

You can use these project templates to build pieces of a C# UWP app.

Expand table

Template	Description
Class Library (Universal Windows)	Creates a managed class library (DLL) in C# that can be used by other UWP apps written in managed code.
Windows Runtime Component (Universal	Creates a Windows Runtime component in C# that can be consumed by any UWP app, regardless of the programming

Template	Description
Windows)	language in which the app is written.
Optional Code Package (Universal Windows)	Creates an optional package with executable C# code that can be loaded by an app. For more information, see Optional packages with executable code.

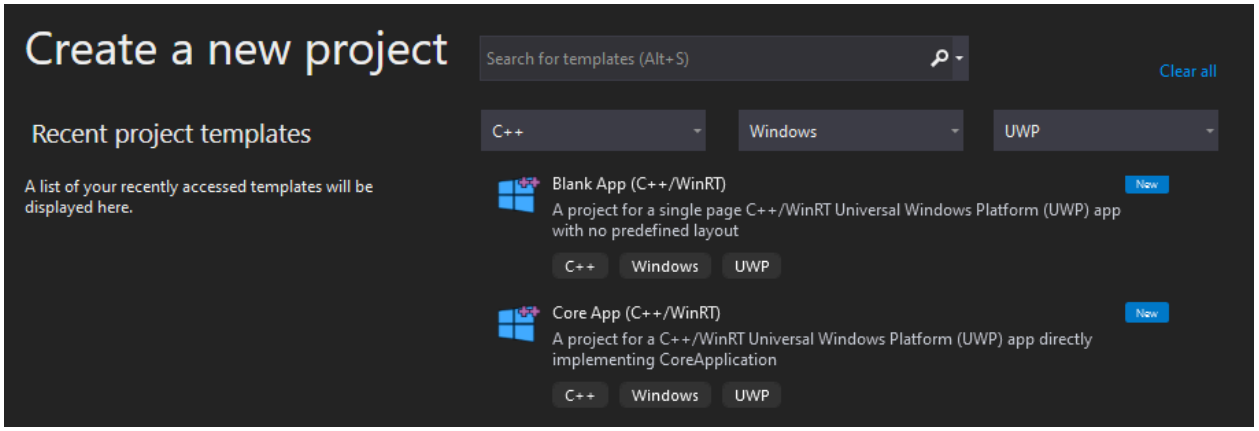
Project templates for C++ and UWP

There are two different technologies you can use to build C++ UWP apps:

- The recommended technology is C++/WinRT. This is a C++ language projection that is implemented entirely in header files, and designed to provide you with first-class access to the modern WinRT API.
- Alternatively, you can use the older C++/CX set of extensions. C++/CX is still supported, but we recommend that you use C++/WinRT instead.

To access the UWP C++ project templates when you create a new project in Visual Studio, filter the language to **C++**, the platform to **Windows**, and the project type to **UWP**.

> **Note**
>
> By default, the **Universal Windows Platform development** workload in Visual Studio only provides access to the C++/CX project templates. To access the C++/WinRT project templates, you must install the C++/WinRT VSIX package.



You can use these project templates to create C++ UWP apps.

Expand table

Template	Description
Blank App (C++/WinRT)	Creates a C++/WinRT UWP app with a XAML user interface. The generated project includes a basic page that derives from the Windows.UI.Xaml.Controls.Page class that you can use to start building your UI.
Core App (C++/WinRT)	Creates a C++/WinRT UWP app that uses CoreApplication to integrate with a variety of UI frameworks instead of a XAML user interface. For a walkthrough that demonstrates how to use this project template to create a simple game that uses DirectX, see Create a simple UWP game with DirectX.
Blank App (Universal Windows - C++/CX)	Creates a C++/WinRT UWP app with a XAML user interface. The generated project includes a basic page that derives from the Windows.UI.Xaml.Controls.Page class in the WinUI library that you can use to start building your UI.
DirectX 11 and XAML App (Universal Windows - C++/CX)	Creates a UWP app that uses DirectX 11 and a **SwapChainPanel** so you can use XAML UI controls. For more information, see DirectX game project templates.
DirectX 11 App (Universal Windows - C++/CX)	Creates a UWP app that uses DirectX 11. For more information, see DirectX game project templates.
DirectX 12 App (Universal Windows - C++/CX)	Creates a UWP app that uses DirectX 12. For more information, see DirectX game project templates.
Unit Test App (Universal Windows - C++/CX)	Creates a unit test project in C++/CX for a UWP app. For more information, see How to test a C++ UWP DLL.

You can use these project templates to build pieces of a C++ UWP app.

Expand table

Template	Description
Windows Runtime Component (C++/WinRT)	Creates a Windows Runtime component in C++/WinRT that can be consumed by any UWP app, regardless of the programming language in which the app is written.
Windows Runtime Component (Universal Windows)	Creates a Windows Runtime component in C++/CX that can be consumed by any UWP app, regardless of the programming language in which the app is written.

Template	Description
DLL (Universal Windows)	A project for creating a dynamic-link library (DLL) in C++/CX that can be used in a UWP app. For more information, see DLLs (C++/CX).
Static Library (Universal Windows)	A project for creating a static library (LIB) in C++/CX that can be used in a UWP app. For more information, see Static libraries (C++/CX).

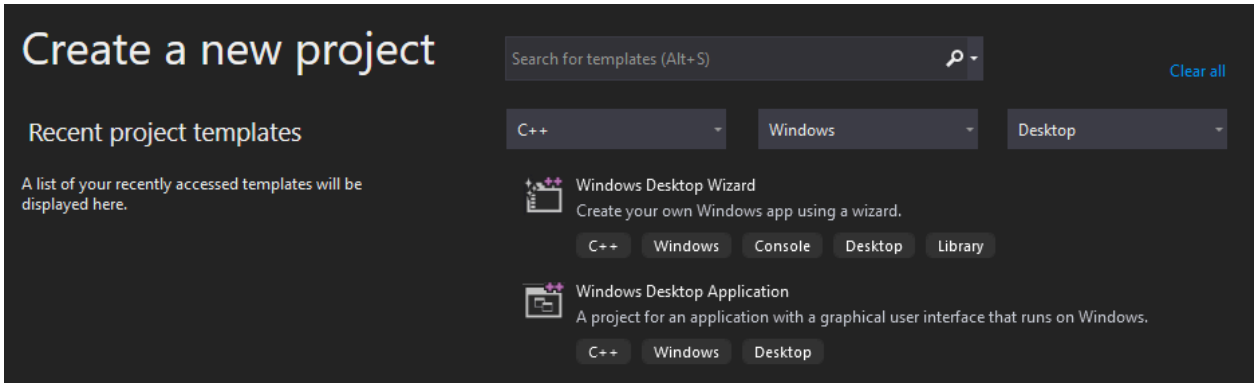
C++ desktop (Win32) templates

Visual Studio provides a variety of project templates for building desktop Windows apps with native C++ and direct access to the Win32 API. To use these project templates, you must include the **Desktop development with C++** workload when you install Visual Studio. This workload includes project templates for building desktop apps, console apps, and libraries.

The recommended technology is C++/WinRT. This is a C++ language projection that is implemented entirely in header files, and designed to provide you with first-class access to the modern WinRT API.

Project templates for C++ desktop apps

To access the C++ project templates for desktop apps when you create a new project in Visual Studio, filter the language to **C++**, the platform to **Windows**, and the project type to **Desktop**.



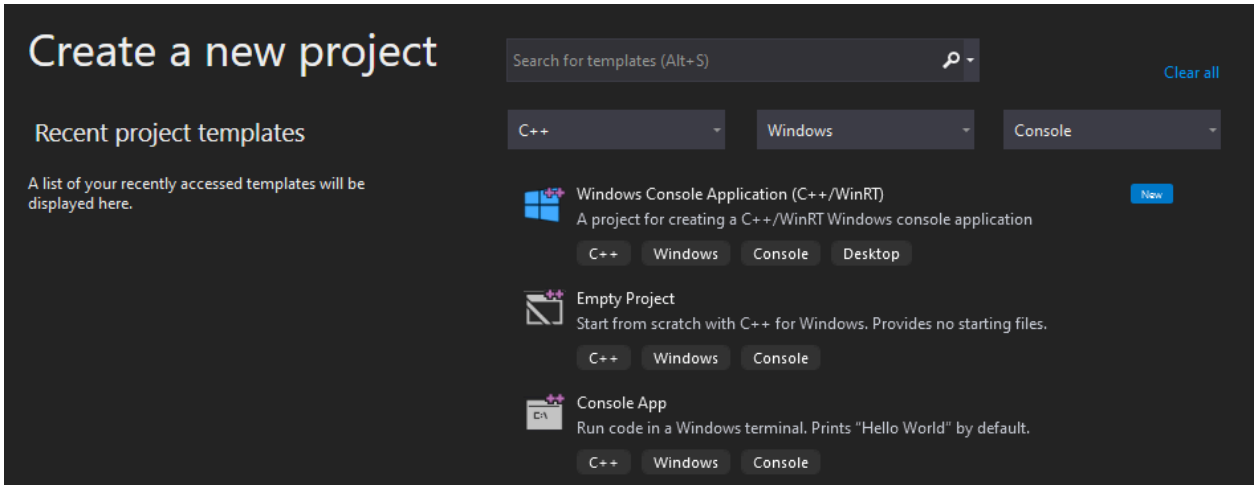
Expand table

Template	Description
Windows Desktop Application	Creates a C++/WinRT client application for Windows desktop. For more information, see Windows Desktop Application (C++/WinRT). This project

Template	Description
(C++/WinRT)	template requires the C++/WinRT VSIX.
Windows Desktop Application	Creates a classic Windows desktop app with C++. For more information, see Walkthrough: Create a traditional Windows Desktop application.
Windows Desktop Wizard	Provides a step-by-step wizard you can use to create one of the following types of projects: a classic Windows desktop app, a console app, a dynamic-link library (DLL), or a static library. For more information, see Windows Desktop Wizard and Walkthrough: Create a traditional Windows Desktop application.
Windows Application Packaging Project	Creates a project that you can use to build a desktop app into an MSIX package. This provides a modern deployment experience, the ability to integrate with Windows features via package extensions, and much more. For more information, see Windows Application Packaging Project.

Project templates for C++ console apps

To access the C++ project templates for console apps, filter the language to **C++**, the platform to **Windows**, and the project type to **Console**.

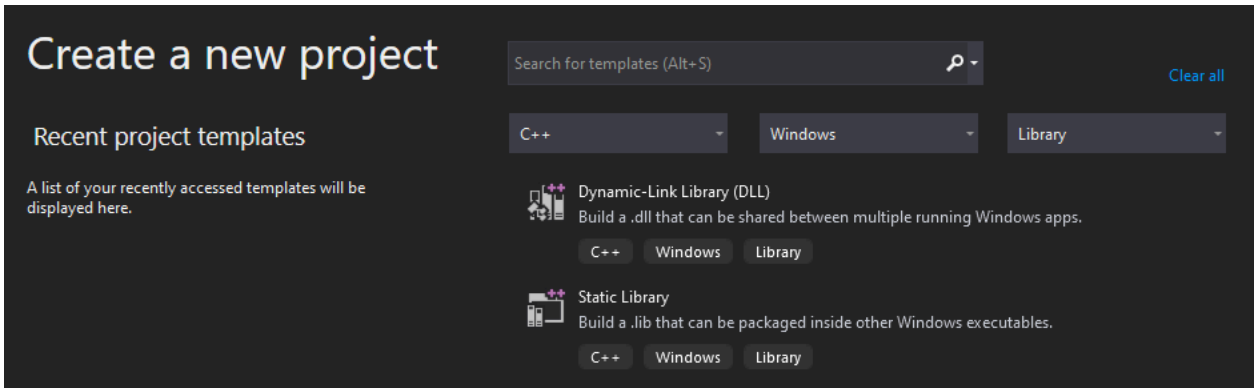


Expand table

Template	Description
Windows Console Application (C++/WinRT)	Creates a C++/WinRT console app without a user interface. For more information, see C++/WinRT quick-start. This project template requires the C++/WinRT VSIX.
Console App	Creates a console app without a user interface. For more information, see Walkthrough: Creating a Standard C++ Program.
Empty Project	An empty project for creating an application, library, or DLL. You must add any code or resources required.

Project templates for C++ libraries

To access the C++ project templates for libraries, filter the language to **C++**, the platform to **Windows**, and the project type to **Library**.



Expand table

Template	Description
Dynamic-Link Library (DLL)	A project for creating a dynamic-link library (DLL). For more information, see Walkthrough: Creating and using a dynamic link library.
Static Library	A project for creating a static library (LIB). For more information, see Walkthrough: Create and use a static library.

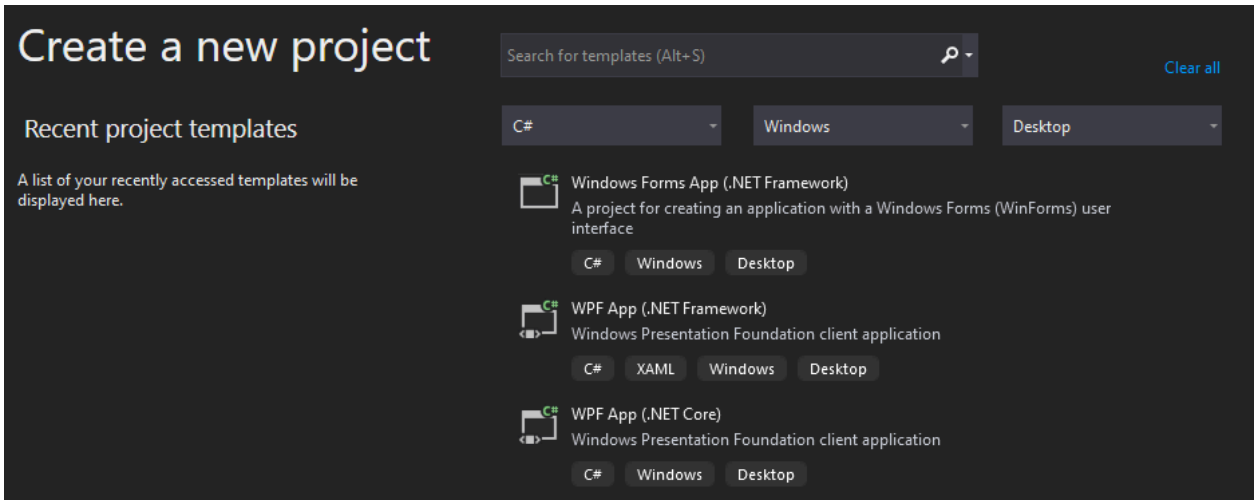
Item templates for C++ desktop apps

The C++ project templates for include many item templates that you can use to perform tasks like adding new files and resources to your project. For a comprehensive list, see Using Visual C++ Add New Item Templates.

.NET templates

Visual Studio provides a variety of project templates for building desktop Windows apps that use .NET and C#. To use these project templates, you must include the **.NET desktop development** workload when you install Visual Studio.

To access the .NET C# project templates when you create a new project in Visual Studio, filter the language to **C#**, the platform to **Windows**, and the project type to **Desktop**.



You can use these project templates to create apps using C# and .NET.

Expand table

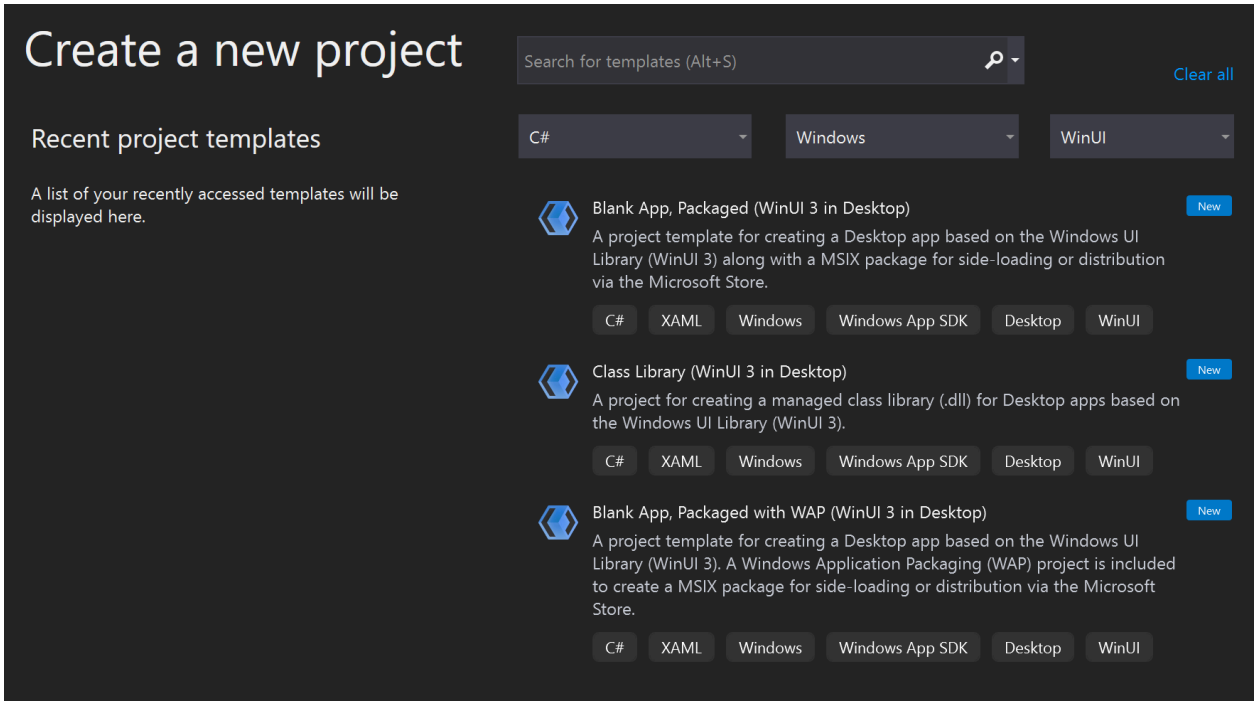
Template	Description
WPF Application	Creates a WPF app that targets .NET 6 (or later). For a walkthrough of this project template, see Create a WPF application.
WPF App (.NET Framework)	Creates a WPF app that targets the .NET Framework. For a walkthrough of this project template, see Tutorial: Create your first WPF application.
Windows Forms App	Creates a Windows Forms app that targets .NET 6 (or later).
Windows Forms App (.NET Framework)	Creates a Windows Forms app that targets the .NET Framework. For a walkthrough of this project template, see Create a Windows Forms app in Visual Studio with C#.
Windows Application Packaging Project	Creates a project that you can use to build a WPF or Windows Forms app into an MSIX package. This provides a modern deployment experience, the ability to integrate with Windows features via package extensions, and much more. For more information, see Windows Application Packaging Project.

WinUI 3 templates in Visual Studio

Article • 07/09/2024

Once you've set up your development computer (see Start developing Windows apps), you're ready to create a WinUI 3 app by starting from one of the WinUI 3 project templates in Visual Studio. This topic describes the available project and item templates.

To access the WinUI 3 project templates, in the **New Project** dialog's drop-down filters, select **C#/C++**, **Windows**, and **WinUI**, respectively. Alternatively, you can search for *WinUI*, and select one of the available C# or C++ templates.



Project templates for WinUI 3

You can use these WinUI 3 project templates to start creating an app.

Blank App, Packaged (WinUI 3 in Desktop)

This project template creates a desktop .NET (C#) or native Win32 (C++) app with a WinUI 3-based user interface. The generated project includes a basic window that derives from the **Microsoft.UI.Xaml.Window** class in the WinUI 3 library that you can use to start building your UI. For more information about using this project template, see Create your first WinUI 3 project.

The features of this project template vary between versions of the Windows App SDK extension.

- **Version 1.0 Preview 2:** Starting with this release, this project template generates an application project with the package manifest and other support needed to build the app into an MSIX package without the use of a separate packaging project. To use this project template, you must also install the single-project MSIX packaging tools extension for Visual Studio.

  > ⓘ **Note**
  >
  > In version 1.0 Preview 2 and later releases, this project template only supports a single executable in the generated MSIX package. If you need to combine multiple executables into a single MSIX package, then you'll need to use the **Blank App, Packaged with Windows Application Packaging Project (WinUI 3 in Desktop)** project template, or add a Windows Application Packaging Project to your solution.

- **Version 1.0 Preview 1 and earlier:** In these releases, this project template generates a solution with a separate Windows Application Packaging Project that is configured to build the app into an MSIX package. You can optionally install the single-project MSIX packaging tools extension for Visual Studio and combine the packaging project settings into your application project so you no longer need to maintain a separate packaging project.

Blank App, Packaged with Windows Application Packaging Project (WinUI 3 in Desktop)

This project template is available in version 1.0 Preview 1 and later releases. It creates a desktop .NET (C#) or native Win32 (C++) app with a WinUI 3-based user interface. The generated project includes a basic window that derives from the **Microsoft.UI.Xaml.Window** class in the WinUI 3 library that you can use to start building your UI. For more info about using this project template, see Create your first WinUI 3 project.

The solution also includes a separate Windows Application Packaging Project that is configured to build the app into an MSIX package. You can optionally install the single-project MSIX packaging tools extension for Visual Studio and combine the packaging project settings into your application project so you no longer need to maintain a separate packaging project.

[Experimental] Blank App (WinUI 3 in UWP)

This project template creates a C# or C++ UWP app that with a WinUI 3-based user interface. The generated project includes a basic page that derives from the **Microsoft.UI.Xaml.Controls.Page** class in the WinUI 3 library, which you can use to start building your UI. For more information about this project template, see Create your first WinUI 3 app.

WinUI project templates for other components

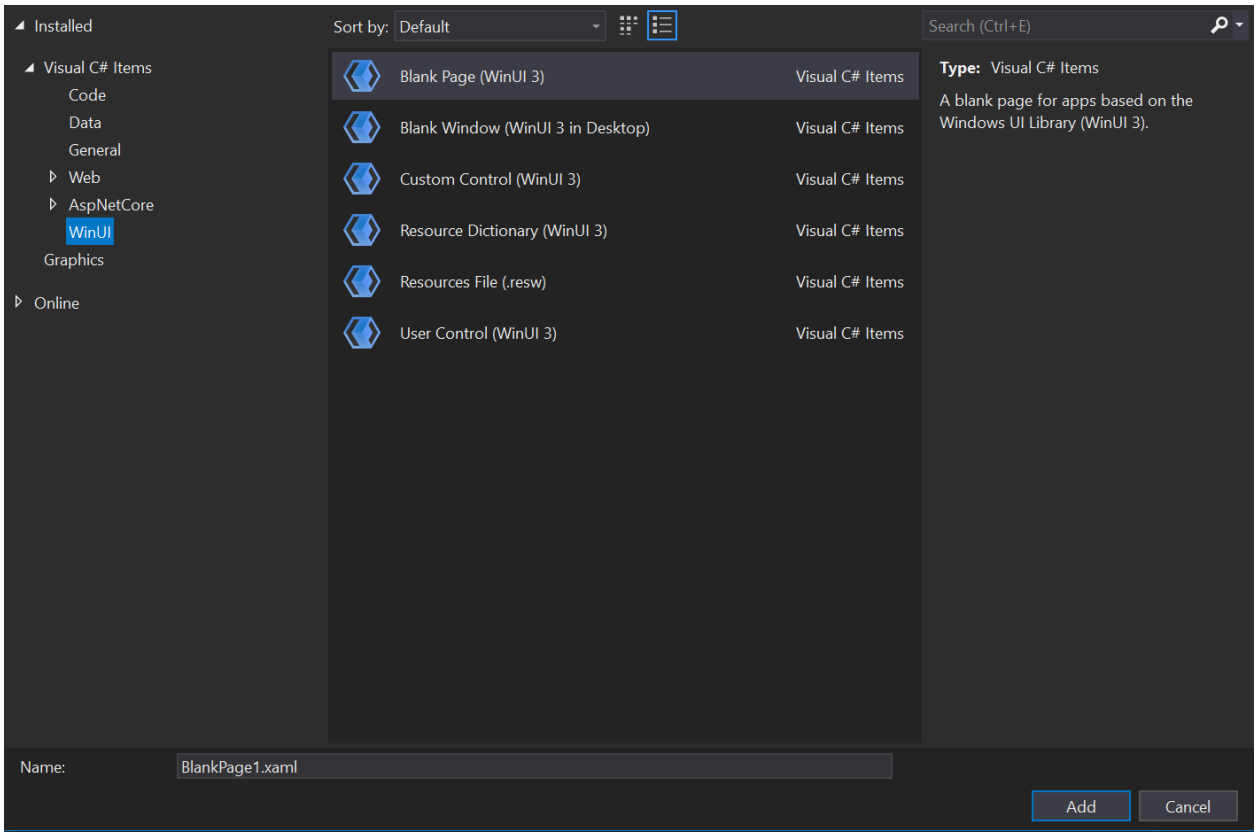
You can use these WinUI 3 project templates to build components that can be loaded and used by a WinUI 3-based app.

Expand table

Template	Language	Description
Class Library (WinUI 3 in Desktop)	C# only	Creates a .NET managed class library (DLL) in C# that can be used by other .NET desktop apps with a WinUI 3-based user interface.
[Experimental] Class Library (WinUI 3 in UWP)	C# only	Creates a managed class library (DLL) in C# that can be used by other UWP apps with a WinUI 3-based user interface.
Windows Runtime Component (WinUI 3)	C++	Creates a Windows Runtime component written in C++/WinRT that can be consumed by any UWP or desktop app with a WinUI 3-based user interface, regardless of the programming language in which the app is written.
[Experimental] Windows Runtime Component (WinUI 3 in UWP)	C#	Creates a Windows Runtime component written in C# that can be consumed by any UWP app with a WinUI 3-based user interface, regardless of the programming language in which the app is written.

Item templates for WinUI 3

The following item templates are available for use in a WinUI 3 project. To access these WinUI 3 item templates, right-click the project node in **Solution Explorer**, select **Add** -> **New item**, and click **WinUI** in the **Add New Item** dialog.



> **Note**
>
> If you have the experimental channel or an older preview release of the Windows App SDK installed, you may see a second set of Item Templates that have the [Experimental] prefix. We recommend that you use those [Experimental] item templates if you're building a non-production/preview app, and use the stable, non-marked item templates if you're building a production desktop app.

Expand table

Template	Language	Description
Blank Page (WinUI 3)	C# and C++	Adds a XAML file and code file that defines a new page derived from the **Microsoft.UI.Xaml.Controls.Page** class in the WinUI 3 library.
Blank Window (WinUI 3 in Desktop)	C# and C++	Adds a XAML file and code file that defines a new window derived from the **Microsoft.UI.Xaml.Window** class in the WinUI 3 library.
Custom Control (WinUI 3)	C# and C++	Adds a code file for creating a templated control with a default style. The templated control is derived from the **Microsoft.UI.Xaml.Controls.Control** class in the WinUI 3 library. For a walkthrough that demonstrates how to use this item template, see Templated XAML controls for UWP and WinUI 3 apps with

Template	Language	Description
		C++/WinRT and Templated XAML controls for UWP and WinUI 3 apps with C#. For more information about templated controls, see Custom XAML Controls.
Resource Dictionary (WinUI 3)	C# and C++	Adds an empty, keyed collection of XAML resources. For more information, see ResourceDictionary and XAML resource references.
Resources File (WinUI 3)	C# and C++	Adds a file for storing string and conditional resources for your app. You can use this item to help localize your app. For more info, see Localize strings in your UI and app package manifest.
User Control (WinUI 3)	C# and C++	Adds a XAML file and code file for creating a user control that derives from the **Microsoft.UI.Xaml.Controls.UserControl** class in the WinUI 3 library. Typically, a user control encapsulates related existing controls and provide its own logic. For more information about user controls, see Custom XAML Controls.

See also

- Windows App SDK
- Start developing Windows apps
- Stable release channel for the Windows App SDK
- Windows App SDK Samples
- Windows App SDK Sample Templates

Use the Windows App SDK in an existing project

Article • 01/14/2025

If you have a desktop project in which you want to use the Windows App SDK, then you can install the Windows App SDK NuGet package in your project (the latest version, or any version that you need). Unpackaged apps (that is, apps that don't use MSIX for their deployment technology) *must* follow this procedure if they're to use the Windows App SDK. But packaged apps *can* do it, too.

> **ⓘ Important**
>
> If you're working on a UWP app, then refer to Migrate from UWP to the Windows App SDK.

> **ⓘ Note**
>
> This procedure is supported in C# .NET 6 (and later) projects and C++ desktop projects. Those project types can use NuGet packages from either the stable release channel, the preview release channel, or the experimental release channel.

Prerequisites

- Visual Studio 2019 or later.
- The workloads and components for Windows app development. For more info, see Install tools for the Windows App SDK.

Instructions

1. Open an existing project in Visual Studio.

   > **ⓘ Note**
   >
   > If you have a C# desktop project, then make sure that the **TargetFramework** element in the project file is set to a Windows 10-specific moniker (such as

net6.0-windows10.0.19041.0) so that you can call Windows Runtime APIs. For more info, see Call Windows Runtime APIs in desktop apps.

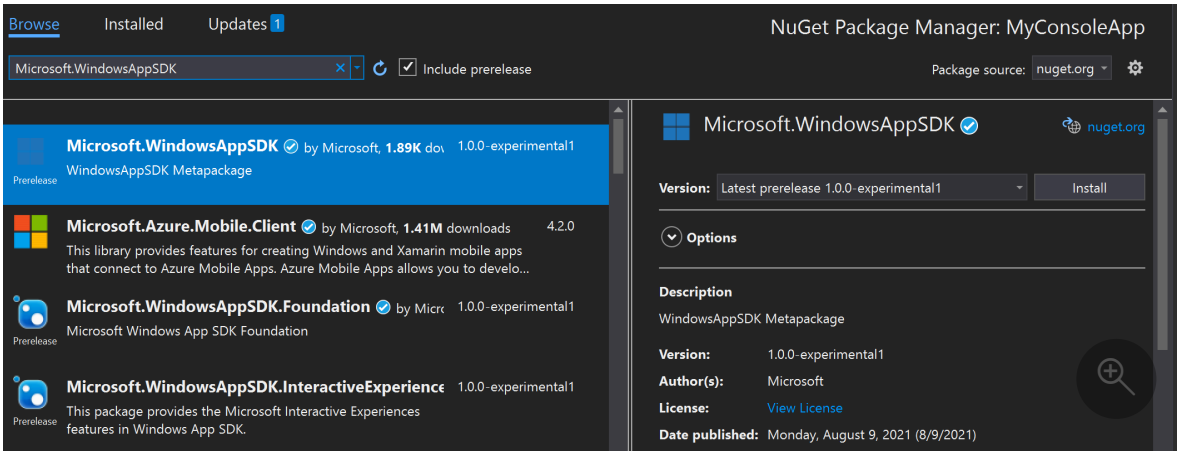
2. Make sure that package references are enabled:
   a. In Visual Studio, click **Tools** > **NuGet Package Manager** > **Package Manager Settings**.
   b. Make sure that **PackageReference** is selected for **Default package management format**.

3. Right-click your project in **Solution Explorer**, and choose **Manage NuGet Packages**.

4. In the **NuGet Package Manager** window, select the **Include prerelease** check box near the top of the window, select the **Browse** tab, and search for one of the following packages:

   - To install one of the 1.0 or later releases, search for the **Microsoft.WindowsAppSDK** package.
   - To install one of the 0.8 releases, search for the **Microsoft.ProjectReunion** package.

5. After you've found the appropriate Windows App SDK NuGet package, select the package, and click **Install** in the right-hand pane of the **NuGet Package Manager** window.



> **Note**
>
> The Windows App SDK NuGet package contains other sub-packages (including **Microsoft.WindowsAppSDK.Foundation**, **Microsoft.WindowsAppSDK.WinUI**, and others) that contain the implementations for specific components in the Windows App SDK. You can't install these sub-packages individually in order to reference only certain

components in your project. You must install the main Windows App SDK NuGet package, which includes all of the components.

6. **For unpackaged apps only**. Before your unpackaged app can use Windows App SDK APIs and components, your app must first load the Windows App SDK runtime to reference the Windows App SDK framework package. For more info, see Use the Windows App SDK runtime for apps packaged with external location or unpackaged and Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK.

7. Your app can now use Windows App SDK APIs and components that are available in the release channel that you installed. For the list of available features, see release channels.

Further info

If your existing project is a C++ project, and you want to call Windows Runtime APIs in the Windows App SDK, then you'll need to add support for C++/WinRT. See Visual Studio support for C++/WinRT, XAML, the VSIX extension, and the NuGet package. Look for info there about the **Microsoft.Windows.CppWinRT NuGet** package. Without that package, your project won't be able to find the namespace header files for Windows Runtime APIs in the Windows App SDK.

If you encounter a *Class not registered* error when you try to use a Windows App SDK component, then you might have to add to your project a dynamic dependency on the Windows App SDK Framework package. For more info, see MSIX framework packages and dynamic dependencies.

See Also

- Windows App SDK
- Release channels and release notes
- Install tools for the Windows App SDK
- Deploy apps that use the Windows App SDK
- Windows App SDK and supported Windows releases

Use the Windows App SDK in a WPF app

Article • 08/21/2024

The Windows App SDK is the next evolution in the Windows app development platform. But this topic shows how you can use Windows App SDK APIs (and Windows Runtime APIs) in a Windows Presentation Foundation (WPF) app!

- In many cases, you'll want to recreate your WPF app in the form of a WinUI 3 app. Just one of the advantages of moving to WinUI 3 is to have access to the Fluent Design System (also see Design and code Windows apps). And WinUI 3 is part of the Windows App SDK—so, naturally, a WinUI 3 app can use the other Windows App SDK features and APIs, as well. This topic doesn't cover the process of migrating your WPF app to WinUI 3.
- But if you find that you're using features of WPF that aren't yet available in WinUI 3, then you can still use Windows App SDK features (such as App Lifecycle, MRT Core, DWriteCore, and others) in your WPF app. This topic shows you how.

And in case you don't already have an existing WPF project—or you want to practice the process—this topic includes steps to create a WPF project so that you can follow along and configure it to call Windows App SDK APIs.

Prerequisites

1. Install tools for the Windows App SDK.
2. This topic covers both unpackaged and packaged WPF apps. If your WPF app is unpackaged (which WPF apps are by default), then ensure that all dependencies for unpackaged apps are installed (see Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged). A quick way to do that is to visit Latest downloads for the Windows App SDK, then download and unzip and run one of the stable release **Runtime downloads**.

> **Important**
>
> The version of the **Runtime** that you install needs to match the version of the **Microsoft.WindowsAppSDK** NuGet package that you'll install in a later step.

For more info about the terms *unpackaged* and *packaged*, see Advantages and disadvantages of packaging your app.

Create a WPF project if you don't already have one

If you already have a WPF project, then you can move on to the next section.

1. In Visual Studio, create a new C# **WPF Application** project (which is a .NET project). Be careful that you choose the project template with the exact name **WPF Application**, and not the **WPF App (.NET Framework)** one.
2. Give the project a name, and accept any default options.

You now have a project that builds an unpackaged WPF app.

Configure your WPF project for Windows App SDK support

First we'll edit the project file.

1. In **Solution Explorer**, right-click your project, and choose **Edit Project File**.

2. This step enables you to call Windows Runtime (WinRT) APIs (including Windows App SDK APIs). Inside the **PropertyGroup** element is the **TargetFramework** element, which is set to a value such as *net6.0*. Append to that target framework value a moniker (specifically, a Target Framework Moniker). For example, use the following if your app targets Windows 10, version 2004:

   ```
   XML

   <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
   ```

3. Also inside the **PropertyGroup** element, add a RuntimeIdentifiers element, as shown below. If you're targeting .NET 8 or later, then use the value `win-x86;win-x64;win-arm64` instead.

   ```
   XML

   <RuntimeIdentifiers>win-x86;win-x64;win-arm64</RuntimeIdentifiers>
   ```

4. By default, a WPF app is unpackaged (meaning that it isn't installed by using MSIX). An unpackaged app must initialize the Windows App SDK runtime before using any other feature of the Windows App SDK. You can do that automatically when your app starts via *auto-initialization*. You just set (also inside the

PropertyGroup element) the `WindowsPackageType` project property appropriately, like this:

```
XML
```

```xml
<WindowsPackageType>None</WindowsPackageType>
```

If you have advanced needs (such as custom error handling, or to load a specific version of the Windows App SDK), then instead of *auto-initialization* you can call the bootstrapper API explicitly—for more info, see Use the Windows App SDK runtime for apps packaged with external location or unpackaged.

5. Save and close the project file.

Next, we'll install the Windows App SDK NuGet package in the project.

6. In **Solution Explorer**, right-click the **Dependencies** node of your project, and choose **Manage Nuget Packages...**.
7. In the **NuGet Package Manager** window, select the **Browse** tab, and install the *Latest stable* **Microsoft.WindowsAppSDK** package.

Use some Windows App SDK features in your WPF app

This section offers a very simple example of calling Windows App SDK APIs from a WPF app. It uses the MRT Core feature (see Manage resources with MRT Core). If this example works for your WPF project (and if you created a new one for this walkthrough, then it will), then you can follow these steps.

1. Add the following markup to `MainWindow.xaml` (you could paste it inside the root **Grid**):

```
XAML
```

```xaml
<StackPanel>
    <Button HorizontalAlignment="Center" Click="Button_Click">Click me!
</Button>
    <TextBlock HorizontalAlignment="Center" x:Name="myTextBlock">Hello,
World!</TextBlock>
</StackPanel>
```

2. Now we'll add some code that uses the ResourceManager class in the Windows App SDK to load a string resource.

a. Add a new **Resources File (.resw)** item to your project (leave it with the default name of *Resources.resw*).

b. With the resources file open in the editor, create a new string resource with the following properties.

- Name: **Message**
- Value: **Hello, resources!**

c. Save and close the resources file.

d. In `MainWindow.xaml.cs`, add the following event handler:

```
C#

private void Button_Click(object sender, RoutedEventArgs e)
{
    // Construct a resource manager using the resource index generated
during build.
    var manager =
      new
Microsoft.Windows.ApplicationModel.Resources.ResourceManager();

    // Look up a string in the resources file using the string's name.
    myTextBlock.Text =
manager.MainResourceMap.GetValue("Resources/Message").ValueAsString;
}
```

3. Build the project, and run the app. Click the button to see the string `Hello, resources!` displayed.

> **Tip**
>
> If at runtime you see a message box indicating that the application needs a particular version of the Windows App Runtime, and asks whether you want to install it now, then click **Yes**. That will take you to **Latest downloads for the Windows App SDK**. For more info, see the **Prerequisites** section above.

Also see Runtime architecture to learn more about the *Framework* package dependency that your app takes when it uses the Windows App SDK, and the additional components required to work in an unpackaged app.

Package and deploy your WPF app with MSIX

Some Windows features and APIs (including the Windows App SDK notifications APIs) require your app to have *package identity* at runtime (in other words, your app needs to be *packaged*). For more info, see Features that require package identity.

1. In **Solution Explorer** in Visual Studio, right-click the solution, and choose **Add** > **New Project...**.
2. In the **Add a new project** dialog box, search for *packaging*, choose the *C#* **Windows Application Packaging Project** project template, and click **Next**.
3. Name the project, and click **Create**.
4. We want to specify which applications in the solution are to be included in the package. So in the packaging project (*not* the WPF project), right-click the **Dependencies** node, and choose **Add Project Reference...**.
5. In the list of projects in the solution, choose your WPF project, and click **OK**.
6. Expand the packaging project's **Dependencies** > **Applications** node, and confirm that your WPF project is referenced and highlighted in bold. This means that it will be used as a starting point for the package.
7. Right-click the packaging project, and choose **Set As Startup Project**.
8. Right-click the WPF project, and choose **Edit Project File**.
9. Delete `<WindowsPackageType>None</WindowsPackageType>`, save, and close.
10. In the **Solution Platforms** drop-down, pick *x64* (instead of *Any Cpu*).
11. Confirm that you can build and run.

Now that you've packaged your WPF app, you can call APIs that require package identity. So in `MainWindow.xaml.cs`, edit your event handler to look like this:

C#

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    var notification = new AppNotificationBuilder()
        .AddArgument("action", "viewConversation")
        .AddArgument("conversationId", "9813")
        .AddText("Andrew sent you a picture")
        .AddText("Check this out, The Enchantments in Washington!")
        .BuildNotification();

    AppNotificationManager.Default.Show(notification);
}
```

Build and run again. Click the button, and confirm that a toast notification is displayed. When called from a process that lacks package identity at runtime, the notifications APIs throw an exception.

> ⓘ **Note**

The steps in this section showed you how to create a *packaged app*. An alternative is to create a *packaged app with external location*. For a reminder of all these terms, see **Advantages and disadvantages of packaging your app**.

Related topics

- Windows Presentation Foundation (WPF)
- Install tools for the Windows App SDK
- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Latest downloads for the Windows App SDK
- Advantages and disadvantages of packaging your app
- Use the Windows App SDK runtime for apps packaged with external location or unpackaged
- Runtime architecture
- Features that require package identity

Use the Windows App SDK in a Windows Forms (WinForms) app

Article • 08/21/2024

The Windows App SDK is the next evolution in the Windows app development platform. But this topic shows how you can use Windows App SDK APIs (and Windows Runtime APIs) in a Windows Forms (WinForms) app!

- In many cases, you'll want to recreate your WinForms app in the form of a WinUI 3 app. Just one of the advantages of moving to WinUI 3 is to have access to the Fluent Design System (also see Design and code Windows apps). And WinUI 3 is part of the Windows App SDK—so, naturally, a WinUI 3 app can use the other Windows App SDK features and APIs, as well. This topic doesn't cover the process of migrating your WinForms app to WinUI 3.
- But if you find that you're using features of WinForms that aren't yet available in WinUI 3, then you can still use Windows App SDK features (such as App Lifecycle, MRT Core, DWriteCore, and others) in your WinForms app. This topic shows you how.

And in case you don't already have an existing WinForms project—or you want to practice the process—this topic includes steps to create a WinForms project so that you can follow along and configure it to call Windows App SDK APIs.

Prerequisites

1. Install tools for the Windows App SDK.
2. This topic covers both unpackaged and packaged WinForms apps. If your WinForms app is unpackaged (which WinForms apps are by default), then ensure that all dependencies for unpackaged apps are installed (see Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged). A quick way to do that is to visit Latest downloads for the Windows App SDK, then download and unzip and run one of the stable release **Runtime downloads**.

> **ⓘ Important**
>
> The version of the **Runtime** that you install needs to match the version of the **Microsoft.WindowsAppSDK** NuGet package that you'll install in a later step.

For more info about the terms *unpackaged* and *packaged*, see Advantages and disadvantages of packaging your app.

Create a WinForms project if you don't already have one

If you already have a WinForms project, then you can move on to the next section.

1. In Visual Studio, create a new C# **Windows Forms App** project (which is a .NET project). Be careful that you choose the project template with the exact name **Windows Forms App**, and not the **Windows Forms App (.NET Framework)** one.
2. Give the project a name, and accept any default options.

You now have a project that builds an unpackaged WinForms app.

Configure your WinForms project for Windows App SDK support

First we'll edit the project file.

1. In **Solution Explorer**, right-click your project, and choose **Edit Project File**.

2. This step enables you to call Windows Runtime (WinRT) APIs (including Windows App SDK APIs). Inside the **PropertyGroup** element is the **TargetFramework** element, which is set to a value such as *net6.0*. Append to that target framework value a moniker (specifically, a Target Framework Moniker). For example, use the following if your app targets Windows 10, version 2004:

   ```
   XML

   <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
   ```

3. Also inside the **PropertyGroup** element, add a RuntimeIdentifiers element, like this:

   ```
   XML

   <RuntimeIdentifiers>win-x86;win-x64;win-arm64</RuntimeIdentifiers>
   ```

4. By default, a WinForms app is unpackaged (meaning that it isn't installed by using MSIX). An unpackaged app must initialize the Windows App SDK runtime before using any other feature of the Windows App SDK. You can do that automatically

when your app starts via *auto-initialization*. You just set (also inside the **PropertyGroup** element) the `WindowsPackageType` project property appropriately, like this:

```
XML

<WindowsPackageType>None</WindowsPackageType>
```

If you have advanced needs (such as custom error handling, or to load a specific version of the Windows App SDK), then instead of *auto-initialization* you can call the bootstrapper API explicitly—for more info, see Use the Windows App SDK runtime for apps packaged with external location or unpackaged.

5. Save and close the project file.

Next, we'll install the Windows App SDK NuGet package in the project.

6. In **Solution Explorer**, right-click the **Dependencies** node of your project, and choose **Manage Nuget Packages...**.
7. In the **NuGet Package Manager** window, select the **Browse** tab, and install the *Latest stable* **Microsoft.WindowsAppSDK** package.

Use some Windows App SDK features in your WinForms app

This section offers a very simple example of calling Windows App SDK APIs from a WinForms app. It uses the MRT Core feature (see Manage resources with MRT Core). If this example works for your WinForms project (and if you created a new one for this walkthrough, then it will), then you can follow these steps.

1. Open `Form1.cs` (using the **View Designer** command), and drag a **Button** and a **Label** out of the **Toolbox** and onto the designer.

2. Double-click *button1* to generate an event handler.

3. Now we'll add some code that uses the ResourceManager class in the Windows App SDK to load a string resource.

   a. Add a new **Resources File (.resw)** item to your project (leave it with the default name of *Resources.resw*).

   b. With the resources file open in the editor, create a new string resource with the following properties.

- Name: **Message**
- Value: **Hello, resources!**

c. Save and close the resources file.

d. Open `Form1.cs` (using the **View Code** command), and edit the event handler to look like this:

```
C#

private void button1_Click(object sender, EventArgs e)
{
    // Construct a resource manager using the resource index generated
during build.
    var manager =
        new
Microsoft.Windows.ApplicationModel.Resources.ResourceManager();

    // Look up a string in the resources file using the string's name.
    label1.Text =
manager.MainResourceMap.GetValue("Resources/Message").ValueAsString;
}
```

4. Build the project, and run the app. Click the button to see the string `Hello, resources!` displayed.

> **Tip**
>
> If at runtime you see a message box indicating that the application needs a particular version of the Windows App Runtime, and asks whether you want to install it now, then click **Yes**. That will take you to **Latest downloads for the Windows App SDK**. For more info, see the **Prerequisites** section above.

Also see Runtime architecture to learn more about the *Framework* package dependency that your app takes when it uses the Windows App SDK, and the additional components required to work in an unpackaged app.

Package and deploy your WinForms app with MSIX

Some Windows features and APIs (including the Windows App SDK notifications APIs) require your app to have *package identity* at runtime (in other words, your app needs to be *packaged*). For more info, see Features that require package identity.

1. In **Solution Explorer** in Visual Studio, right-click the solution, and choose **Add** > **New Project...**.
2. In the **Add a new project** dialog box, search for *packaging*, choose the C# **Windows Application Packaging Project** project template, and click **Next**.
3. Name the project, and click **Create**.
4. We want to specify which applications in the solution are to be included in the package. So in the packaging project (*not* the WinForms project), right-click the **Dependencies** node, and choose **Add Project Reference...**.
5. In the list of projects in the solution, choose your WinForms project, and click **OK**.
6. Expand the packaging project's **Dependencies** > **Applications** node, and confirm that your WinForms project is referenced and highlighted in bold. This means that it will be used as a starting point for the package.
7. Right-click the packaging project, and choose **Set As Startup Project**.
8. Right-click the WinForms project, and choose **Edit Project File**.
9. Delete `<WindowsPackageType>None</WindowsPackageType>`, save, and close.
10. In the **Solution Platforms** drop-down, pick *x64* (instead of *Any Cpu*).
11. Confirm that you can build and run.

Now that you've packaged your WinForms app, you can call APIs that require package identity. So open `Form1.cs` (using the **View Code** command), and edit the event handler to look like this:

C#

```
private void button1_Click(object sender, EventArgs e)
{
    var notification = new AppNotificationBuilder()
        .AddArgument("action", "viewConversation")
        .AddArgument("conversationId", "9813")
        .AddText("Andrew sent you a picture")
        .AddText("Check this out, The Enchantments in Washington!")
        .BuildNotification();

    AppNotificationManager.Default.Show(notification);
}
```

Build and run again. Click the button, and confirm that a toast notification is displayed. When called from a process that lacks package identity at runtime, the notifications APIs throw an exception.

> **Note**
>
> The steps in this section showed you how to create a *packaged app*. An alternative is to create a *packaged app with external location*. For a reminder of all these terms,

see Advantages and disadvantages of packaging your app.

Related topics

- Windows Forms (WinForms)
- Install tools for the Windows App SDK
- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Latest downloads for the Windows App SDK
- Advantages and disadvantages of packaging your app
- Use the Windows App SDK runtime for apps packaged with external location or unpackaged
- Runtime architecture
- Features that require package identity

Update existing projects to a different release of the Windows App SDK

Article • 04/19/2023

If you created a project with an earlier version of the Windows App SDK (previously called Project Reunion) or WinUI 3, then you can update the project to use a more recent release. To learn more about what's currently available in each release channel, see Windows App SDK release channels.

> **Note**
>
> These instructions might have issues due to the uniqueness of each app's individual scenario. Please carefully follow them, and if you find an issue then please file a bug against the microsoft-ui-xaml GitHub repo.

Update between versions released after 1.0

If your project isn't referencing the version of the Windows App SDK NuGet package that you need, then you can use the **NuGet Package Manager** in Visual Studio to update your project's NuGet package references. For example, if you create a new project by using a stable release of the Windows App SDK VSIX, then your project will reference a stable release of the Windows App SDK. But you can easily reconfigure that project to reference, say, an experimental release of the Windows App SDK. Or reconfigure it to reference the latest stable release.

For steps, see the instructions in Use the Windows App SDK in an existing project.

Update from 0.8 to 1.0

If you created a project using version 0.8 (for example, version 0.8.4), then you can follow these instructions to update your project to the 1.0 release.

Prerequisite: Download and install the latest release of the Windows App SDK. For more information, see Install tools for the Windows App SDK.

Instructions

1. In the `.wapproj` file, if your **TargetPlatformMinVersion** is older than `10.0.17763.0`, then change it to `10.0.17763.0`.

2. In Visual Studio, go to **Tools** > **Nuget Package Manager** > **Package Manager Console**. This process consists of uninstalling existing Project Reunion package references from `.csproj`/`.vcxproj` and `.wapproj` files, and then installing the `WindowsAppSDK` package references to those files.

3. Enter the following commands to uninstall existing `ProjectReunion` packages from your `.csproj`/`.vcxproj`

```
Console

uninstall-package Microsoft.ProjectReunion -ProjectName {yourProject}
uninstall-package Microsoft.ProjectReunion.Foundation -ProjectName
{yourProject}
uninstall-package Microsoft.ProjectReunion.WinUI -ProjectName
{yourProject}
```

4. Then, run the following to uninstall existing `ProjectReunion` packages from your `.wapproj`:

```
Console

uninstall-package Microsoft.ProjectReunion
uninstall-package Microsoft.ProjectReunion.WinUI
```

5. Now run the commands to install the stable `WindowsAppSDK` package.

6. To add the `WindowsAppSDK` package reference to your `.csproj`/`.vcxproj`:

```
Console

install-package Microsoft.WindowsAppSDK -ProjectName {yourProject} -
Version 1.0.0
```

7. To add the `WindowsAppSDK` package reference to your `.wapproj`:

```
Console

install-package Microsoft.WindowsAppSDK -Version 1.0.0
```

Update from 0.8 or 0.8 Preview to 1.0 Experimental or Preview 3

> **ⓘ Important**
>
> Version 1.0 Preview 1 and Preview 2 contain a critical bug. If you've already installed one of these previews, see Important issue impacting 1.0 Preview 1 and Preview 2. We recommend using version Version 1.0 Preview 3 (1.0.0-preview3) instead.

If you created a project using version 0.8 Preview or any version of 0.8 (for example, version 0.8.1), you can follow these instructions to update your project to the 1.0 Preview 3 or Experimental release.

Before starting, make sure you have all the Windows App SDK prerequisites installed, including the latest VSIX and NuGet package. For more details, see Install tools for the Windows App SDK.

First, do the following:

- In the .wapproj file, if your **TargetPlatformMinVersion** is older than 10.0.17763.0, change it to 10.0.17763.0.

Next, make these changes to your project:

1. In Visual Studio, go to **Tools** > **Nuget Package Manager** > **Package Manager Console**.
2. Enter the following commands for 1.0 Preview 3:

   Console

   ```
   uninstall-package Microsoft.ProjectReunion -ProjectName {yourProject}
   uninstall-package Microsoft.ProjectReunion.Foundation -ProjectName
   {yourProject}
   uninstall-package Microsoft.ProjectReunion.WinUI -ProjectName
   {yourProject}
   install-package Microsoft.WindowsAppSDK -Version 1.0.0-preview3 -
   ProjectName {yourProjectName}
   ```

   Or the following commands for 1.0 Experimental:

   Console

   ```
   uninstall-package Microsoft.ProjectReunion -ProjectName {yourProject}
   uninstall-package Microsoft.ProjectReunion.Foundation -ProjectName
   ```

```
{yourProject}
uninstall-package Microsoft.ProjectReunion.WinUI -ProjectName
{yourProject}
install-package Microsoft.WindowsAppSDK -Version 1.0.0-experimental1 -
ProjectName {yourProjectName}
```

3. Make the following changes in your Application (package).wapproj:

   a. Remove this item group (if you're updating from a different version than 0.8.0, you will see that corresponding version number referenced in this item group):

   XML

   ```
   <ItemGroup>
       <PackageReference Include="Microsoft.ProjectReunion" Version="
   [0.8.0]">
       <IncludeAssets>build</IncludeAssets>
       </PackageReference>
       <PackageReference Include="Microsoft.ProjectReunion.WinUI"
   Version="[0.8.0]">
       <IncludeAssets>build</IncludeAssets>
       </PackageReference>
   </ItemGroup>
   ```

   b. Add this item group to replace it with 1.0 Preview 3:

   XML

   ```
   <ItemGroup>
       <PackageReference Include="Microsoft.WindowsAppSDK" Version="
   [1.0.0-preview3]">
       <IncludeAssets>build</IncludeAssets>
       </PackageReference>
   </ItemGroup>
   ```

   Or this item group to replace it with 1.0 Experimental:

   XML

   ```
   <ItemGroup>
       <PackageReference Include="Microsoft.WindowsAppSDK" Version="
   [1.0.0-experimental1]">
       <IncludeAssets>build</IncludeAssets>
       </PackageReference>
       <PackageReference Include="Microsoft.WindowsAppSDK.WinUI"
   Version="[1.0.0-experimental1]">
       <IncludeAssets>build</IncludeAssets>
       </PackageReference>
   </ItemGroup>
   ```

4. Make the following changes to your project (.csproj or .vcproj) file:

    a. Remove this item group (if you're updating from a different version than 0.8.0, you will see that corresponding version number referenced in this item group):

    ```
    XML
    ```

    ```xml
    <ItemGroup>
        <PackageReference Include="Microsoft.ProjectReunion"
    Version="0.8.0" />
        <PackageReference Include="Microsoft.ProjectReunion.Foundation"
    Version="0.8.0" />
        <PackageReference Include="Microsoft.ProjectReunion.WinUI"
    Version="0.8.0" />
        <Manifest Include="$(ApplicationManifest)" />
    </ItemGroup>
    ```

    b. Add this item group to replace it with 1.0 Preview 3:

    ```
    XML
    ```

    ```xml
    <ItemGroup>
        <PackageReference Include="Microsoft.WindowsAppSDK"
    Version="1.0.0-preview3" />
        <Manifest Include="$(ApplicationManifest)" />
    </ItemGroup>
    ```

    Or this item group to replace it with 1.0 Experimental:

    ```
    XML
    ```

    ```xml
    <ItemGroup>
        <PackageReference Include="Microsoft.WindowsAppSDK"
    Version="1.0.0-experimental1" />
        <PackageReference Include="Microsoft.WindowsAppSDK.Foundation"
    Version="1.0.0-experimental1" />
        <PackageReference Include="Microsoft.WindowsAppSDK.WinUI"
    Version="1.0.0-experimental1" />
        <Manifest Include="$(ApplicationManifest)" />
    </ItemGroup>
    ```

5. If your solution fails to build, clean the build output, restart Visual Studio, and try re-running the app.

Update from 0.8 Preview to 0.8 or between stable 0.8 versions

If you created a project using version 0.8 Preview, you can follow these instructions to update your project to a stable version of 0.8. These instructions also apply if you've created a project with an older stable version of 0.8 (for example, 0.8.0) and want to update your project to a newer stable version (for example, 0.8.2).

> **Note**
>
> You may be able to automatically update your project through the Visual Studio Extension Manager, without going through the manual steps below. In Visual Studio 2019, click on **Extensions** > **Manage Extensions** and select **Updates** from the left menu bar. Select "Project Reunion" from the list and click **Update**.

Before starting, make sure you have all the Windows App SDK prerequisites installed, including the latest VSIX and NuGet package. For more details, see Install tools for the Windows App SDK.

First, do the following:

- In the .wapproj file, if your **TargetPlatformMinVersion** is older than 10.0.17763.0, change it to 10.0.17763.0.

Next, make these changes to your project:

1. In Visual Studio, go to **Tools** > **Nuget Package Manager** > **Package Manager Console**.
2. Enter the following commands:

   Console

   ```
   uninstall-package Microsoft.ProjectReunion -ProjectName {yourProject}
   uninstall-package Microsoft.ProjectReunion.Foundation -ProjectName
   {yourProject}
   uninstall-package Microsoft.ProjectReunion.WinUI -ProjectName
   {yourProject}
   install-package Microsoft.ProjectReunion -Version 0.8.2 -ProjectName
   {yourProjectName}
   ```

3. Make the following changes in your Application (package).wapproj:

   a. Remove this item group (if you're updating from a different version than 0.8 Preview, you will see that corresponding version number referenced in this item group):

      XML

```
<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion" Version="
[0.8.0-preview]">
    <IncludeAssets>build</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.ProjectReunion.WinUI"
Version="[0.8.0-preview]">
    <IncludeAssets>build</IncludeAssets>
    </PackageReference>
</ItemGroup>
```

b. Add this item group to replace it:

XML

```
<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion" Version="
[0.8.2]">
    <IncludeAssets>build</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.ProjectReunion.WinUI"
Version="[0.8.2]">
    <IncludeAssets>build</IncludeAssets>
    </PackageReference>
</ItemGroup>
```

4. Make the following changes to your project (.csproj or .vcproj) file:
   a. Remove this item group (if you're updating from a different version than 0.8 Preview, you will see that corresponding version number referenced in this item group):

XML

```
<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion"
Version="0.8.0-preview" />
    <PackageReference Include="Microsoft.ProjectReunion.Foundation"
Version="0.8.0-preview" />
    <PackageReference Include="Microsoft.ProjectReunion.WinUI"
Version="0.8.0-preview" />
    <Manifest Include="$(ApplicationManifest)" />
</ItemGroup>
```

   b. Add this item group to replace it:

XML

```
<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion"
```

```
Version="0.8.2" />
    <PackageReference Include="Microsoft.ProjectReunion.Foundation"
Version="0.8.2" />
    <PackageReference Include="Microsoft.ProjectReunion.WinUI"
Version="0.8.2" />
    <Manifest Include="$(ApplicationManifest)" />
</ItemGroup>
```

5. If your solution fails to build, clean the build output, restart Visual Studio, and try re-running the app.

Update from 0.5 to 0.8

If you created a project using version 0.5 stable, you can follow these instructions to update your project to version 0.8 stable.

> **Note**
>
> You may be able to automatically update your project through the Visual Studio Extension Manager, without going through the manual steps below. In Visual Studio 2019, click on **Extensions** > **Manage Extensions** and select **Updates** from the left menu bar. Select "Project Reunion" from the list and click **Update**.

Before starting, make sure you have all the Windows App SDK prerequisites installed, including the latest VSIX and NuGet package. For more details, see Install tools for the Windows App SDK.

First, do the following:

- In the .wapproj file, if your **TargetPlatformMinVersion** is older than 10.0.17763.0, change it to 10.0.17763.0.

Next, make these changes to your project:

1. In Visual Studio, go to **Tools** > **Nuget Package Manager** > **Package Manager Console**.

2. Enter the following commands:

```
Console

uninstall-package Microsoft.ProjectReunion -ProjectName {yourProject}
uninstall-package Microsoft.ProjectReunion.Foundation -ProjectName
{yourProject}
uninstall-package Microsoft.ProjectReunion.WinUI -ProjectName
```

```
{yourProject}
install-package Microsoft.ProjectReunion -Version 0.8.0 -ProjectName
{yourProjectName}
```

3. Add the following line to your project (.csproj or .vcproj) file, inside the first `<PropertyGroup>`:

```XML
<UseWinUI>true</UseWinUI>
```

4. Make the following changes in your Application (package).wapproj:

   a. Add this section:

```XML
<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion" Version="
[0.8.0]">
    <IncludeAssets>build</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.ProjectReunion.WinUI"
Version="[0.8.0]">
    <IncludeAssets>build</IncludeAssets>
    </PackageReference>
</ItemGroup>
```

   b. Locate the following line:

```XML
<AssetTargetFallback>net5.0-
windows$(TargetPlatformVersion);$(AssetTargetFallback)
</AssetTargetFallback>
```

   Move this line and place it on a new line directly beneath the `<TargetPlatformVersion>` tag.

   c. Remove this item group (if you're updating from an earlier version than 0.5.7, you will see an earlier version number referenced in this item group):

```XML
<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion" Version="
[0.5.7]">
```

```
    <IncludeAssets>build</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.ProjectReunion.WinUI"
Version="[0.5.7]">
    <IncludeAssets>build</IncludeAssets>
    </PackageReference>
</ItemGroup>
```

5. If your solution fails to build, clean the build output, restart Visual Studio, and try re-running the app.

Update from 0.5 Preview to 0.8 Preview

If you created a project using version 0.5 preview, you can follow these instructions to update your project to use version 0.8 preview.

> **Note**
>
> You may be able to automatically update your project through the Visual Studio Extension Manager, without going through the manual steps below. In Visual Studio 2019, click on **Extensions** > **Manage Extensions** and select **Updates** from the left menu bar. Select "Project Reunion" from the list and click **Update**.

Before starting, make sure you have all the Windows App SDK prerequisites installed, including the latest VSIX and NuGet package. For more details, see Install tools for the Windows App SDK.

First, do the following:

- In the .wapproj file, if your **TargetPlatformMinVersion** is older than 10.0.17763.0, change it to 10.0.17763.0.
- The default project templates for both C++ and C# apps included the following lines. The `Application.Suspending` event is no longer called for desktop apps, so be sure to remove these lines (and any other uses of this event) if they are still present in your code:

  C#

  ```
  this.Suspending += OnSuspending;
  ```

  C++

```
Suspending({ this, &App::OnSuspending });
```

Next, make these changes to your project:

1. In Visual Studio, go to **Tools** > **Nuget Package Manager** > **Package Manager Console**.

2. Enter the following commands:

```
Console

uninstall-package Microsoft.ProjectReunion -ProjectName {yourProject}
uninstall-package Microsoft.ProjectReunion.Foundation -ProjectName
{yourProject}
uninstall-package Microsoft.ProjectReunion.WinUI -ProjectName
{yourProject}
install-package Microsoft.ProjectReunion -Version 0.8.0-preview -
ProjectName {yourProjectName}
```

3. Make the following changes in your Application (package).wapproj:

   a. Add this section:

```
XML

<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion" Version="
[0.8.0-preview]">
    <IncludeAssets>build</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.ProjectReunion.WinUI"
Version="[0.8.0-preview]">
    <IncludeAssets>build</IncludeAssets>
    </PackageReference>
</ItemGroup>
```

   b. Add the following line to a new line directly beneath the `<TargetPlatformVersion>` tag.

```
XML

<AssetTargetFallback>net5.0-
windows$(TargetPlatformVersion);$(AssetTargetFallback)
</AssetTargetFallback>
```

   c. Remove this line:

```
XML

<AppxTargetsLocation
Condition="'$(AppxTargetsLocation)'==''">$(MSBuildThisFileDirectory)
build\</AppxTargetsLocation>
```

And these lines:

```
XML

<Import Project="$(Microsoft_ProjectReunion_AppXReference_props)" />
<Import Project="$(Microsoft_WinUI_AppX_targets)" />
```

And this item group:

```
XML

<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion" Version="
[0.5.0]" GeneratePathProperty="true">
      <ExcludeAssets>all</ExcludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.ProjectReunion.WinUI"
Version="[0.5.0]" GeneratePathProperty="true">
      <ExcludeAssets>all</ExcludeAssets>
    </PackageReference>
</ItemGroup>
```

4. Make the following changes to your project (.csproj or .vcproj) file:

   a. Remove this item group:

```
XML

<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion"
Version="0.5.0-prerelease" />
    <PackageReference Include="Microsoft.ProjectReunion.Foundation"
Version="0.5.0-prerelease" />
    <PackageReference Include="Microsoft.ProjectReunion.WinUI"
Version="0.5.0-prerelease" />
    <Manifest Include="$(ApplicationManifest)" />
</ItemGroup>
```

   b. Add this item group:

```
XML
```

```
<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion"
Version="0.8.0-preview" />
    <PackageReference Include="Microsoft.ProjectReunion.Foundation"
Version="0.8.0-preview" />
    <PackageReference Include="Microsoft.ProjectReunion.WinUI"
Version="0.8.0-preview" />
    <Manifest Include="$(ApplicationManifest)" />
</ItemGroup>
```

5. If your solution fails to build, clean the build output, restart Visual Studio, and try re-running the app.

Update from 0.5 Preview to 0.5

If you created a project using version 0.5 preview, you can follow these instructions to update your project to stable version 0.5.7.

Before starting, make sure you have all the Windows App SDK prerequisites installed, including the latest VSIX and NuGet package. For more details, see Install tools for the Windows App SDK.

First, do the following:

- **[Desktop apps only]** In the .wapproj file, if your **TargetPlatformMinVersion** is older than 10.0.17763.0, change it to 10.0.17763.0.
- The default project templates for both C++ and C# apps included the following lines. The `Application.Suspending` event is no longer called for desktop apps, so be sure to remove these lines (and any other uses of this event) if they are still present in your code:

  C#

  ```
  this.Suspending += OnSuspending;
  ```

  C++

  ```
  Suspending({ this, &App::OnSuspending });
  ```

Next, make these changes to your project:

1. In Visual Studio, go to **Tools** > **Nuget Package Manager** > **Package Manager Console**.

2. Enter the following commands:

```
Console

uninstall-package Microsoft.ProjectReunion -ProjectName {yourProject}
uninstall-package Microsoft.ProjectReunion.Foundation -ProjectName
{yourProject}
uninstall-package Microsoft.ProjectReunion.WinUI -ProjectName
{yourProject}
install-package Microsoft.ProjectReunion -Version 0.5.7 -ProjectName
{yourProjectName}
```

3. If you have a UWP app, your update process should be complete at this stage. If you have a desktop app, make the following changes in your Application (package).wapproj:

   a. Add this section:

```
XML

<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion" Version="
[0.5.7]">
        <IncludeAssets>build</IncludeAssets>
    </PackageReference>
</ItemGroup>
```

   b. Locate the `<TargetPlatformVersion>` tag, and add the following on a new line directly beneath that tag

```
XML

<AssetTargetFallback>net5.0-
windows$(TargetPlatformVersion);$(AssetTargetFallback)
</AssetTargetFallback>
```

   c. Remove this line:

```
XML

<AppxTargetsLocation
Condition="'$(AppxTargetsLocation)'==''">$(MSBuildThisFileDirectory)
build\</AppxTargetsLocation>
```

   And these lines:

```
XML
```

```
<Import Project="$(Microsoft_ProjectReunion_AppXReference_props)" />
<Import Project="$(Microsoft_WinUI_AppX_targets)" />
```

And this item group:

XML

```
<ItemGroup>
    <PackageReference Include="Microsoft.ProjectReunion" Version="
[0.5.0-prerelease]" GeneratePathProperty="true">
      <ExcludeAssets>all</ExcludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.ProjectReunion.WinUI"
Version="[0.5.0-prerelease]" GeneratePathProperty="true">
      <ExcludeAssets>all</ExcludeAssets>
    </PackageReference>
</ItemGroup>
```

Update from WinUI 3 Preview 4 to 0.5

If you created a desktop app using WinUI 3 Preview 4, you can follow these instructions to update your project to Project Reunion 0.5.

Before starting, make sure you have all the Windows App SDK prerequisites installed, including the latest VSIX and NuGet package. For more details, see Install tools for the Windows App SDK.

First, do the following:

- In the .wapproj file, if your TargetPlatformMinVersion is older than 10.0.17763.0, change it to 10.0.17763.0.
- The default project templates for both C++ and C# apps included the following lines. The `Application.Suspending` event is no longer called for desktop apps, so be sure to remove these lines (and any other uses of this event) if they are still present in your code:

  C#

  ```
  this.Suspending += OnSuspending;
  ```

  C++

```
Suspending({ this, &App::OnSuspending });
```

Next, make these changes to your project:

1. In Visual Studio, go to **Tools** > **Nuget Package Manager** > **Package Manager Console**.

2. Enter the following commands:

```
Console

uninstall-package Microsoft.WinUI -ProjectName {yourProject}
install-package Microsoft.ProjectReunion -Version 0.5.7 -ProjectName
{yourProjectName}
```

3. Make the following changes in your Application (package).wapproj:

   a. Add this section:

```
XML

<ItemGroup>
  <PackageReference Include="Microsoft.ProjectReunion" Version="
[0.5.7]">
    <IncludeAssets>build</IncludeAssets>
  </PackageReference>
</ItemGroup>
```

   b. Remove the following lines:

```
XML

<AppxTargetsLocation
Condition="'$(AppxTargetsLocation)'==''">$(MSBuildThisFileDirectory)
build\</AppxTargetsLocation>
```

```
XML

<Import Project="$(AppxTargetsLocation)Microsoft.WinUI.AppX.targets"
/>
```

4. Delete the existing `Microsoft.WinUI.AppX.targets` file under the {YourProject} (package)/build/ folder of your project.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

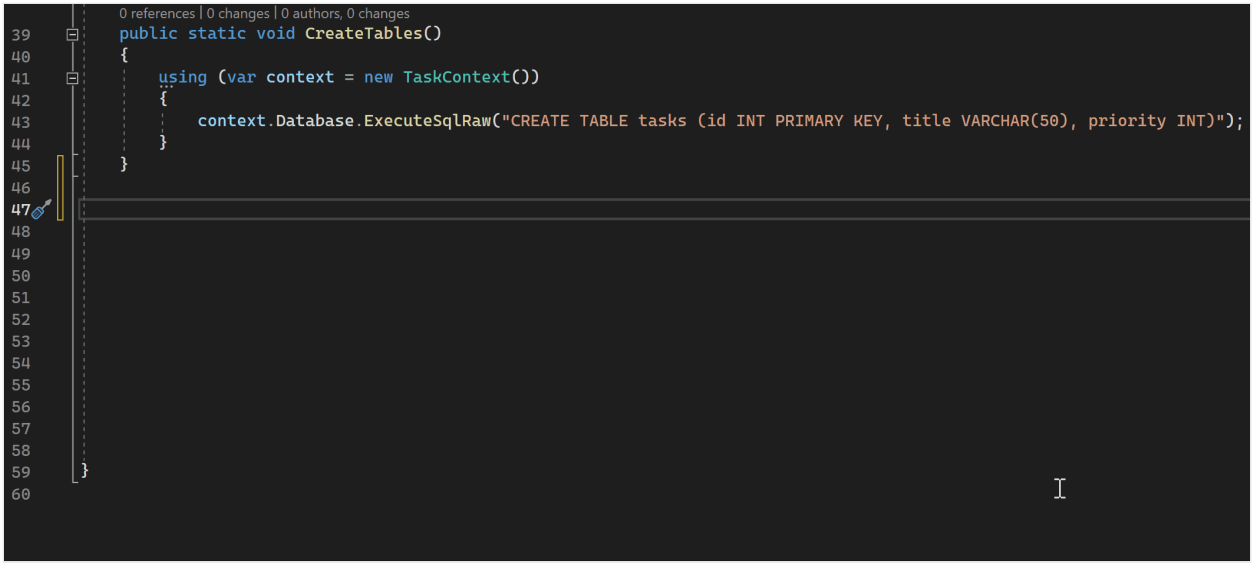
Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Use GitHub Copilot to create WinUI 3 / Windows App SDK apps in Visual Studio

Article • 11/27/2024

In this how-to, we'll demonstrate how GitHub Copilot can be used to build WinUI 3 / Windows App SDK desktop apps in Visual Studio. This guide builds upon *What is GitHub Copilot Completions for Visual Studio?*, offering tailored tips and best practices for Copilot-assisted WinUI 3 app development.



Prerequisites

- Visual Studio 2022 (v17.10+) with the Windows application development workload applied (see Get started with WinUI for additional configuration details). GitHub Copilot is included in Visual Studio 2022 v17.10 and later by default.
- An active subscription to GitHub Copilot associated with the GitHub account that you sign in to Visual Studio with.
- Familiarity with C#, WinUI, and Windows App SDK.

Use GitHub Copilot

Autocomplete your code snippets

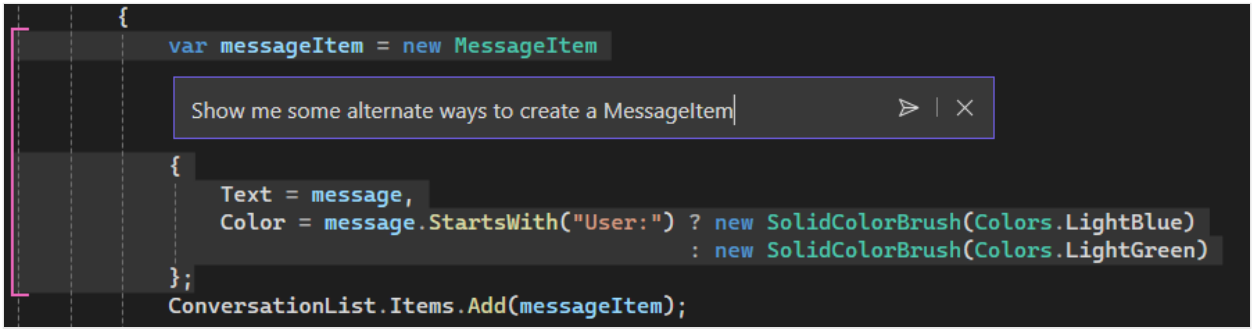
GitHub Copilot in Visual Studio provides real-time code suggestions and completions based on the code you write. The most straightforward way to use Copilot is to start typing code in the editor, and Copilot will try to autocomplete your code snippet. You can then accept or dismiss the suggestions:

```
<ItemsControl x:Name="ConversationList" Width="300">
    <ItemsControl.ItemTemplate>
        <DataTemplate>
            <Image x:Name="AvatarImage" Source="{Binding Avatar}" Width="50" Height="50" HorizontalAlignment="Left"/>
        </DataTemplate>
    </ItemsControl.ItemTemplate>
</ItemsControl>
```

> **Tip**
>
> If you don't see the GitHub Copilot suggestions, you can enable different aspects of the feature in Visual Studio's options under `Tools` -> `Options` -> `GitHub` -> `Copilot`.

Ask Copilot for suggestions

Right-click in the code editor and select `Ask Copilot`. A prompt window will open where you can chat inline with Copilot to get a list of suggestions based on the current cursor position and your prompt:



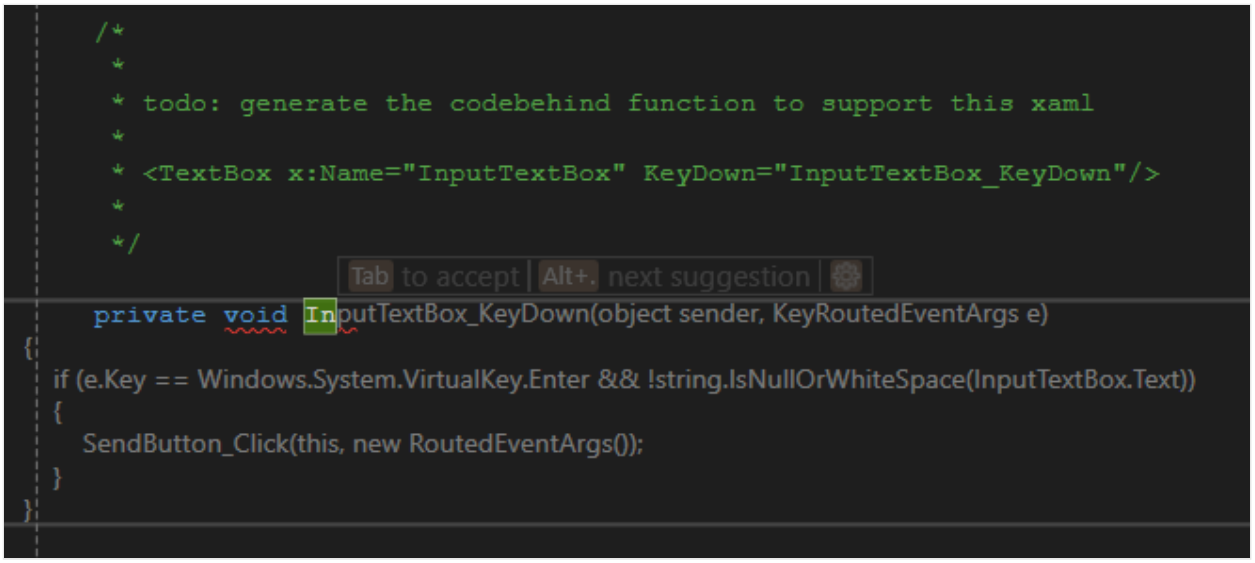
Prompt Copilot with plain-language comments

Although Copilot is used primarily for code completion, you can also use natural language comments to guide Copilot in generating specific code snippets. For example, you can use comments to request a specific feature or functionality:

```
<ItemsControl.ItemTemplate>
    <DataTemplate>
        <!-- we want an Image with max width of 200 -->
        <Image Source="{Binding}" MaxWidth="200"/>
    </DataTemplate>
</ItemsControl.ItemTemplate>
```

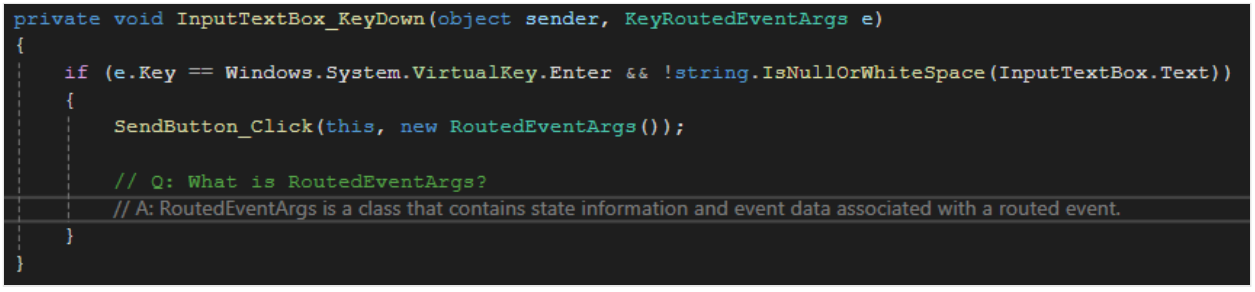
Use temporary comments to add code from other files to Copilot's context

If you're working on a code-behind file and want Copilot to incorporate context from the associated XAML file, you can use temporary comments to include this additional code within Copilot's context. Here's an example of how you can specify the XAML code first, and then have Copilot generate the corresponding C# code:



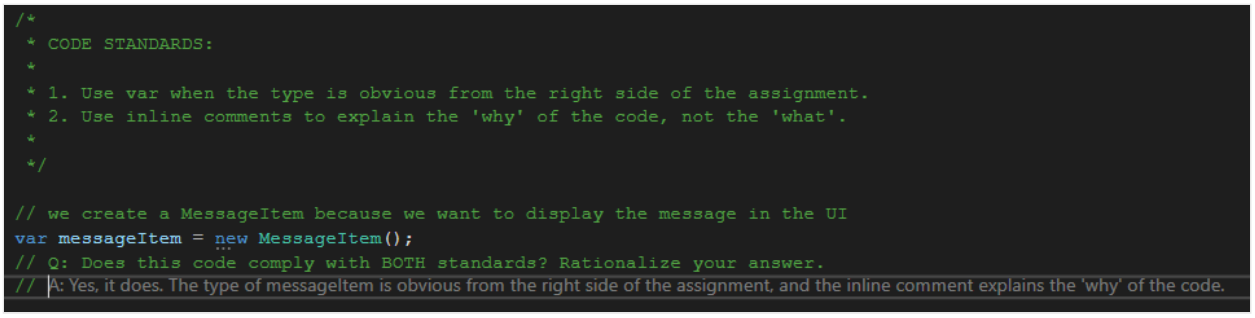
Ask Copilot to explain how something works with inline comments

You can use inline comments to ask Copilot to explain how a specific piece of code works. This is similar to using the inline Ask Copilot feature or the Copilot Chat window, except your prompt is typed directly into the code editor:



Use Copilot to test code standards

You can use Copilot to generate code that adheres to your project's coding standards, and to test any given snippet's adherence to those standards. Here's an example of how you can use inline comments to specify two conventions, and then have Copilot validate the code snippet against these conventions:



Recap

In this how-to, we demonstrated how to use GitHub Copilot in Visual Studio to assist you with WinUI 3 / Windows App SDK desktop app development. We covered how to:

- Autocomplete your code snippets.
- Generate autocomplete suggestions inline with Ask Copilot.
- Prompt Copilot with plain-language comments.
- Use temporary comments to add code from other files to Copilot's context.
- Ask Copilot to explain how something works with inline comments.
- Use Copilot to test and enforce code standards.

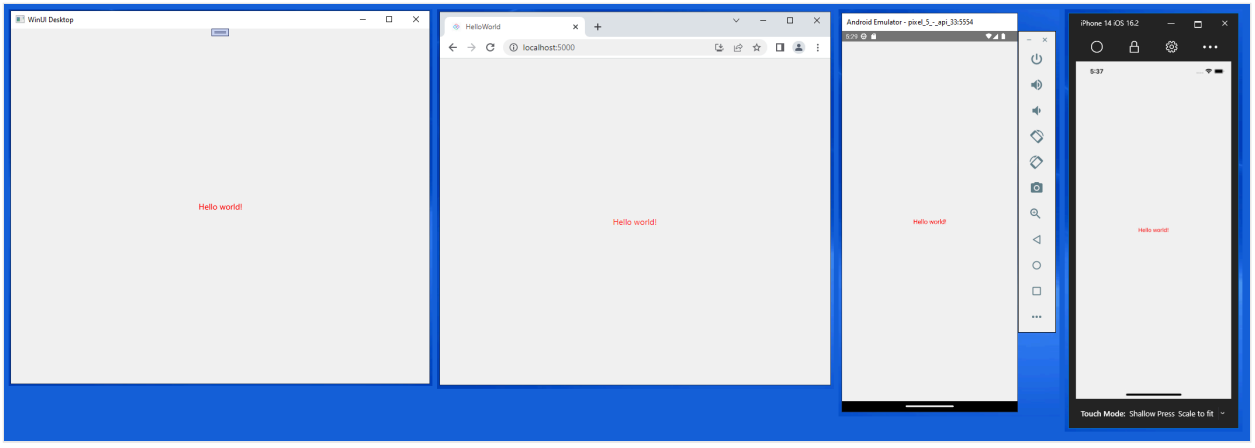
Related content

- Sample applications for Windows development
- Windows developer FAQ
- Windows development best practices

How to target multiple platforms with your WinUI 3 app

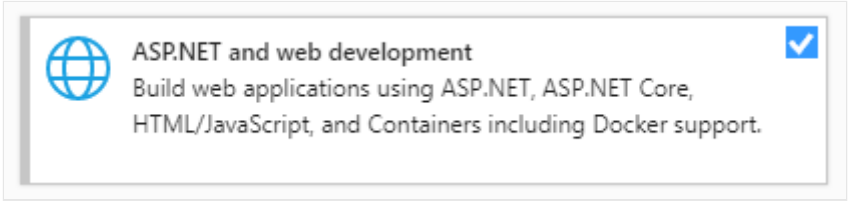
Article • 07/09/2024

Once you've created a starter Hello World WinUI 3 app, you might be wondering how to reach more users with a single codebase. This how-to will use Uno Platform to expand the reach of your existing application enabling reuse of the business logic and UI layer across native mobile, web, and desktop.

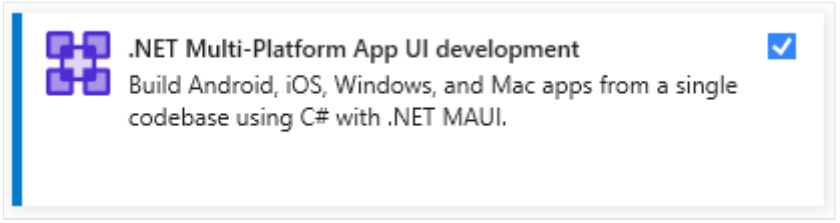


Prerequisites

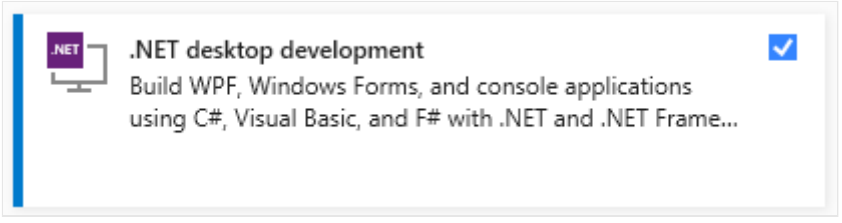
- Visual Studio 2022 17.4 or later
- Set up your development computer (see Get started with WinUI)
- ASP.NET and web development workload (for WebAssembly development)



- .NET Multi-platform App UI development installed (for iOS, Android, Mac Catalyst development).



- .NET desktop development installed (for Gtk, Wpf, and Linux Framebuffer development)



Finalize your environment

1. Open a command-line prompt, Windows Terminal if you have it installed, or else Command Prompt or Windows Powershell from the Start Menu.

2. Install or update the `uno-check` tool:

   - Use the following command:

     .NET CLI

     ```
     dotnet tool install -g uno.check
     ```

   - To update the tool, if you already have previously installed an older version:

     .NET CLI

     ```
     dotnet tool update -g uno.check
     ```

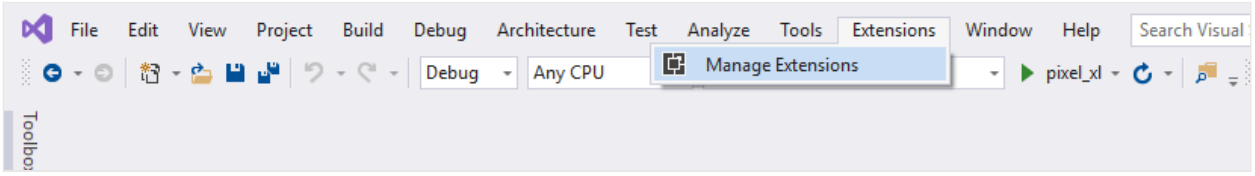
3. Run the tool with the following command:

   ```
   uno-check
   ```

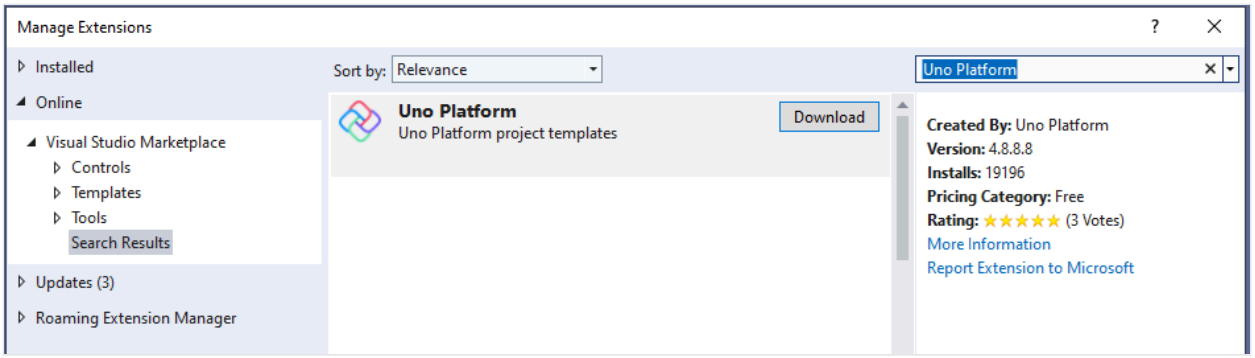
4. Follow the instructions indicated by the tool. Because it needs to modify your system, you may be prompted for elevated permissions.

Install the Uno Platform solution templates

Launch Visual Studio, then click `Continue without code`. Click `Extensions` -> `Manage Extensions` from the Menu Bar.



In the Extension Manager expand the **Online** node and search for `Uno`, install the `Uno Platform` extension, or download it from the Visual Studio Marketplace, then restart Visual Studio.

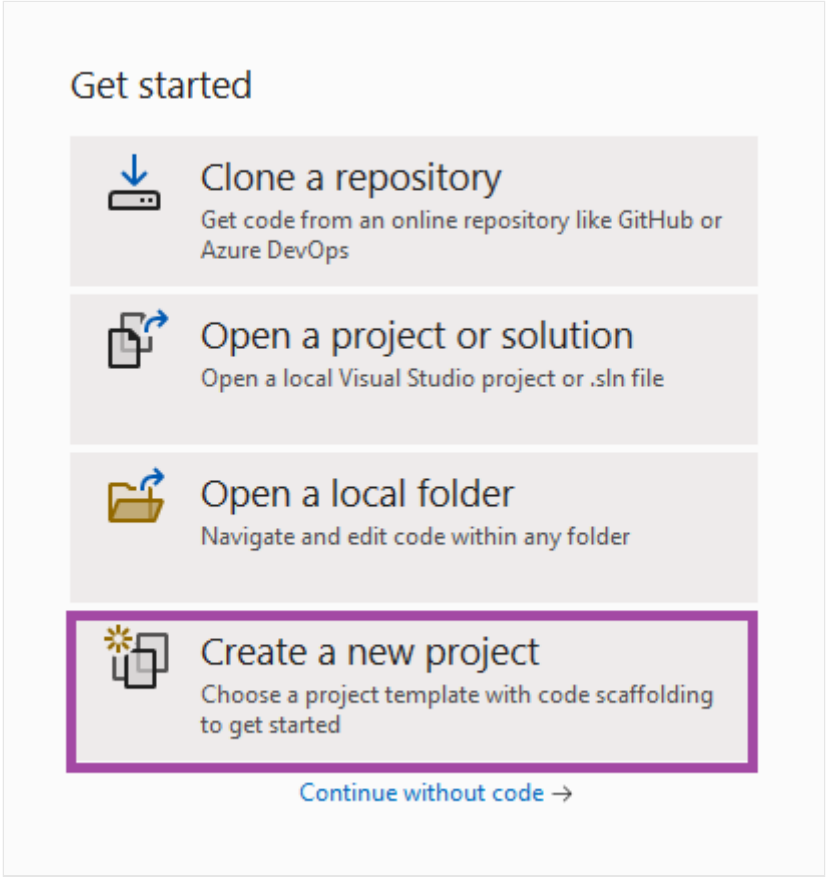


Create an application

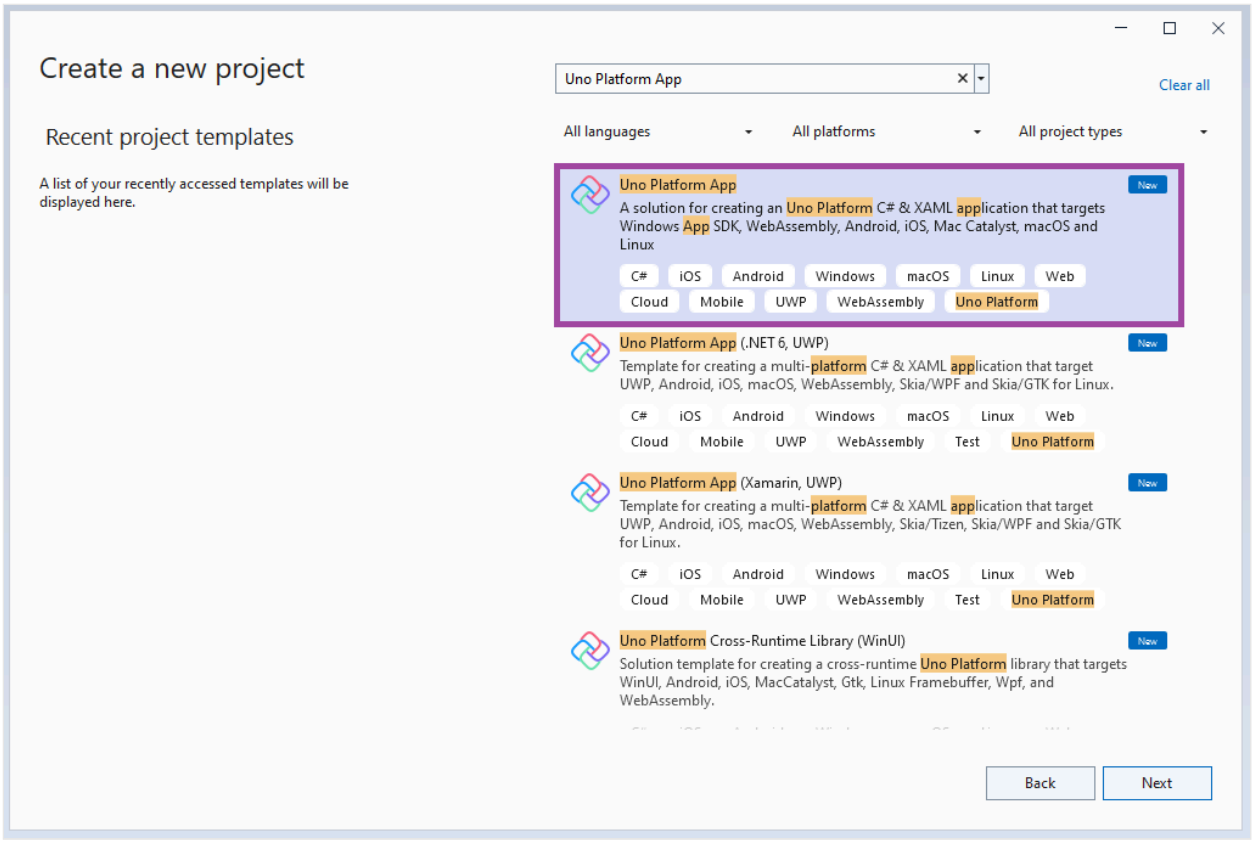
Now that we are ready to create a multi-platform application, the approach we'll take is to create a new Uno Platform application. We will copy XAML code from the previous tutorial's Hello World WinUI 3 project into our multi-platform project. This is possible because Uno Platform lets you reuse your existing codebase. For features dependent on OS APIs provided by each platform, you can easily make them work over time. This approach is especially useful if you have an existing application that you want to port to other platforms.

Soon enough, you will be able to reap the benefits of this approach, as you can target more platforms with a familiar XAML flavor and the codebase you already have.

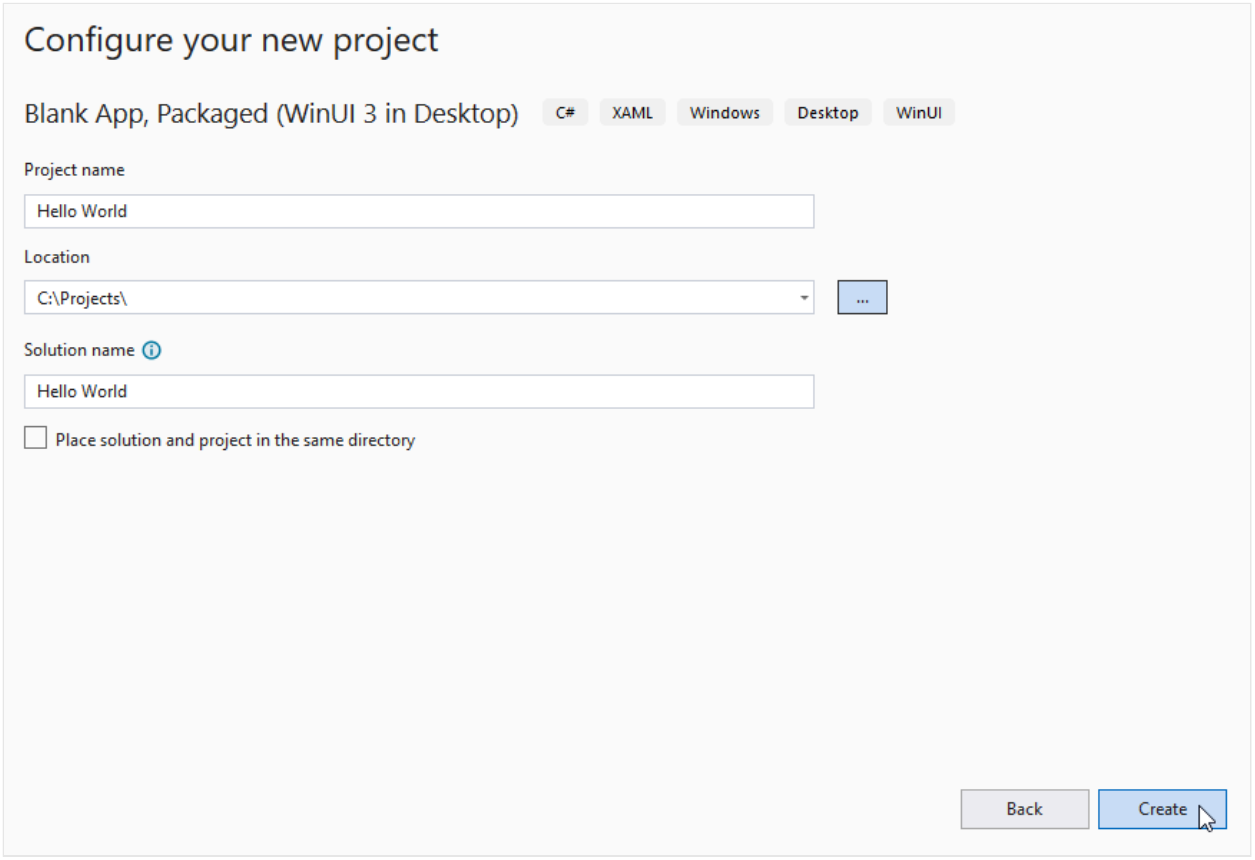
Open Visual Studio and create a new project via `File` > `New` > `Project`:



Search for Uno and select the Uno Platform App project template:

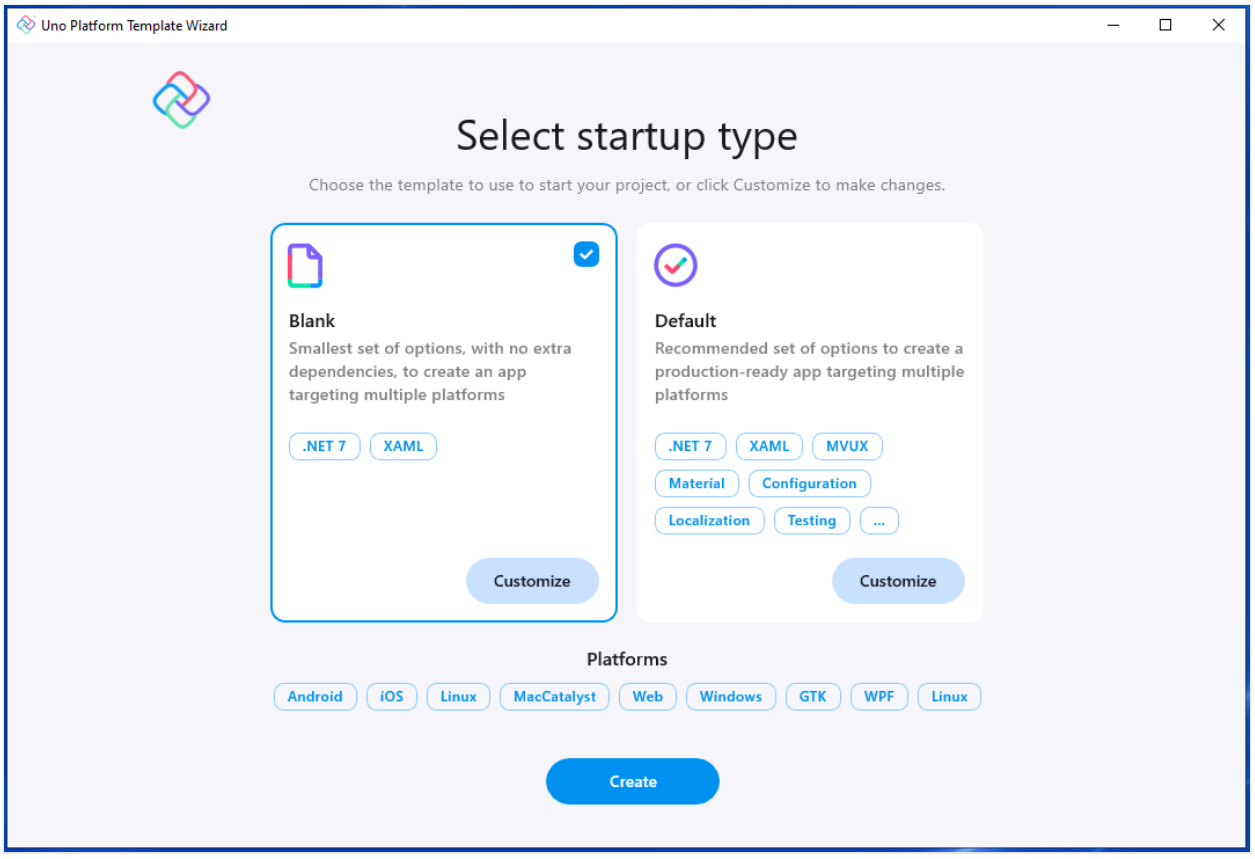


Specify a project name, solution name, and directory. In this example, our Hello World MultiPlatform project belongs to a Hello World MultiPlatform solution, which will live in C:\Projects:



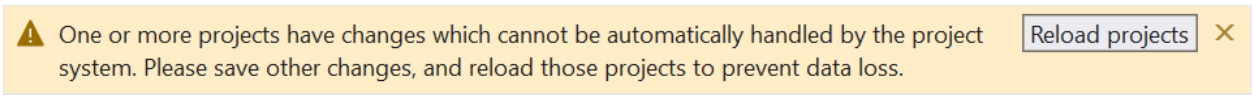
Create a new C# solution using the **Uno Platform App** type from Visual Studio's **Start Page**. To avoid conflicting with the code from the previous tutorial, we'll give this solution a different name, "Hello World Uno".

Now you'll choose a base template to take your Hello World application multi-platform. The Uno Platform App template comes with two preset options that allow you to quickly get started with either a Blank solution or the Default configuration which includes references to the Uno.Material and Uno.Toolkit libraries. The Default configuration also includes Uno.Extensions which is used for dependency injection, configuration, navigation, and logging, and it uses MVUX in place of MVVM, making it a great starting point for rapidly building real-world applications.



To keep things simple, select the **Blank** preset. Then, click the **Create** button. Wait for the projects to be created and their dependencies to be restored.

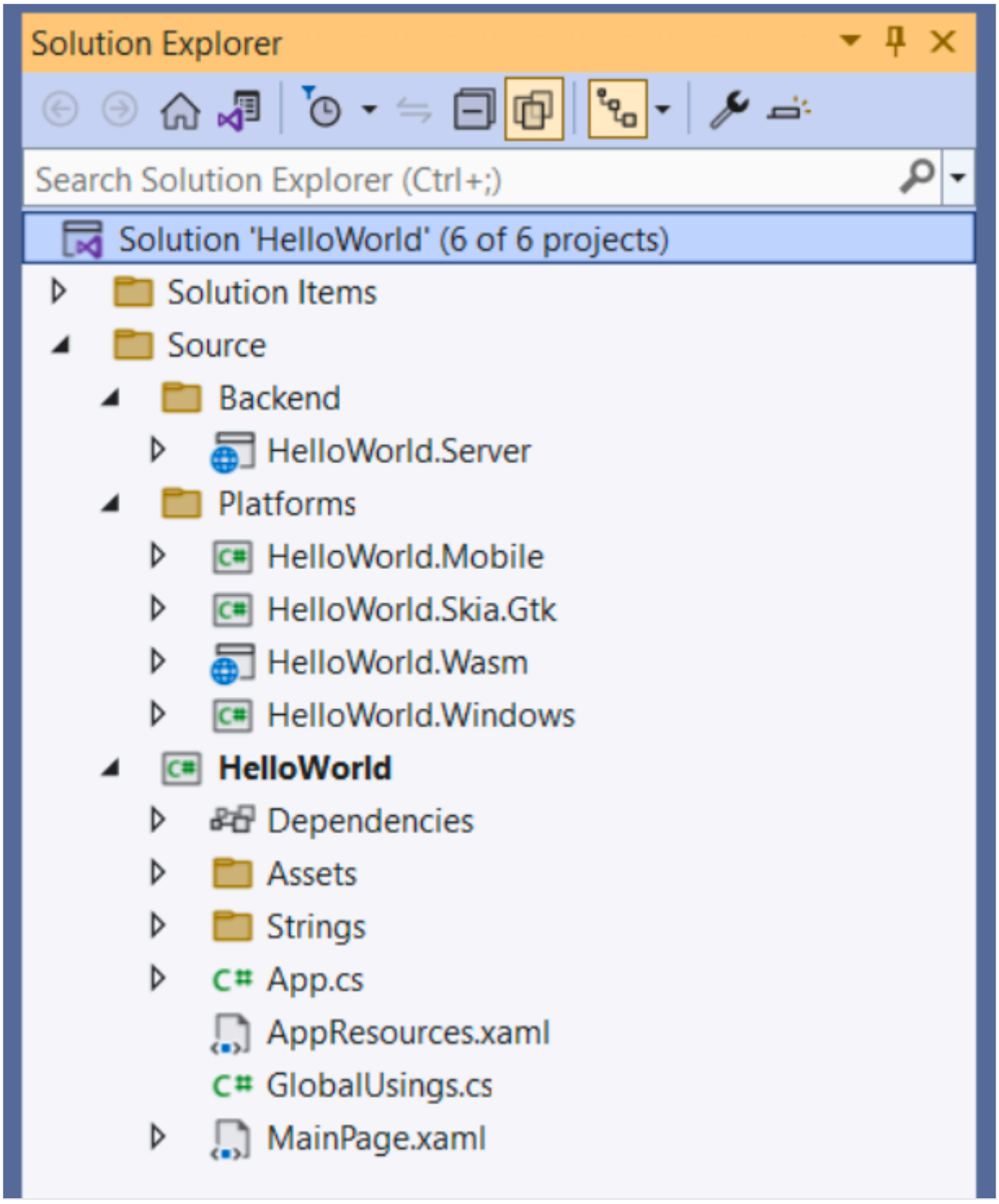
A banner at the top of the editor may ask to reload projects, click **Reload projects**:



Building your app

Now that you've generated the functional starting point of your multi-platform WinUI application, you can copy markup into it from the Hello World WinUI 3 project outlined in the previous tutorial.

You should see the following default file structure in your Solution Explorer:



Make sure Visual Studio has your WinUI 3 project open, then copy the child XAML elements from `MainWindow.xaml` in the WinUI 3 project to your `MainPage.xaml` file in the Uno Platform project. The `MainPage` view XAML should look like this:

XML

```
<Page x:Class="HelloWorld.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:local="using:HelloWorld"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

        <!-- Below is the code you copied from MainWindow: -->
        <StackPanel Orientation="Horizontal"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center">
            <TextBlock x:Name="myText"
                       Text="Hello world!"
                       Foreground="Red"/>
        </StackPanel>
</Page>
```

Launch the HelloWorld.Windows target. Observe that this WinUI app is identical to the previous tutorial.

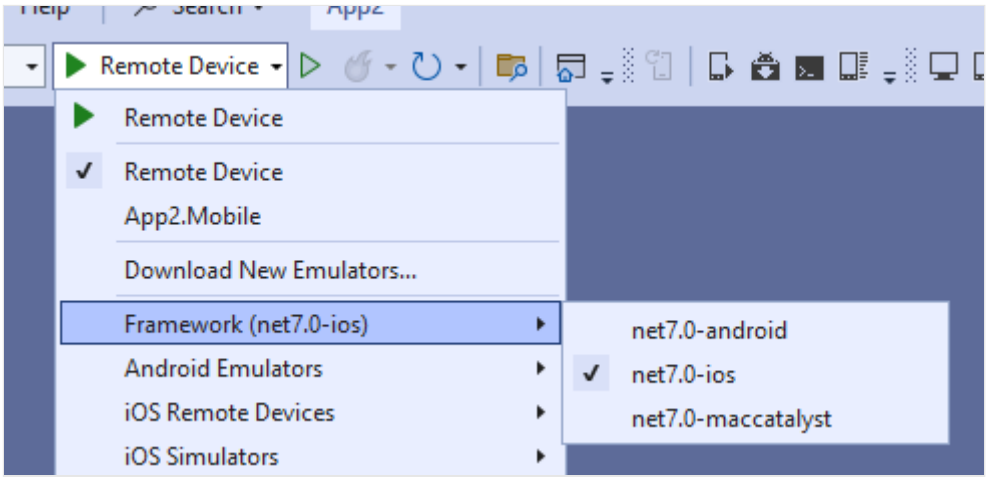
You can now build and run your app on any of the supported platforms. To do so, you can use the debug toolbar drop-down to select a target platform to deploy:

- To run the **WebAssembly** (Wasm) head:
  - Right-click on the `HelloWorld.Wasm` project, select **Set as startup project**
  - Press the `HelloWorld.Wasm` button to deploy the app
  - If desired, you can use the `HelloWorld.Server` project as an alternative

- To debug for **iOS**:

  - Right-click on the `HelloWorld.Mobile` project, select **Set as startup project**

  - In the debug toolbar drop-down, select an active iOS device or the simulator. You'll need to be paired with a Mac for this to work.



- To debug for **Mac Catalyst**:
  - Right-click on the `HelloWorld.Mobile` project, select **Set as startup project**

    - In the debug toolbar drop-down, select a remote macOS device. You'll need to be paired with one for this to work.

- To debug the **Android** platform:
    - Right-click on the `HelloWorld.Mobile` project, select **Set as startup project**
    - In the debug toolbar drop-down, select either an active Android device or the emulator
        - Select an active device in the "Device" sub-menu

- To debug on **Linux** with **Skia GTK**:
    - Right-click on the `HelloWorld.Skia.Gtk` project, and select **Set as startup project**
    - Press the `HelloWorld.Skia.Gtk` button to deploy the app

Now you're ready to start building your multi-platform application!

See also

- Uno Platform documentation
- Uno Extensions features

Create your first WinUI 3 (Windows App SDK) project

Article • 11/21/2024

In this topic we'll see how to use Visual Studio to create a new Windows App SDK project for a C# .NET or C++ app that has a WinUI 3 user interface (UI). We'll also take a look at some of the code in the resulting project, what it does, and how it works.

Links to full installation details are in the steps below. We recommend that you install and target the latest Stable release of the Windows App SDK (see Stable channel release notes).

> **Tip**
>
> No matter what version of the Windows App SDK you choose to install and target (or what version of Visual Studio you use), it's important to check any *limitations and known issues* in the release notes for that version (see **Windows App SDK release channels**). By knowing about any *limitations and known issues* for your version of the Windows App SDK, you'll be able to work around them should you run into any of them while following along with the steps in this topic.
>
> If you encounter any other issues, then you'll likely find info about them in GitHub issues, or on the **Discussions tab**, of the **WindowsAppSDK** GitHub repo; or via an online search.

> **Important**
>
> If you're working on a UWP app, then refer to **Migrate from UWP to the Windows App SDK**.

Packaged, unpackaged, and packaged with external location

Packaging is an important consideration of any Windows App SDK project. For more info about your packaging options, see Advantages and disadvantages of packaging your app.

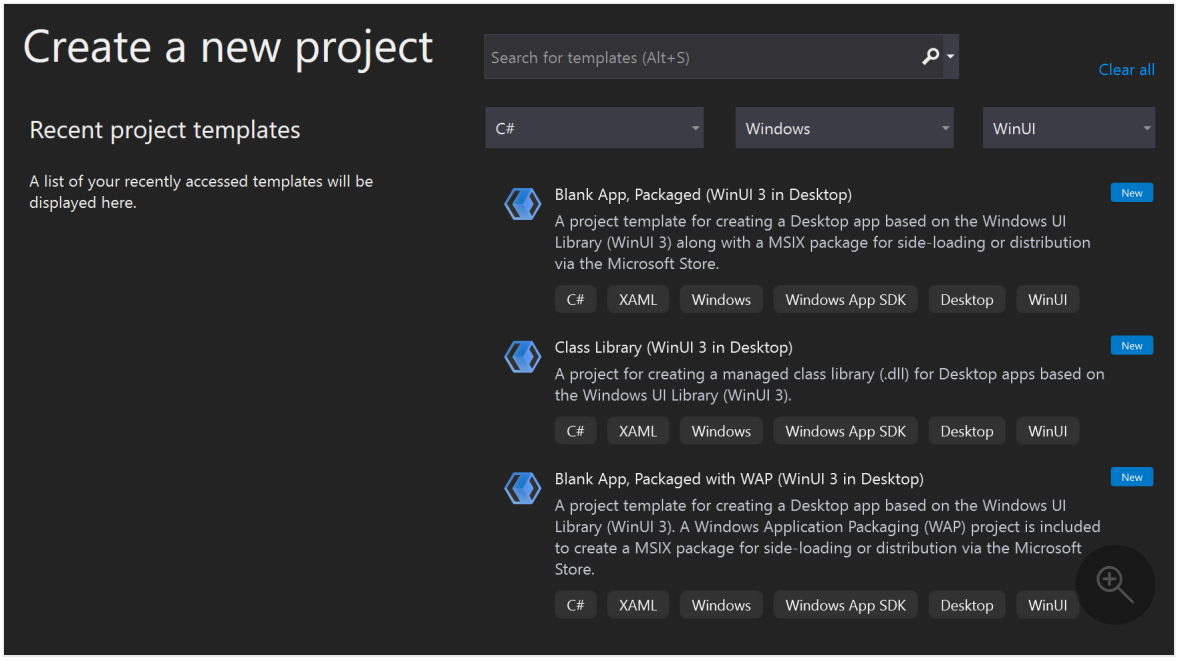
Packaged: Create a new project for a packaged C# or C++ WinUI 3 desktop app

1. To set up your development computer, see Install tools for the Windows App SDK.

2. In Visual Studio, select **File** > **New** > **Project**.

3. In the **New Project** dialog's drop-down filters, select **C#** or **C++**, **Windows**, and **WinUI**, respectively.

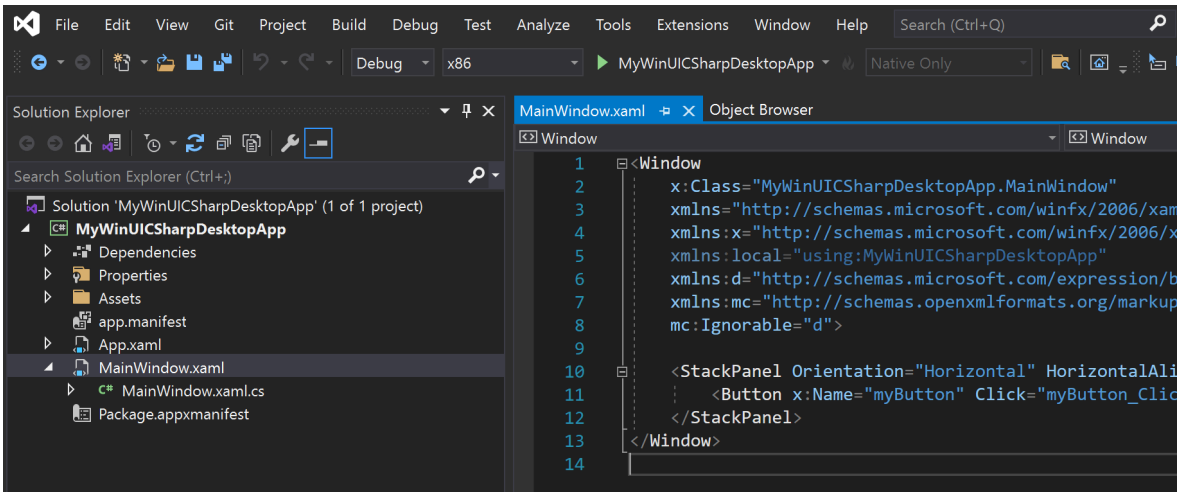
4. Select the **Blank App, Packaged (WinUI 3 in Desktop)** project template, and click **Next**. That template creates a desktop app with a WinUI 3-based user interface. The generated project is configured with the package manifest and other support needed to build the app into an MSIX package (see What is MSIX?). For more information about this project template, see Package your app using single-project MSIX.



5. Enter a project name, choose any other options as desired, and click **Create**.

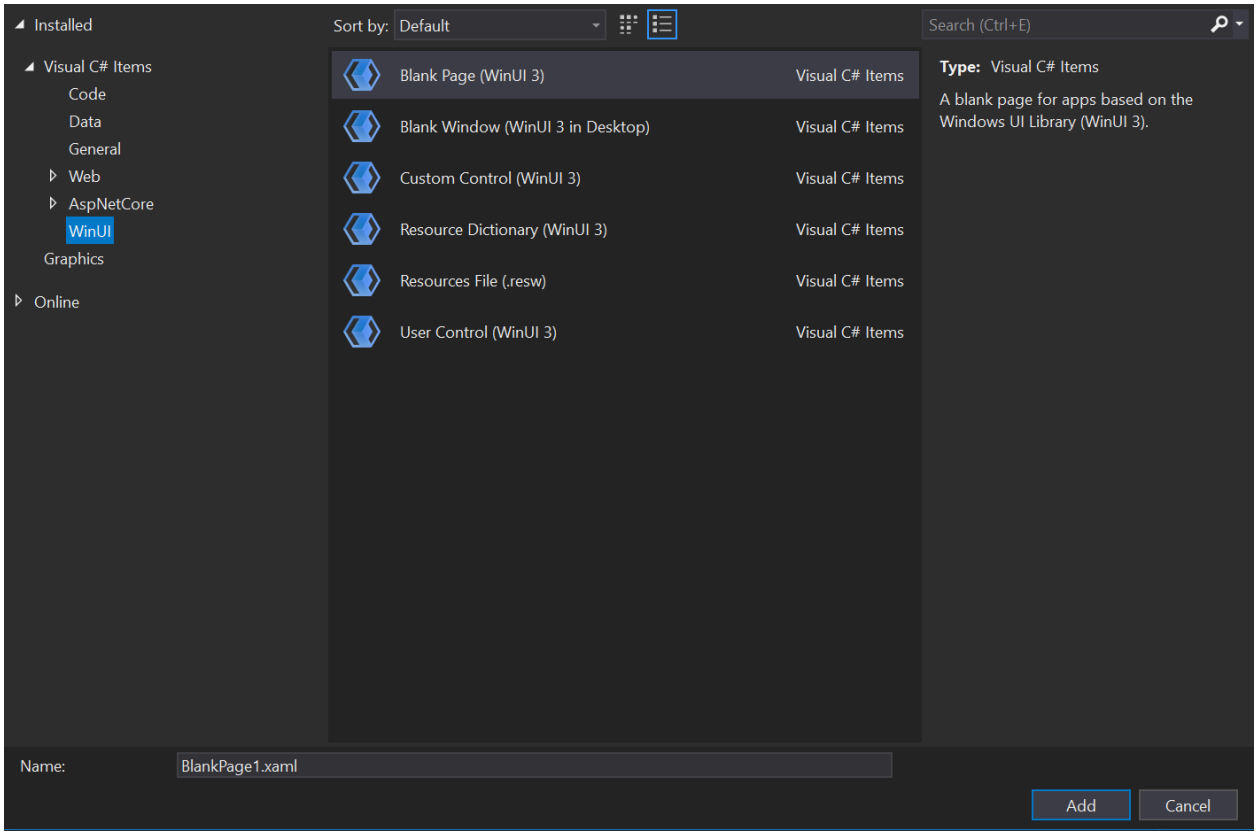
   The project that Visual Studio generates contains your app's code. The **App.xaml** file and code-behind file(s) define an **Application**-derived class that represents your running app. The **MainWindow.xaml** file and code-behind file(s) define a **MainWindow** class that represents the main window displayed by your app. Those classes derive from types in the **Microsoft.UI.Xaml** namespace provided by WinUI 3.

   The project also includes the package manifest for building the app into an MSIX package.



6. Press F5 or use the **Debug** > **Start Debugging** menu option to build and run your solution on your development computer and confirm that the app runs without errors.

The Blank App template creates a simple app with a single window for your content. Apps are typically more complex, and you'll want to add elements as your app progresses, like new windows, pages, or custom user controls. For more details about the available items, and instructions for how to add them, see WinUI 3 templates in Visual Studio.



Unpackaged: Create a new project for an unpackaged C# or C++ WinUI 3 desktop app

> **ⓘ Important**
>
> Beginning in the Windows App SDK 1.0, the default approach to loading the Windows App SDK from an app packaged with external location or an unpackaged app is to use *auto-initialization* via the `<WindowsPackageType>` project property (as well as making additional configuration changes). For the steps involved in auto-initialization in the context of WinUI 3 project, continue reading this section. Or, if you have an existing project that's not WinUI 3, then see **Use the Windows App SDK in an existing project**.

1. To set up your development computer, see Install tools for the Windows App SDK.
2. Download and run the latest *installer* for the Windows App SDK from Downloads for the Windows App SDK. That will install the runtime package dependencies required to run and deploy an app packaged with external location or an unpackaged app on the target device (see Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged).
3. **C++**. Install the Microsoft Visual C++ Redistributable (VCRedist) appropriate for the architecture of the target device.

   - The latest version of the VCRedist is compatible with the latest Visual Studio generally-available (GA) release (that is, not preview), as well as all versions of Visual Studio that can be used to build Windows App SDK binaries.
   - Insider builds of Visual Studio might have installed a later version of the VCRedist, and running the public version will then fail with this error (which you can ignore): **Error 0x80070666: Cannot install a product when a newer version is installed.**

   > **ⓘ Note**
   >
   > If you don't have the VCRedist installed on the target device, then dynamic links to `c:\windows\system32\vcruntime140.dll` fail. That failure can manifest to end users in various ways.

4. In Visual Studio, select **File** > **New** > **Project**.
5. In the New Project dialog's drop-down filters, select **C#** or **C++**, **Windows**, and **WinUI**, respectively.

6. You need to start with a packaged project in order to use XAML diagnostics. So select the **Blank App, Packaged (WinUI 3 in Desktop)** project template, and click **Next**.

   > **ⓘ Important**
   >
   > Make sure that the project you just created is targeting the version of the Windows App SDK that you installed with the *installer* in step 2. To do that, in Visual Studio, click **Tools** > **NuGet Package Manager** > **Manage NuGet Packages for Solution...** > **Updates**. And if necessary update the reference to the *Microsoft.WindowsAppSDK* NuGet package. You can see which version is installed on the **Installed** tab.

7. Add the following property to your project file—either your `.csproj` (C#) or `.vcxproj` (C++) file. Put it inside the **PropertyGroup** element that's already there (for C++, the element will have `Label="Globals"`):

   XML

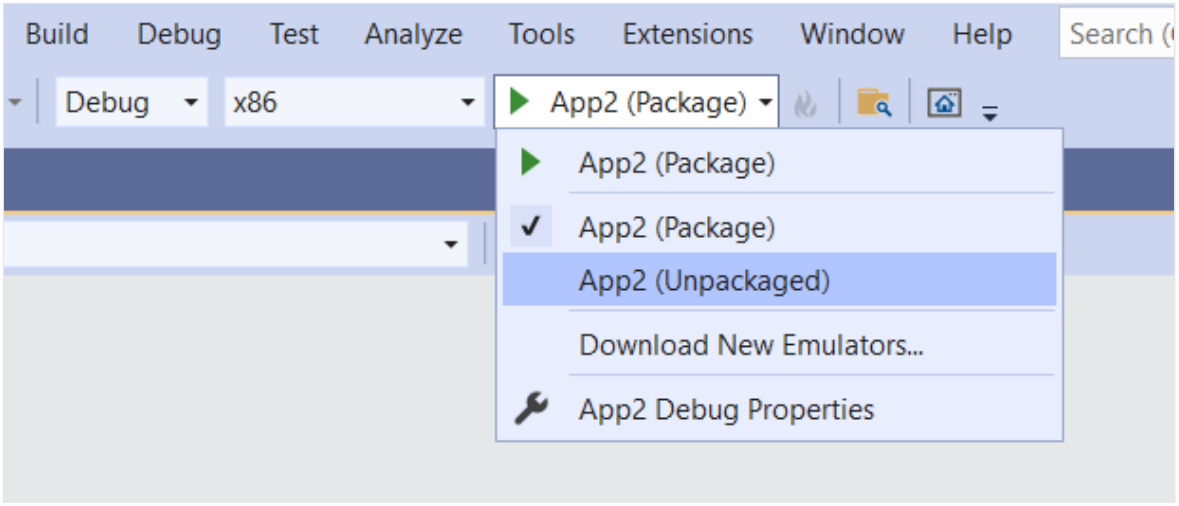
   ```
   <Project ...>
     ...
     <PropertyGroup>
       ...
       <WindowsPackageType>None</WindowsPackageType>
       ...
     </PropertyGroup>
     ...
   </Project>
   ```

8. **C++**. In your C++ project (`.vcxproj`) file, inside the **PropertyGroup** element that's already there, set the *AppxPackage* property to *false*:

   XML

   ```
   <Project ...>
     ...
     <PropertyGroup Label="Globals">
       ...
       <AppxPackage>false</AppxPackage>
       ...
     </PropertyGroup>
     ...
   </Project>
   ```

9. **C#**. To start a C# app from Visual Studio (either **Debugging** or **Without Debugging**), select the *Unpackaged* launch profile from the **Start** drop-down. If the *Package* profile is selected, then you'll see a deployment error in Visual Studio. This step isn't necessary if you start the application (`.exe`) from the command line or from Windows File Explorer.



10. Build and run.

The bootstrapper API

Setting the `<WindowsPackageType>None</WindowsPackageType>` project property causes the *auto-initializer* to locate and load a version of the Windows App SDK version that's most appropriate for your app.

If you have advanced needs (such as custom error handling, or to load a specific version of the Windows App SDK), then you can instead call the bootstrapper API explicitly. For more info, see Use the Windows App SDK runtime for apps packaged with external location or unpackaged, and Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK.

For more info about the bootstrapper, see Deployment architecture and overview for framework-dependent apps.

A look at the code in the project template

In this walkthrough, we used the **Blank App, Packaged (WinUI 3 in Desktop)** project template, which creates a desktop app with a WinUI 3-based user interface. Let's take a look at some of the code that comes with that template, and what it does. For more info on available WinUI 3 project and item templates, see WinUI 3 templates in Visual Studio.

The app's entry point

When the Windows operating system (OS) runs an app, the OS begins execution in the app's *entry point*. That entry point takes the form of a **Main** (or **wWinMain** for C++/WinRT) function. Ordinarily, a new project configures that function to be auto-generated by the Visual Studio build process. And it's hidden by default, so you don't need to be concerned with it. But if you *are* curious for more info, then see Single-instancing in Main or wWinMain.

The App class

The app as a whole is represented by a class that's typically called simply **App**. That class is defined in **App.xaml** and in its code-behind file(s) (`App.xaml.cs`, or `App.xaml.h` and `.cpp`). **App** is derived from the WinUI 3 **Microsoft.UI.Xaml.Application** class.

The generated code in the entry point creates an instance of **App**, and sets it running.

In the constructor of **App**, you'll see the **InitializeComponent** method being called. That method essentially parses the contents of **App.xaml**, which is XAML markup. And that's important because **App.xaml** contains merged resources that need to be resolved and loaded into a dictionary for the running app to use.

Another interesting method of **App** is **OnLaunched**. In there we create and activate a new instance of the **MainWindow** class (which we'll look at next).

The MainWindow class

The main window displayed by the app is of course represented by the **MainWindow** class. That class is defined in **MainWindow.xaml** and in its code-behind file(s) (`MainWindow.xaml.cs`, or `MainWindow.xaml.h` and `.cpp`). **MainWindow** is derived from the WinUI 3 **Microsoft.UI.Xaml.Window** class.

The constructor of **MainWindow** calls its own **InitializeComponent** method. Again, its job is to turn the XAML markup inside **MainWindow.xaml** into a graph of user interface (UI) objects.

In **MainWindow.xaml** you'll see the basic layout of **MainWindow**. At the layout root is a dynamic panel called a **Microsoft.UI.Xaml.Controls.StackPanel**. For more info about layout panels, see Layout panels.

Inside that **StackPanel** is a **Microsoft.UI.Xaml.Controls.Button**. And that **Button** uses the markup `Click="myButton_Click"` to declaratively hook up an event handler method for

the **Click** event.

That method is named **myButton_Click**, and you can find the implementation of that method in `MainWindow.xaml.cs`, or in `MainWindow.xaml.cpp`. In it, the content of the button is changed from the default "Click Me" to "Clicked".

C++. If you created a C++ project, then you'll also see a `MainWindow.idl` file. For more info, see the C++/WinRT documentation. XAML controls; bind to a C++/WinRT property is a good place to start learning about the purpose and usage of `.idl` files.

Next steps

This topic showed how to create a Visual Studio project for a packaged or an unpackaged app. For an example of adding functionality to such an app, see Tutorial: Create a simple photo viewer with WinUI 3. That topic walks through the process of building a simple app to display photos.

Then, to continue your development journey with the Windows App SDK, see Develop Windows desktop apps.

Related topics

- WinUI 3
- Windows App SDK release channels
- Install tools for the Windows App SDK
- What is MSIX?
- Package your app using single-project MSIX
- WinUI 3 project templates in Visual Studio
- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Microsoft Visual C++ Redistributable (VCRedist)
- Use the Windows App SDK runtime for apps packaged with external location or unpackaged
- Deployment architecture for the Windows App SDK
- Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK
- Develop Windows desktop apps

Create a WinUI 3 app using Preview and Experimental channels of the Windows App SDK

Article • 10/24/2022

The Windows App SDK includes WinUI 3 project templates that enable you to create desktop apps with an entirely WinUI-based user interface. When you create apps using these project templates, the entire user interface of your application is implemented using windows, controls, and other UI types provided by WinUI 3. For a complete list of the project templates, see WinUI 3 templates in Visual Studio.

Using the Windows App SDK Stable version: To build a WinUI 3 app using the stable version of the Windows App SDK, see Create your first WinUI 3 project.

Prerequisites

To use the WinUI 3 project templates described in this article, configure your development computer and install the Windows App SDK extension for Visual Studio. For details, see Install tools for preview and experimental channels of the Windows App SDK.

> **Note**
>
> Certain fundamental WinRT types including CoreWindow, ApplicationView, CoreApplicationView CoreDispatcher, and their dependencies are not available in desktop apps. These types were designed specifically for UI scenarios in UWP apps, and they do not behave properly in desktop apps due to threading models and other platform differences. For more information including recommended alternative APIs, see Windows Runtime APIs not supported in desktop apps.

Instructions for WinUI 3 packaged desktop apps

To create a WinUI 3 application with MSIX packaging, choose from one of the following sets of instructions depending on the project language and the version of the Windows App SDK you have installed.

C# with 1.0 Preview 3

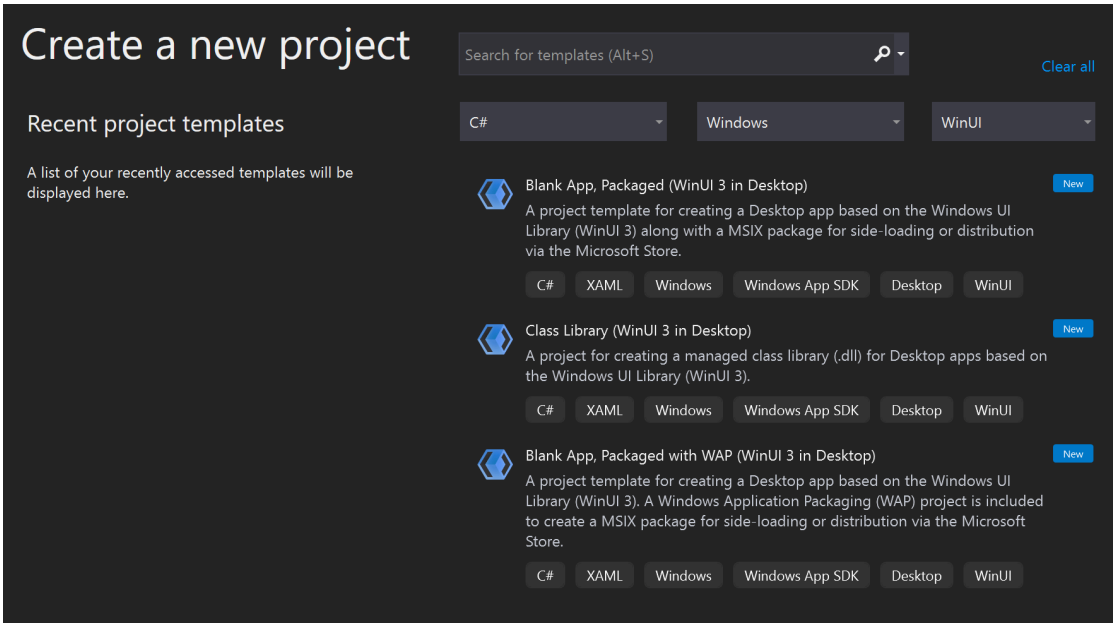
To create a WinUI 3 desktop app with C# using Windows App SDK 1.0 Preview 3:

1. In Visual Studio, select **File** -> **New** -> **Project**.
2. In the project drop-down filters, select **C#**, **Windows**, and **WinUI**, respectively.
3. Select one of the following project types and click **Next**.

    - **Blank App, Packaged (WinUI 3 in Desktop)**: Creates a desktop C# .NET app with a WinUI-based user interface. The generated project is configured with the package manifest and other support needed to build the app into an MSIX package without the use of a separate packaging project. For more information about this project type, see Package your app using single-project MSIX.

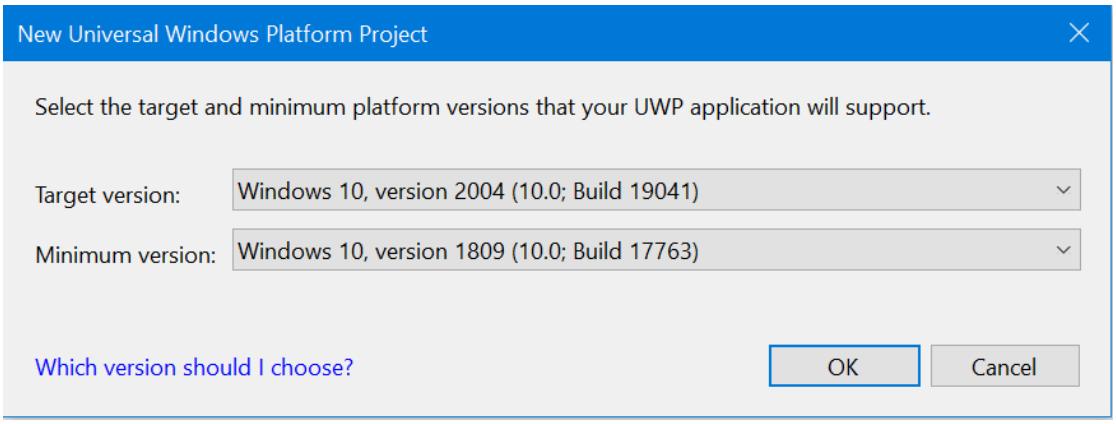
        > **Note**
        >
        > If you installed the Windows App SDK 1.0 Preview 2 with Visual Studio 2019, this project template has a known issue that results in a build error. To resolve this issue, install the Single-project MSIX Packaging Tools for Visual Studio 2019 VSIX extension after you install the Windows App SDK 1.0 Preview 2.

    - **Blank App, Packaged with WAP (WinUI 3 in Desktop)**: Creates a desktop C# .NET app with a WinUI-based user interface. The generated solution includes a separate Windows Application Packaging Project that is configured to build the app into an MSIX package.



4. Enter a project name, choose any other options as desired, and click **Create**.

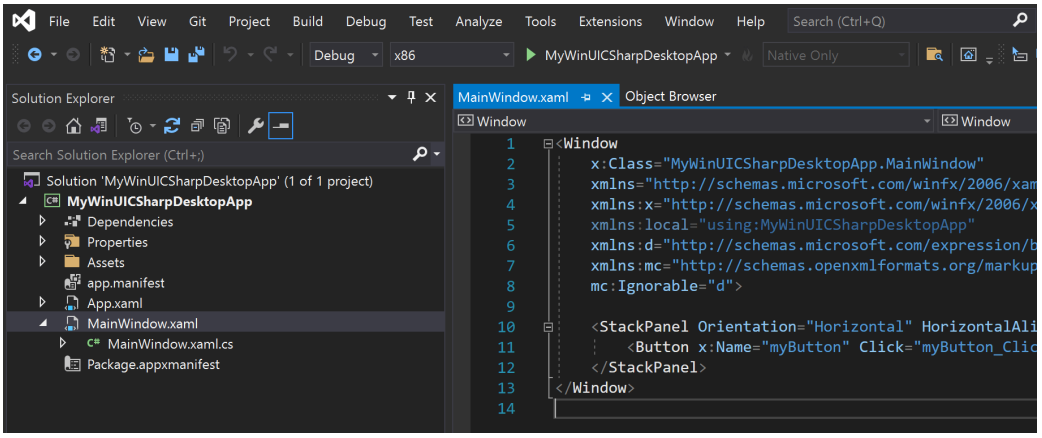
5. In the following dialog box, set the **Target version** to Windows 10, version 2004 (build 19041) and **Minimum version** to Windows 10, version 1809 (build 17763) and then click **OK**.



6. At this point, Visual Studio generates one or more projects:

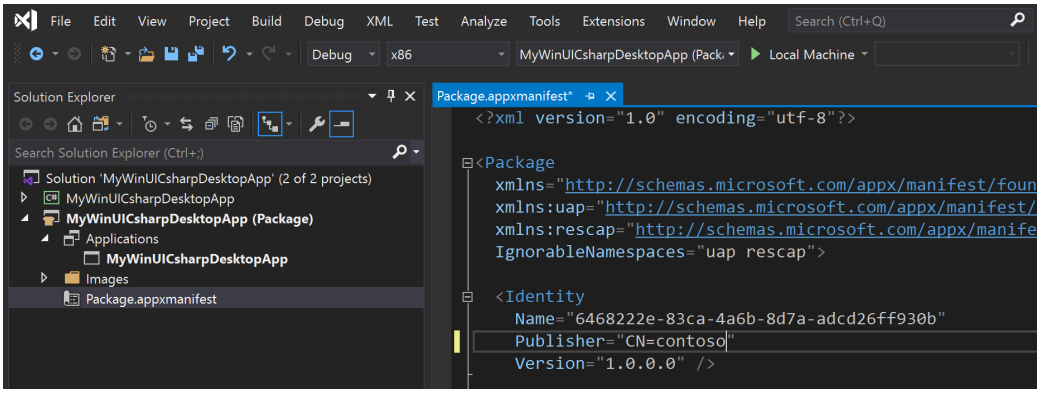
   - ***Project name*** **(Desktop)**: This project contains your app's code. The **App.xaml** file and **App.xaml.cs** code-behind file define an `Application` class that represents your app instance. The **MainWindow.xaml** file and **MainWindow.xaml.cs** code-behind file define a `MainWindow` class that represents the main window displayed by your app. These classes derive from types in the **Microsoft.UI.Xaml** namespace provided by WinUI.

     If you used the **Blank App, Packaged (WinUI 3 in Desktop)** project template, this project also includes the package manifest for building the app into an MSIX package.



   - ***Project name*** **(Package)**: This project is generated only if you use the **Blank App, Packaged with WAP (WinUI 3 in Desktop)** project template. This project is a Windows Application Packaging Project that is configured to build the app into an MSIX package. This project contains

the package manifest for your app, and it is the startup project for your solution by default.

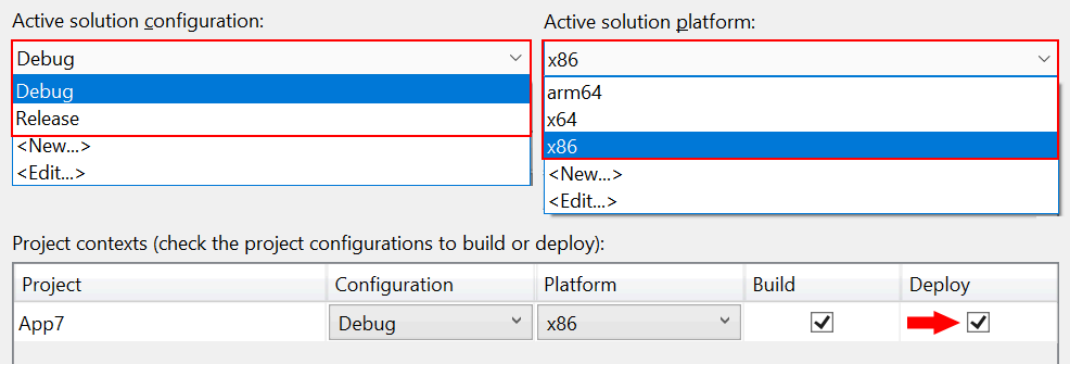


7. Enable deployment for your project in **Configuration Manager**. If you do not follow these steps to enable deployment, you will encounter the following error when you try to run or debug your project on your development computer: "The project needs to be deployed before we can debug. Please enable Deploy in the Configuration Manager".

   a. Select **Build** -> **Configuration Manager**.

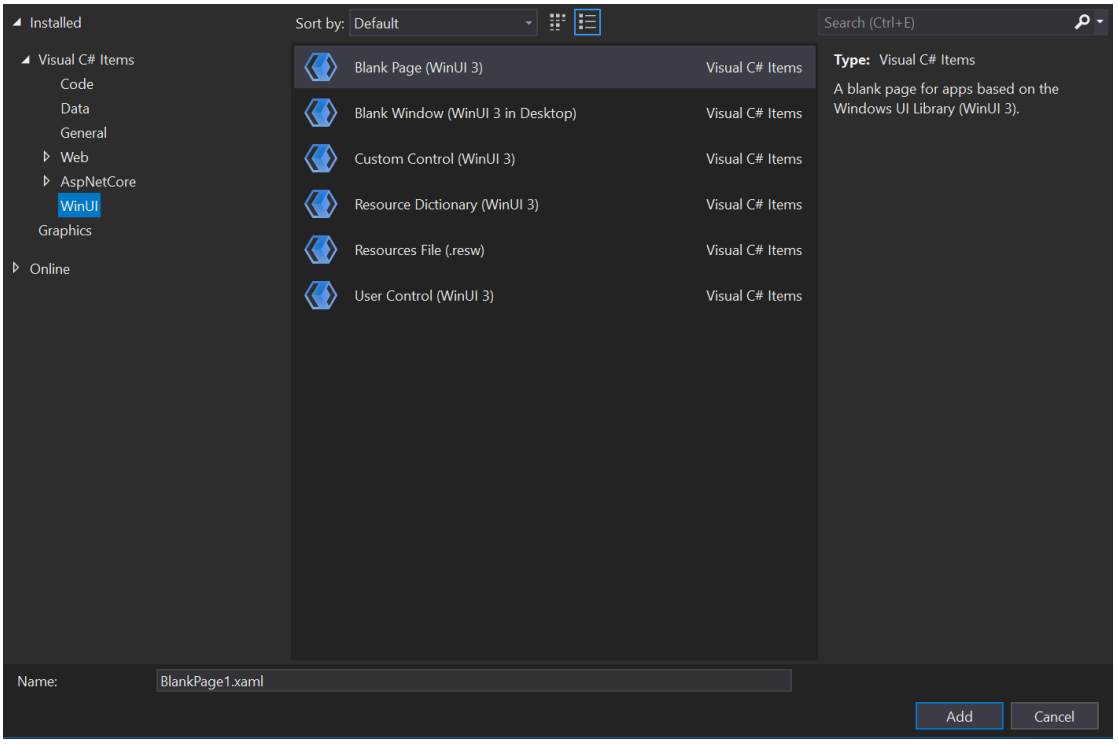
   b. In **Configuration Manager**, click the **Deploy** check box for every combination of configuration and platform (for example, **Debug** and **x86**, **Debug** and **arm64**, **Release** and **x64**, and more).

   > **Note**
   >
   > Be sure to use the **Active solution configuration** and **Active solution platform** drop-downs at the top instead of the **Configuration** and **Platform** drop-downs in the same row as the **Deploy** check box.



8. To add a new item to your app project, right-click the ***Project name*** **(Desktop)** project node in **Solution Explorer** and select **Add** -> **New Item**. In the **Add New Item** dialog box, select the **WinUI** tab, choose the item you want to add,

and then click **Add**. For more details about the available items, see WinUI 3 templates in Visual Studio.



9. Build and run your solution on your development computer to confirm that the app runs without errors.

Localize your WinUI desktop app

To support multiple languages in a WinUI desktop app, and ensure proper localization of your packaged project, add the appropriate resources to the project (see App resources and the Resource Management System) and declare each supported language in the **package.appxmanifest** file of your project. When you build the project, the specified languages are added to the generated app manifest (**AppxManifest.xml**) and the corresponding resources are used.

1. Open the .wapproj's `package.appxmanifest` in a text editor and locate the following section:

```xml
<Resources>
    <Resource Language="x-generate"/>
</Resources>
```

2. Replace the `<Resource Language="x-generate">` with `<Resource />` elements for each of your supported languages. For example, the following markup specifies that "en-US" and "es-ES" localized resources are available:

XML

```
<Resources>
    <Resource Language="en-US"/>
    <Resource Language="es-ES"/>
</Resources>
```

Instructions for WinUI 3 unpackaged desktop apps

> **Note**
>
> WinUI 3 support for unpackaged apps is currently in **preview**, and is not production-ready. You must download the Windows App SDK Preview Extension for Visual Studio to get the project templates and build unpackaged desktop apps with WinUI 3. For a list of the **known issues**, see Windows App SDK 1.0 Preview 3 release notes.

To create a WinUI 3 application without MSIX packaging, choose from one of the following sets of instructions depending on the project language and the version of the Windows App SDK you have installed.

C# with 1.0 Preview 3

To create a WinUI 3 desktop app with C# using Windows App SDK 1.0 Preview 3:

1. Install the Single-project MSIX Packaging Tools.
2. Install the Visual Studio 2019 C# extension or Visual Studio 2022 C# extension, depending on your version of Visual Studio.
3. Install the Windows App SDK runtime and MSIX packages. These are required to run and deploy your app.
4. Create a new app using the "Blank App, Packaged (WinUI 3 in Desktop)" project template. Starting with a packaged app is required to use XAML diagnostics.
5. Add the following property to your project file—either your `.csproj` (C#) or `.vcxproj` (C++) file:

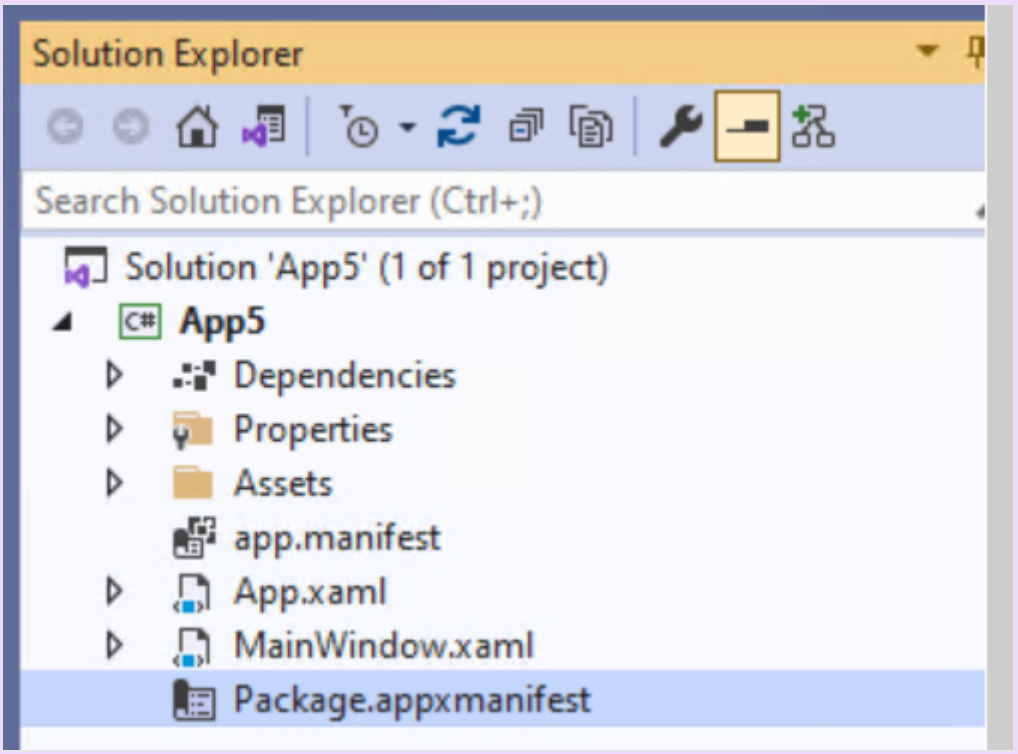
XML

```
<Project ...>
  ...
  <PropertyGroup>
    ...
    <WindowsPackageType>None</WindowsPackageType>
  </PropertyGroup>
  ...
</Project>
```

6. Delete package.appxmanifest from project.

   Otherwise, you will see this error: **Improper project configuration: WindowsPackageType is set to None, but AppxManifest is specified**.

   > ⓘ Note
   >
   > You may need to close the Visual Studio solution to manually delete this file from the filesystem.



7. To debug in Visual Studio, change the debug properties from 'MsixPackage' to 'Project'. Otherwise, you'll see an error: "The project doesn't know how to run the profile ..."

   > ⓘ Note

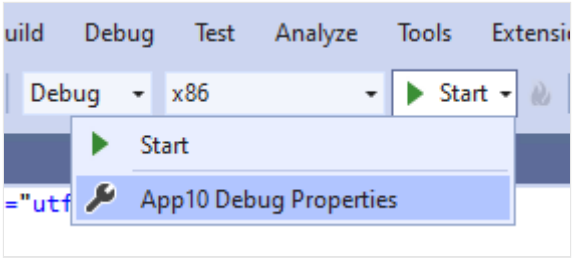
This isn't necessary if you execute the application (`.exe`) from the command line or Windows File explorer.

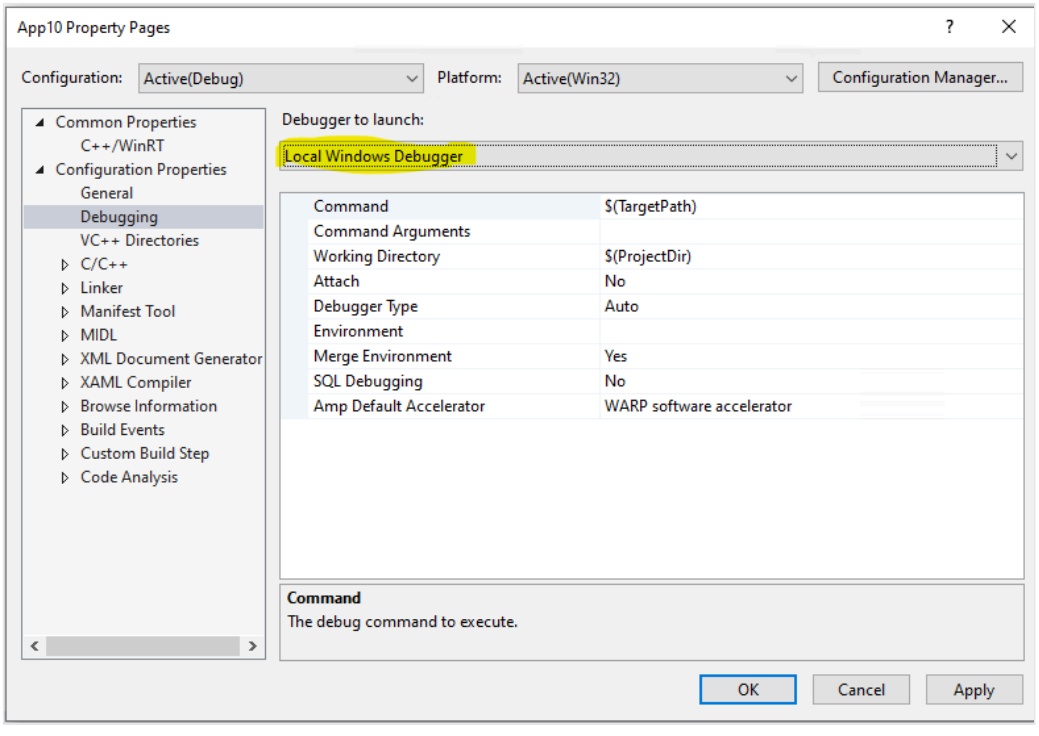
- In **Visual Studio 2022**: Open the launchSettings.json and change the profile with 'MsixPackage' to 'Project'.

```JSON
{
    "profiles": {
        "Preview3": {
            "commandName": "Project"
        }
    }
}
```

- In **Visual Studio 2019 and Visual Studio 2022**: You can use the Visual Studio UI to change the launch settings:

  Open the Debug Properties and change the launch profile to 'Project'





8. If you haven't already done so, **install the Windows App SDK runtime and MSIX packages, which are required to run and deploy your app.**

   Download latest installer & MSIX packages

9. Build and run. For additional deployment information, see the Windows App SDK tutorial for deploying packaged with external location and unpackaged apps (Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK). That tutorial will guide you through using the bootstrapper API (see Use the Windows App SDK runtime for apps packaged with external location or unpackaged) to initialize the Windows App SDK runtime so your app can use Windows App SDK and WinUI 3 APIs.

ASTA to STA threading model

If you're migrating code from an existing UWP app to a new WinUI 3 project that uses the Windows App SDK, be aware that the new project uses the single-threaded apartment (STA) threading model instead of the Application STA (ASTA) threading model used by UWP apps. If your code assumes the non re-entrant behavior of the ASTA threading model, your code may not behave as expected.

See also

- Windows App SDK
- Windows App SDK Samples

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Tutorial: Build and deploy an unpackaged app using Preview and Experimental channels of the Windows App SDK

Article • 06/14/2023

Using the Windows App SDK Stable version: You can auto-initialize the Windows App SDK through the `WindowsPackageType` project property when you Create your first WinUI 3 project. You can also follow a tutorial (Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK) in which you configure a packaged with external location or unpackaged app to load the Windows App SDK runtime, and call Windows App SDK APIs.

This article provides a step-by-step tutorial for configuring a packaged with external location or unpackaged app so that it can load the Windows App SDK runtime and call Windows App SDK APIs. This tutorial demonstrates this scenario using a basic Console app project, but the steps apply to any unpackaged desktop app that uses the Windows App SDK.

Before completing this tutorial, we recommend that you review Runtime architecture to learn more about the Framework package dependency your app takes when it uses Reunion, and the additional components required to work in an unpackaged app.

> ⓘ **Note**
>
> The dynamic dependencies and bootstrapper APIs fail when called by an elevated process. As a result, Visual Studio should not be launched elevated. See issue for more details.

Prerequisites

1. Install tools for the Windows App SDK.
2. Ensure that all dependencies for the app are installed (see Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged). The simplest solution is to run the Windows App SDK runtime installer.

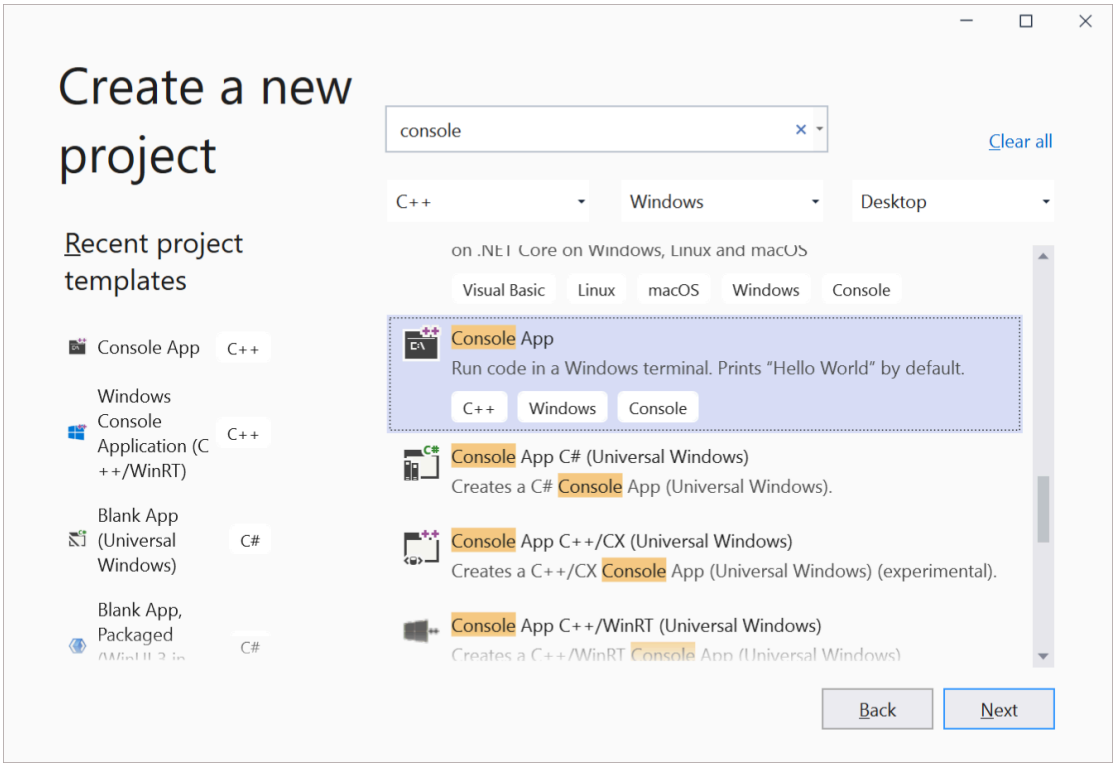
Instructions

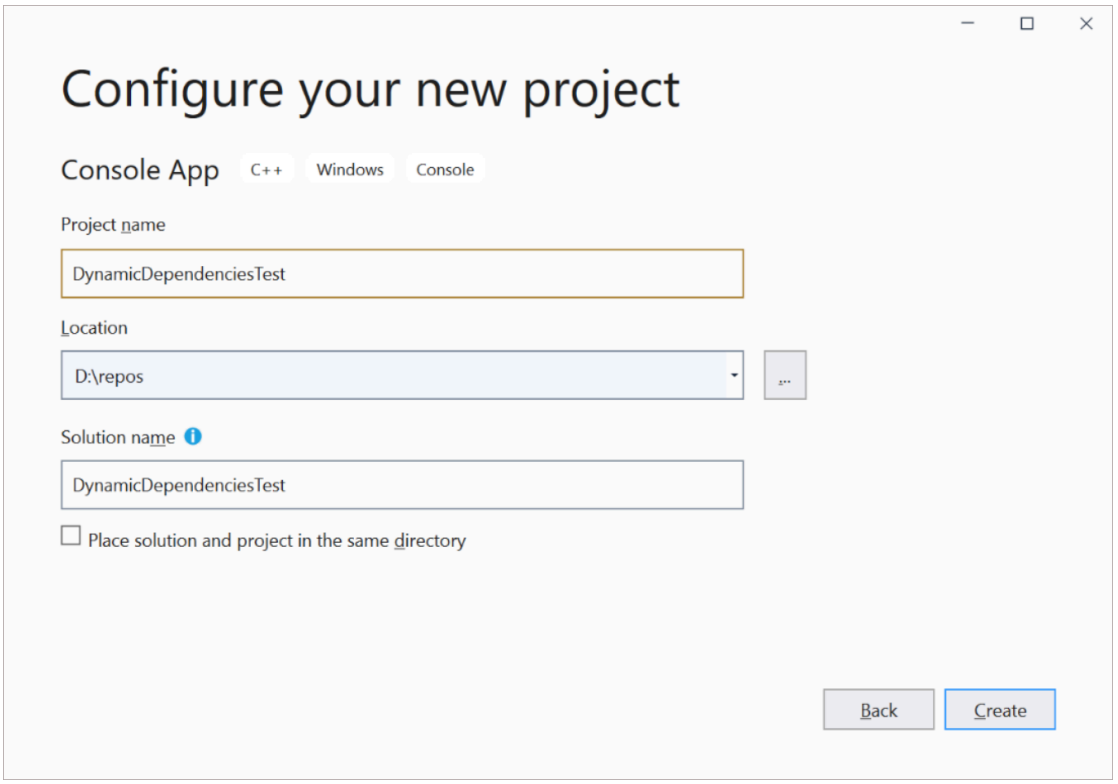
You can choose to follow this tutorial using a C++ project or a C# project.

C++

Follow these instructions to configure a C++ project. Starting in 1.0 Preview 3, you can also configure a C++ project that includes WinUI 3 unpackaged support.

1. In Visual Studio, create a new C++ **Console App** project. Name the project **DynamicDependenciesTest**.





After you create the project, you should have a 'Hello World' C++ console app.

2. Next, install the Windows App SDK NuGet package in your project.
   a. In **Solution Explorer**, right-click the **References** node and choose **Manage Nuget Packages**.
   b. In the **NuGet Package Manager** window, select the **Include prerelease** check box near the top of the window, select the **Browse** tab, and install one of the following packages:

      - To install 1.0 Preview 3 or 1.0 Experimental, search for **Microsoft.WindowsAppSDK**.
      - To install 0.8 Preview, search for **Microsoft.ProjectReunion**.

3. For unpackaged apps, you will need to call the Bootstrapper API to Use the Windows App SDK runtime for apps packaged with external location or unpackaged. This enables you to use Windows App SDK APIs at runtime.

   a. Add the following include files to the top of your **DynamicDependenciesTest.cpp** file. The mddbootstrap.h header is available via the Windows App SDK NuGet package.

```
C++
```

```
#include <windows.h>
#include <MddBootstrap.h>
```

b. Next, add this code at the beginning of your `main` method to call the MddBootstrapInitialize function to initialize the Bootstrapper and handle any errors. This code defines what version of the Windows App SDK the app is dependent upon when initializing the Bootstrapper.

```
C++


// The following code is for 1.0 Preview 3. If using 1.0
Experimental,
// replace with versionTag{ L"experimental1" }. If using version
0.8 Preview,
// replace with majorMinorVersion{ 0x00000008 } and versionTag{
L"preview" }.
const UINT32 majorMinorVersion{ 0x00010000 };
PCWSTR versionTag{ L"preview3" };
const PACKAGE_VERSION minVersion{};

const HRESULT hr{ MddBootstrapInitialize(majorMinorVersion,
versionTag, minVersion) };

// Check the return code for errors. If there is an error,
display the result.
if (FAILED(hr))
{
    wprintf(L"Error 0x%X in MddBootstrapInitialize(0x%08X, %s,
%hu.%hu.%hu.%hu)\n",
        hr, majorMinorVersion, versionTag, minVersion.Major,
minVersion.Minor, minVersion.Build, minVersion.Revision);
    return hr;
}
```

c. Finally, add this code to display the string `Hello World!` and call the MddBootstrapShutdown function to uninitialize the Bootstrapper.

```
C++

std::cout << "Hello World!\n";

// Release the DDLM and clean up.
MddBootstrapShutdown();
```

d. Your final code should look like this.

```
C++

#include <iostream>
#include <windows.h>
#include <MddBootstrap.h>
```

```
int main()
{

    // Take a dependency on Windows App SDK 1.0 Preview 3. If
using 1.0 Experimental,
    // replace with versionTag{ L"experimental1" }. If using
version 0.8 Preview,
    // replace with majorMinorVersion{ 0x00000008 } and
versionTag{ L"preview" }.
    const UINT32 majorMinorVersion{ 0x00010000 };
    PCWSTR versionTag{ L"preview3" };
    const PACKAGE_VERSION minVersion{};

    const HRESULT hr{ MddBootstrapInitialize(majorMinorVersion,
versionTag, minVersion) };

    // Check the return code. If there is a failure, display the
result.
    if (FAILED(hr))
    {
        wprintf(L"Error 0x%X in MddBootstrapInitialize(0x%08X,
%s, %hu.%hu.%hu.%hu)\n",
            hr, majorMinorVersion, versionTag, minVersion.Major,
minVersion.Minor, minVersion.Build, minVersion.Revision);
        return hr;
    }

    std::cout << "Hello World!\n";

    // Release the DDLM and clean up.
    MddBootstrapShutdown();
}
```

4. Press F5 to build and run your app.

Related topics

- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Runtime architecture

Tutorial: Create a simple photo viewer with WinUI 3

Article • 08/19/2024

> **Note**
>
> For info about the benefits of WinUI 3, as well as other app type options, see Overview of framework options.

In this topic we walk through the process of creating a new WinUI 3 project in Visual Studio; and then building a simple app to display photos. We'll use controls, layout panels, and data-binding. And we'll be writing both XAML markup (which is *declarative*) and your choice of either C# or C++ code (which are *imperative*, or *procedural*). Use the language picker above the topic title to choose C# or C++/WinRT.

> **Tip**
>
> The source code in this topic is provided in both C# and C++/WinRT. If you're a C++ developer, then for more details and concepts explaining how the code that we show here works, see the C++/WinRT documentation. Relevant topics there include XAML controls; bind to a C++/WinRT property, XAML items controls; bind to a C++/WinRT collection, and Photo Editor C++/WinRT sample application.

Step 1: Install tools for the Windows App SDK

To set up your development computer, see Get started with WinUI. In that article, you'll also find instructions to create and launch a WinUI 3 project.

> **Important**
>
> You'll find release notes topics along with the Windows App SDK release channels topic. There are release notes for each channel. Be sure to check any *limitations and known issues* in those release notes, since those might affect the results of following along with this tutorial and/or running the app we'll build.

Step 2: Create a new project

In Visual Studio, create your choice of a new C# or C++ project from the **Blank App, Packaged (WinUI 3 in Desktop)** project template. Name the project *SimplePhotos*, and (so that your folder structure will match the one described in this tutorial) uncheck **Place solution and project in the same directory**. You can target the most recent release (not preview) of the client operating system.

Step 3: Copy asset files

The app that we'll be building carries image files around with it in the form of asset files; and those are the photos that it displays. In this section you'll add those assets to your project. But first you'll need to obtain a copy of the files.

1. So clone (or download as a `.zip`) the Windows App SDK samples repo (see WindowsAppSDK-Samples). Having done that, you'll find the asset files that we'll be using in the folder `\WindowsAppSDK-Samples\Samples\PhotoEditor\cs-winui\Assets\Samples` (use this folder for both a C# and a C++/WinRT project). If you want to see those files in the repo online, then you can visit WindowsAppSDK-Samples/Samples/PhotoEditor/cs-winui/Assets/Samples/.

2. In **File Explorer**, select that **Samples** folder, and copy it to the clipboard.

C#

3. Go to **Solution Explorer** in Visual Studio. Right-click the **Assets** folder (it's a child of the project node), and click **Open Folder in File Explorer**. That opens the **Assets** folder in **File Explorer**.

4. Paste (into the **Assets** folder) the **Samples** folder that you just copied.

Step 4: Add a GridView control

Our app needs to display rows and columns of photos. In other words, a grid of images. For a UI like that, the main controls to use are List view and grid view.

1. Open `MainWindow.xaml`. Currently, there's a **Window** element, and within that a **StackPanel** layout panel. Inside the **StackPanel** is a **Button** control, which is hooked up to an event handler method.

The main window of any app represents the view that you see first when you run the app. In the app we'll be building, the main window's job is to load the photos from the **Samples** folder, and to display a tiled view of those images together with various info about them.

2. Replace the **StackPanel** and **Button** markup with the **Grid** layout panel and the **GridView** control shown in the listing below.

```
XAML
```

```
<Window ...>
    <Grid>
        <GridView x:Name="ImageGridView"/>
    </Grid>
</Window>
```

> **Tip**
>
> `x:Name` identifies a XAML element so that you can refer to it elsewhere in the XAML and in the code-behind.

3. **C#**. Open `MainWindow.xaml.cs`, and delete the **myButton_Click** method.

4. **C++/WinRT**. Open `MainWindow.xaml.h` and `MainWindow.xaml.cpp`, and delete the **myButton_Click** method.

You *can* build and run now, but the window will be empty at this stage. For the **GridView** control to show anything, we need to give it a collection of objects to show. We'll make a start on that next.

For background info about some of the types we've just mentioned, see Layout panels and Controls for Windows apps.

Step 5: The ImageFileInfo model

A *model* (in the sense of models, views, and view models) is a class that to some degree represents a real-world object or concept (such as a bank account). It's an *abstraction* of that real-world thing. In this section we'll be adding to our project a new class called **ImageFileInfo**. **ImageFileInfo** will be a model of an image file, such as a photo. This section will take us a step closer to being able to display photos in the app's user interface (UI).

> **Tip**
>
> In preparation for the code example below, let's introduce the term *observable*. A property that can be dynamically bound to a XAML control (so that the UI updates each time the property value changes) is known as an *observable* property. This idea is based on the software design pattern known as the observer pattern. In the app that we build in this tutorial, the properties of our **ImageFileInfo** model won't change. But even so, we'll show how to make **ImageFileInfo** observable, by having it implement the INotifyPropertyChanged interface.

C#

1. Right-click the project node (**SimplePhotos**), and click **Add** > **New Item...**. Under **C# Items** > **Code**, select **Class**. Set the name to *ImageFileInfo.cs*, and click **Add**.

2. Replace the contents of `ImageFileInfo.cs` with the code listing below.

C#

```
using Microsoft.UI.Xaml.Media.Imaging;
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using System.Threading.Tasks;
using Windows.Storage;
using Windows.Storage.FileProperties;
using Windows.Storage.Streams;

namespace SimplePhotos
{
    public class ImageFileInfo : INotifyPropertyChanged
    {
        public ImageFileInfo(ImageProperties properties,
            StorageFile imageFile,
            string name,
            string type)
        {
            ImageProperties = properties;
            ImageName = name;
            ImageFileType = type;
            ImageFile = imageFile;
            var rating = (int)properties.Rating;
            var random = new Random();
            ImageRating = rating == 0 ? random.Next(1, 5) : rating;
        }
```

```
        public StorageFile ImageFile { get; }

        public ImageProperties ImageProperties { get; }

        public async Task<BitmapImage> GetImageSourceAsync()
        {
            using IRandomAccessStream fileStream = await
ImageFile.OpenReadAsync();

            // Create a bitmap to be the image source.
            BitmapImage bitmapImage = new();
            bitmapImage.SetSource(fileStream);

            return bitmapImage;
        }

        public async Task<BitmapImage> GetImageThumbnailAsync()
        {
            StorageItemThumbnail thumbnail =
                await
ImageFile.GetThumbnailAsync(ThumbnailMode.PicturesView);
            // Create a bitmap to be the image source.
            var bitmapImage = new BitmapImage();
            bitmapImage.SetSource(thumbnail);
            thumbnail.Dispose();

            return bitmapImage;
        }

        public string ImageName { get; }

        public string ImageFileType { get; }

        public string ImageDimensions => $"{ImageProperties.Width}
x {ImageProperties.Height}";

        public string ImageTitle
        {
            get => string.IsNullOrEmpty(ImageProperties.Title) ?
ImageName : ImageProperties.Title;
            set
            {
                if (ImageProperties.Title != value)
                {
                    ImageProperties.Title = value;
                    _ = ImageProperties.SavePropertiesAsync();
                    OnPropertyChanged();
                }
            }
        }

        public int ImageRating
        {
            get => (int)ImageProperties.Rating;
            set
```

```
            {
                if (ImageProperties.Rating != value)
                {
                    ImageProperties.Rating = (uint)value;
                    _ = ImageProperties.SavePropertiesAsync();
                    OnPropertyChanged();
                }
            }
        }

        public event PropertyChangedEventHandler PropertyChanged;

        protected void OnPropertyChanged([CallerMemberName] string
propertyName = null) =>
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
    }
}
```

3. Save and close the `ImageFileInfo.cs` file.

Step 6: Define and populate a property for a collection of images

In this section we'll add a new property to the **MainWindow** class. The property (named **Images**) will be a collection class that contains the images that we want to display.

C#

1. Define the property like this, in `MainWindow.xaml.cs`:

```
C#

...
using System.Collections.ObjectModel;
...
namespace SimplePhotos
{
    public sealed partial class MainWindow : Window
    {
        public ObservableCollection<ImageFileInfo> Images { get; }
=
            new ObservableCollection<ImageFileInfo>();
        ...
```

```
    }
}
```

2. The code to populate the new collection property with images is shown in the **GetItemsAsync** and **LoadImageInfoAsync** methods below. Paste the `using` directives, and the two method implementations, into `MainWindow.xaml.cs`, too. These methods are members of the **MainWindow** class, so paste them inside there just as you did with the **Images** property above.

C#

```
...
using System.Threading.Tasks;
using Windows.ApplicationModel;
using Windows.Storage;
using Windows.Storage.Search;
...
private async Task GetItemsAsync()
{
    StorageFolder appInstalledFolder =
Package.Current.InstalledLocation;
    StorageFolder picturesFolder = await
appInstalledFolder.GetFolderAsync("Assets\\Samples");

    var result = picturesFolder.CreateFileQueryWithOptions(new
QueryOptions());

    IReadOnlyList<StorageFile> imageFiles = await
result.GetFilesAsync();
    foreach (StorageFile file in imageFiles)
    {
        Images.Add(await LoadImageInfoAsync(file));
    }

    ImageGridView.ItemsSource = Images;
}

public async static Task<ImageFileInfo>
LoadImageInfoAsync(StorageFile file)
{
    var properties = await
file.Properties.GetImagePropertiesAsync();
    ImageFileInfo info = new(properties,
                             file, file.DisplayName,
file.DisplayType);

    return info;
}
```

3. The last thing we need to do in this section is to update the constructor of **MainWindow** to call **GetItemsAsync**.

```
C#

public MainWindow()
{
    ...
    GetItemsAsync();
}
```

You can build and run now if you like (to confirm you've followed the steps ok), but there's not much to see in the window at this stage. That's because what we've done so far is to ask the **GridView** to render a collection of objects of type **ImageFileInfo**; and the **GridView** doesn't quite know how to do that yet.

Remember that the **Images** property is an observable collection of **ImageFileInfo** objects. And the last line of **GetItemsAsync** tells the **GridView** (which is named *ImageGridView*) that the source of its items (**ItemsSource**) is the **Images** property. And the job of the **GridView** is then to display those items.

But we haven't yet told the **GridView** anything about the **ImageFileInfo** class. So the best it can do so far is to display the **ToString** value of each **ImageFileInfo** object in the collection. And by default, that's just the type's name. In the next section we'll create a data template to define how we want an **ImageFileInfo** object to be displayed.

> **Tip**
>
> I used the term *observable collection* above. In the app that we build in this tutorial, the number of images doesn't change (and as we've said, neither do the values of the properties of each image, either). But it's still convenient, and a good practice, to use data-binding to initially connect the UI to the data. So that's what we'll do.

Step 7: Add a data template

To begin with, let's use a sketch-like placeholder data template. That will serve until we're done exploring some layout options. After that we can update the data template to show the actual photos.

> **Tip**

That's actually a very practical thing to do. It's been found that if a UI looks like a sketch (in other words, low-fidelity), then people are more willing to suggest and/or test out quick ideas with it—sometimes involving quite substantial changes. That's because we guess (correctly) that such changes will be cheap to try.

On the other hand, the more finished a UI looks (the higher fidelity it has), the more we guess (again, correctly) that a lot of work has gone into the current look of it. And that makes us less inclined to suggest, or to try out, new ideas.

1. Open `MainWindow.xaml`, and change the contents of the **Window** so that they look like this markup:

   XAML

   ```
   <Window ...>
       <Grid>
           <Grid.Resources>
               <DataTemplate x:Key="ImageGridView_ItemTemplate">
                   <Grid/>
               </DataTemplate>
           </Grid.Resources>
           <GridView x:Name="ImageGridView"
                     ItemTemplate="{StaticResource
   ImageGridView_ItemTemplate}">
           </GridView>
       </Grid>
   </Window>
   ```

   To the layout root, we've added a simple **DataTemplate** resource, and given it a key of `ImageGridView_ItemTemplate`. And we've used that same key to set the **ItemTemplate** of the **GridView**. Items controls such as **GridView** have an **ItemTemplate** property (just like they have the **ItemsSource** property that we saw earlier). An item template is a data template; and it's used for displaying each item in the collection.

   For more info, see Item containers and templates.

2. Now we can take a few edit passes over the data template—adding to, and editing, the elements inside it to make it more interesting and useful. We'll give the root **Grid** a height and a width of 300, and a margin of 8. Then we'll add two row definitions, and set the height of the second row definition to **Auto**.

   XAML

   ```
   <DataTemplate x:Key="ImageGridView_ItemTemplate">
       <Grid Height="300"
   ```

```
          Width="300"
          Margin="8">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
    </Grid>
</DataTemplate>
```

For more info, see Alignment, margin, padding.

3. We want the data template to display each photo's image, name, file type, dimensions, and rating. So we'll be adding, respectively, an **Image** control, some **TextBlock** controls, and a **RatingControl** control. We'll lay out the text inside **StackPanel** layout panels. The **Image** will, initially, display the project's sketch-like Microsoft Store logo as a placeholder.

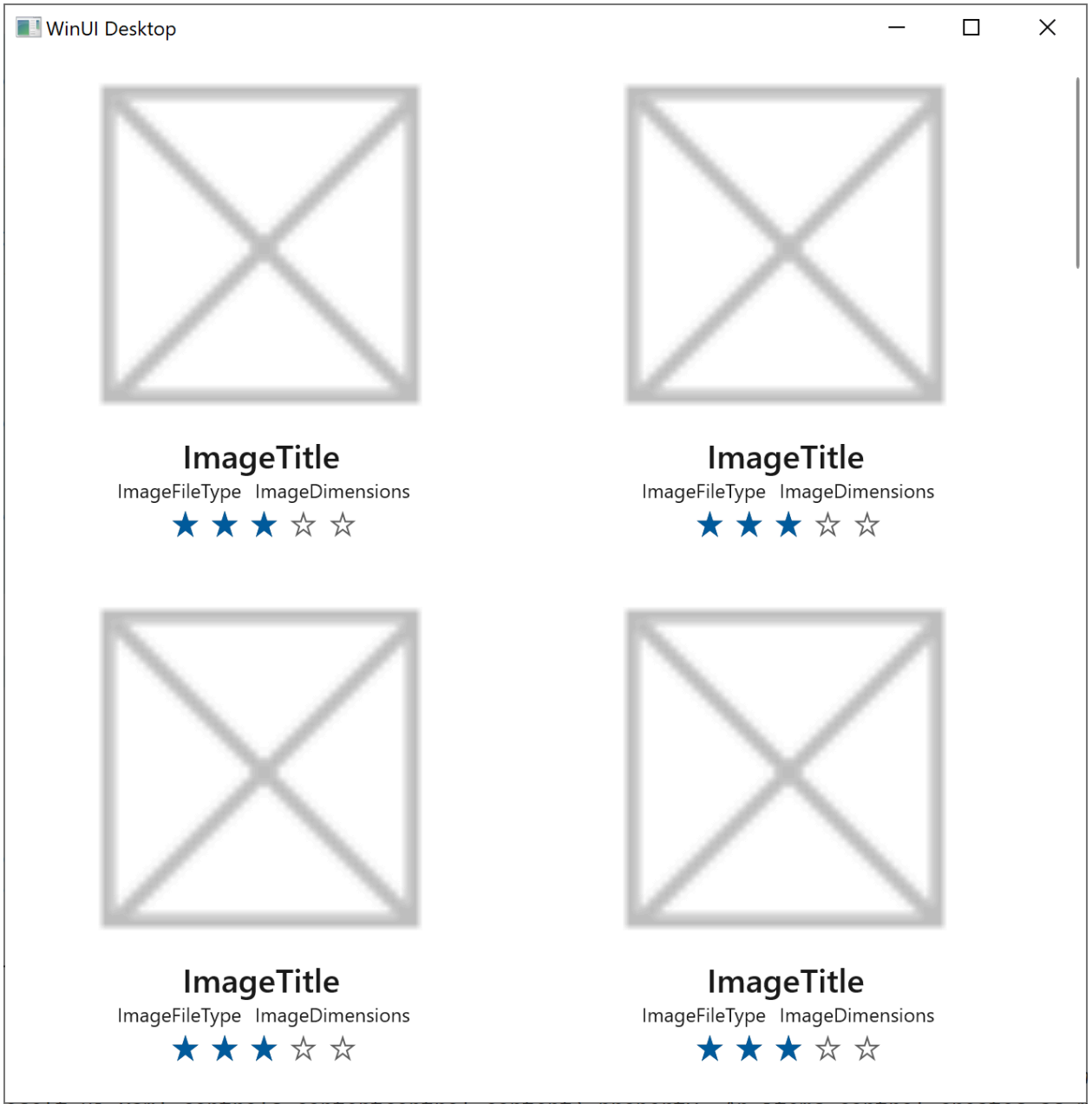
4. After all those edits, here's how the data template looks:

XAML

```
<DataTemplate x:Key="ImageGridView_ItemTemplate">
    <Grid Height="300"
          Width="300"
          Margin="8">
        <Grid.RowDefinitions>
            <RowDefinition />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>

        <Image x:Name="ItemImage"
               Source="Assets/StoreLogo.png"
               Stretch="Uniform" />

        <StackPanel Orientation="Vertical"
                    Grid.Row="1">
            <TextBlock Text="ImageTitle"
                       HorizontalAlignment="Center"
                       Style="{StaticResource SubtitleTextBlockStyle}"
/>
            <StackPanel Orientation="Horizontal"
                        HorizontalAlignment="Center">
                <TextBlock Text="ImageFileType"
                           HorizontalAlignment="Center"
                           Style="{StaticResource
CaptionTextBlockStyle}" />
                <TextBlock Text="ImageDimensions"
                           HorizontalAlignment="Center"
                           Style="{StaticResource
CaptionTextBlockStyle}"
                           Margin="8,0,0,0" />
```

```
            </StackPanel>

            <RatingControl Value="3" IsReadOnly="True"/>
        </StackPanel>
    </Grid>
</DataTemplate>
```

Build the project now, and run the app to see the **GridView** control with the item template that you just created. Next, we'll take a look at how the items are laid out. We'll change some brushes, and we'll add space between the items.



Step 8: Edit the item container's style

Another concept related to items controls such as **GridView** is the *item container*. An item container is a *content control* that displays an item as the value of its Content

property. An items control creates as many item containers as it needs in order to display the items that are visible on the screen at any time.

Being a control, an item container has a style and a control template. Its style and control template determine how the item container looks in its various states (such as selection, pointer over, and focus). And, as we've seen, the item template (which is a data template) determines how the item itself looks.

For **GridView**, the type of its item containers is **GridViewItem**.

So in this section we'll focus on designing the item container's style. And for that we'll be creating a **Style** resource for **GridViewItem**, and then setting that as the **ItemContainerStyle** of the ***GridView**. In the style, we'll set the **Background** and **Margin** properties of the item container to give it a gray background and a bit of a margin around the outside of it.

1. In `MainWindow.xaml`, add a new **Style** resource to the same **Grid.Resources** XML element that we put the data template into.

   XAML

   ```
   <Grid>
       <Grid.Resources>
       ...
           <Style x:Key="ImageGridView_ItemContainerStyle"
               TargetType="GridViewItem">
               <Setter Property="Background" Value="Gray"/>
               <Setter Property="Margin" Value="8"/>
           </Style>
       </Grid.Resources>
   ```

2. Next, we use the `ImageGridView_ItemContainerStyle` key to set the **ItemContainerStyle** of the **GridView**.

   XAML

   ```
   <GridView x:Name="ImageGridView"
           ...
           ItemContainerStyle="{StaticResource
   ImageGridView_ItemContainerStyle}">
   </GridView>
   ```

Build and run the app, and see how it looks now. As you resize the window, the **GridView** control takes care of rearranging the items to best fit into the space. At some widths, there's a lot of space on the right-hand side of the app window. It would look better if the **GridView**, and/or its contents, were centered. So we'll take care of that next.

> **Tip**
>
> If you want to experiment, try setting the **Background** and **Margin** setters to different values to see what effect that has.

Step 9: Experimenting with layout

You might be wondering whether it's best to center the **GridView** itself, or to center its contents. Let's try centering the **GridView** first.

1. To make it easy to see exactly where the **GridView** is in the Window—and what happens as we experiment with layout—we'll set its **Background** property to red.

   XAML

   ```
   <GridView x:Name="ImageGridView"
             ...
             Background="Red">
   </GridView>
   ```

2. And now we'll set its **HorizontalAlignment** property to *Center*.

   XAML

   ```
   <GridView x:Name="ImageGridView"
             ...
             HorizontalAlignment="Center">
   </GridView>
   ```

   Also see Alignment, margin, padding.

Build and run now, and experiment with adjusting the width of the window. You can see that there's an equal amount of empty space on either side of the **GridView**'s red background. So we have achieved the goal of centering the images. But it's now clearer than before that the scroll bar belongs to the **GridView**, and not to the window. So we need to change the **GridView** back to filling the window. We've demonstrated that (instead of centering the **GridView** in the window) we need to center the images in the **GridView**.

3. So now delete the **HorizontalAlignment** attribute that you added in the previous step.

Step 10: Edit the items panel template

Items controls lay out their item containers inside what's known as an *items panel*. We can define what kind of panel is used, and set properties on that panel, by editing the **GridView**'s *items panel template*. So that's what we'll do in this section.

1. In `MainWindow.xaml`, add an **ItemsPanelTemplate** resource to our resource dictionary. The items panel is of type **ItemsWrapGrid**, and we're setting its **HorizontalAlignment** property to **Center**.

   XAML

   ```
   <Grid>
       <Grid.Resources>
       ...
           <ItemsPanelTemplate x:Key="ImageGridView_ItemsPanelTemplate">
               <ItemsWrapGrid Orientation="Horizontal"
                              HorizontalAlignment="Center"/>
           </ItemsPanelTemplate>
       </Grid.Resources>
   ```

2. Next, we use the `ImageGridView_ItemsPanelTemplate` key to set the **ItemsPanel** of the **GridView**.

   XAML

   ```
   <GridView x:Name="ImageGridView"
           ...
           ItemsPanel="{StaticResource ImageGridView_ItemsPanelTemplate}">
   </GridView>
   ```

When you build and run this time, and experiment with adjusting the width of the window, there's an equal amount of the **GridView**'s red background on either side of the images. And because the **GridView** fills the window, the scroll bar aligns nicely with the edge of the window, where users might expect it to be.

3. Now that we're done experiment with layout, remove `Background="Red"` from the **GridView**.

Step 11: Replace the placeholder image with a photo

Now it's time to move our sketch to a higher level of fidelity; and that means replacing the placeholder image with the real ones, and replacing the "lorem ipsum"-style

placeholder text with real data. Let's take care of the images first.

> **ⓘ Important**
>
> The technique we'll use to display the photos in the `Assets\Samples` folder involves updating the **GridView**'s items progressively. Specifically, that's the code in the **ImageGridView_ContainerContentChanging** and **ShowImage** methods in the code example below, including use of the **ContainerContentChangingEventArgs.InRecycleQueue** and **ContainerContentChangingEventArgs.Phase** properties. For more info, see ListView and GridView UI optimization. But in a nutshell, the **GridView** will let us know (by way of an event) when one of its item containers is ready to display its item. And then we'll keep track of which phase of its update lifecycle the item container is in so that we can determine when it's ready to display photo data.

C#

1. In `MainWindow.xaml.cs`, add a new method to **MainWindow** named **ImageGridView_ContainerContentChanging**. This is an event-handling method, and the event it handles is ContainerContentChanging. We also need to provide the implementation of the **ShowImage** method that **ImageGridView_ContainerContentChanging** depends on. Paste the `using` directive and the two method implementations into `MainWindow.xaml.cs`:

   C#

   ```
   ...
   using Microsoft.UI.Xaml.Controls;
   ...
   private void ImageGridView_ContainerContentChanging(
       ListViewBase sender,
       ContainerContentChangingEventArgs args)
   {
       if (args.InRecycleQueue)
       {
           var templateRoot = args.ItemContainer.ContentTemplateRoot
   as Grid;
           var image = templateRoot.FindName("ItemImage") as Image;
           image.Source = null;
       }

       if (args.Phase == 0)
       {
           args.RegisterUpdateCallback(ShowImage);
           args.Handled = true;
   ```

```
    }
}

private async void ShowImage(ListViewBase sender,
ContainerContentChangingEventArgs args)
{
    if (args.Phase == 1)
    {
        // It's phase 1, so show this item's image.
        var templateRoot = args.ItemContainer.ContentTemplateRoot
as Grid;
        var image = templateRoot.FindName("ItemImage") as Image;
        var item = args.Item as ImageFileInfo;
        image.Source = await item.GetImageThumbnailAsync();
    }
}
```

2. Then, in `MainWindow.xaml`, register the **ImageGridView_ContainerContentChanging** event-handler with the **GridView**'s **ContainerContentChanging** event.

XAML

```
<GridView x:Name="ImageGridView"
        ...

ContainerContentChanging="ImageGridView_ContainerContentChanging">
</GridView>
```

Step 12: Replace the placeholder text with real data

In this section we'll use one-time data-bindings. A one-time binding is great for data that doesn't change at run-time. And that means that one-time bindings are high-performance and easy to create.

1. In `MainWindow.xaml`, find the *ImageGridView_ItemTemplate* data template resource. We're going to tell the data template that its job is to be a template for the **ImageFileInfo** class, which you'll recall is the type of the items that our **GridView** is displaying.

2. To do that, add an `x:DataType` value to the template, like this:

XAML

```
<DataTemplate x:Key="ImageGridView_ItemTemplate"
              x:DataType="local:ImageFileInfo">
    ...
```

If you're unfamiliar with the `local:` syntax shown above (or with the `xmlns:local` syntax already in the opening **Window** tag), then see XAML namespaces and namespace mapping.

Now that we've set an `x:DataType`, we can use `x:Bind` data-binding expressions in the data template to bind to properties of the data type we specified (**ImageFileInfo**, in this case).

3. In the data template, find the first **TextBlock** element (the one with its **Text** currently set to *ImageTitle*). Replace its **Text** value as shown below.

> **Tip**
>
> You can either copy and paste the markup below, or you can use IntelliSense in Visual Studio. To do that, select the current value that's inside the quotation marks, and type `{`. IntelliSense automatically adds the closing brace, and displays a code-completion list. You *could* scroll down to `x:Bind`, and double-click it. But it might be more efficient to type `x:` (note how `x:Bind` is then filtered to the top of the completion list), and press the TAB key. Now press the SPACE key, and type `ImageT` (as much of the property name `ImageTitle` as necessary to get it to the top of the completion list), and TAB.

XAML

```
<TextBlock Text="{x:Bind ImageTitle}"
    ... />
```

An `x:Bind` expression links the value of a UI property with the value of a **data-object** property. Of course, that depends on first setting `x:DataType` to the type of that **data-object** so that the tooling and the runtime know what properties are available to bind to.

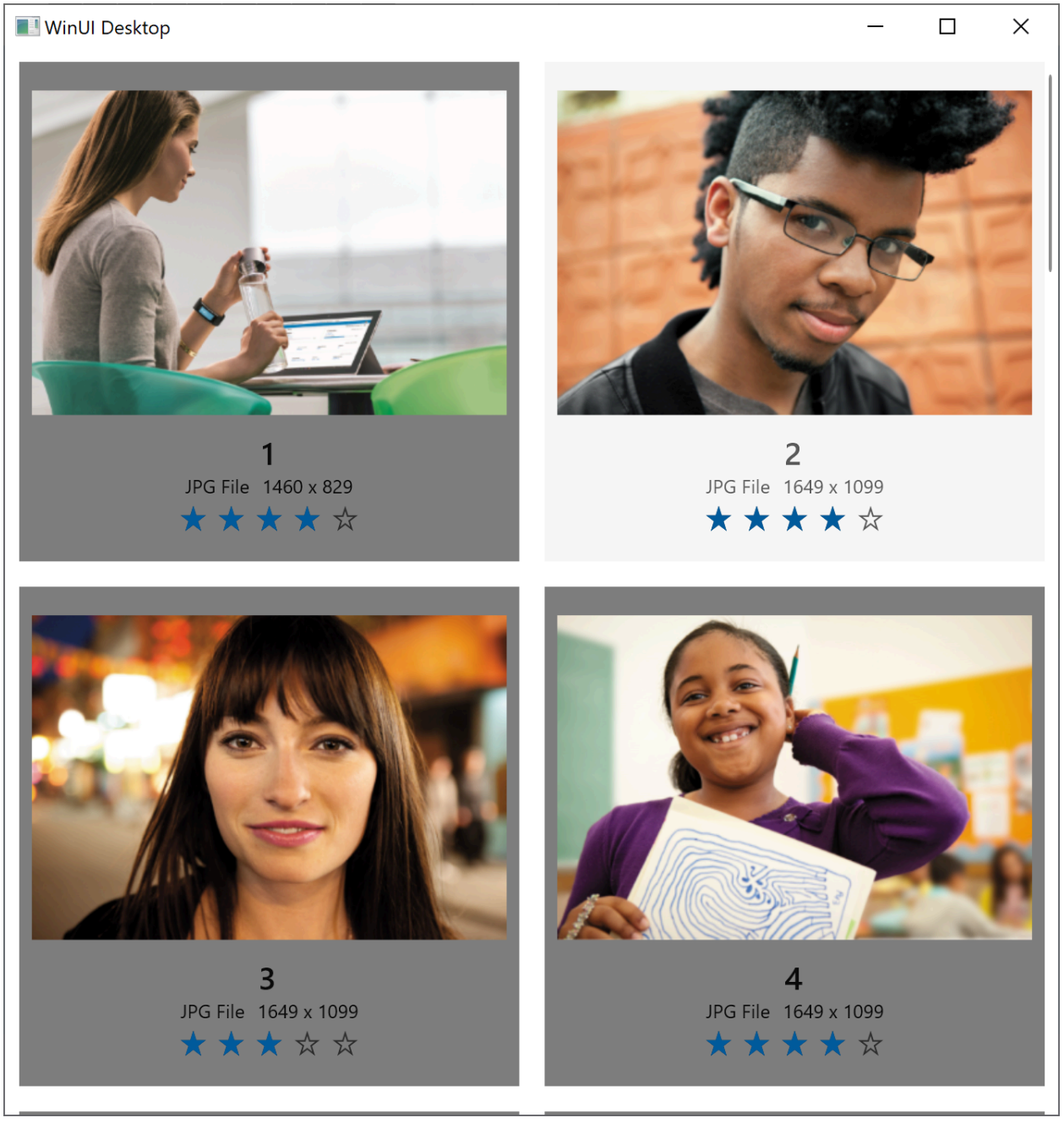
For more info, see {x:Bind} markup extension and Data binding in depth.

4. In the same way, replace the values of the other **TextBlock**s and the **RatingControl**. Here's the result:

XAML

```
<TextBlock Text="{x:Bind ImageTitle}" ... />
<StackPanel ... >
    <TextBlock Text="{x:Bind ImageFileType}" ... />
    <TextBlock Text="{x:Bind ImageDimensions}" ... />
</StackPanel>
<RatingControl Value="{x:Bind ImageRating}" ... />
```

If you build and run the app now, instead of placeholders you'll see real photos, and real text (and other data). Visually and functionally, this simple little app is now complete. But as a coda, let's do one last little bit of data binding.



Step 13: Bind the GridView to the Images collection (C# only)

> **Important**
>
> Perform this last step only if you created a C# project.

> **Tip**
>
> You'll find that there are some things (usually related to dynamically-generated UI) that you can't do in XAML markup. But in general if you *can* do something in markup, then that's preferable. It gives a slightly cleaner separation between the *view* that the XAML markup represents and the *model* (or *view model*) that the imperative code represents. And that tends to improve workflow in tooling and between team members.

We're currently using imperative code to associate the **GridView**'s ItemsSource) property with **MainWindow**'s **Images** property. But we can do that in markup instead.

1. In the **MainWindow** class, delete (or comment-out) the last line of **GetItemsAsync**, which sets the **ItemsSource** of `ImageGridView` to the value of the **Images** property.

2. And then in `MainWindow.xaml`, find the **GridView** named *ImageGridView*, and add an **ItemsSource** attribute like this. You can use IntelliSense to make this change if you like.

   XAML

   ```
   <GridView x:Name="ImageGridView"
             ...
             ItemsSource="{x:Bind Images}"
   ```

The **Images** property value doesn't change at run-time for this particular app. But because **Images** is of type `ObservableCollection<T>`, the *contents* of the collection can change (that is, elements can be added or deleted), and the binding will automatically notice the changes and update the UI.

Conclusion

In this tutorial we walked through the process of using Visual Studio to build a simple WinUI 3 app that displays photos. Hopefully this tutorial has given you experience working in a WinUI 3 app with controls, layout panels, data-binding, and GridView UI optimization.

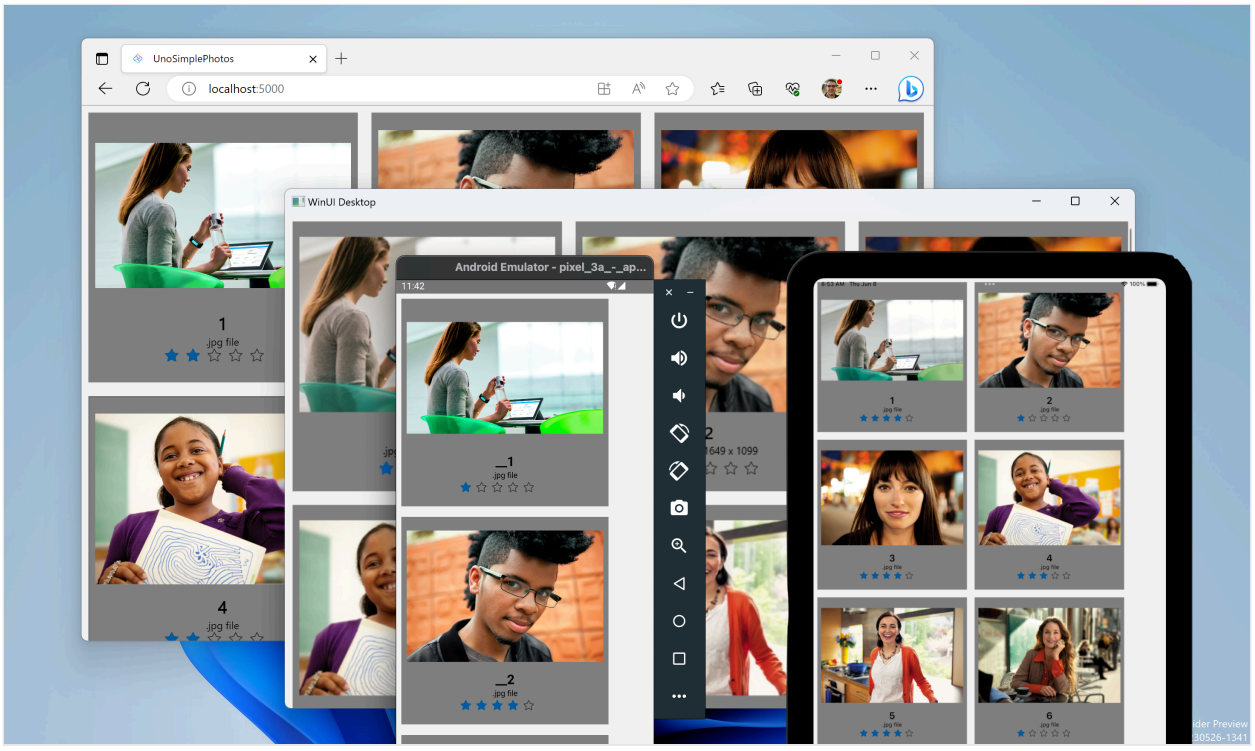
See also

Tutorial: Build a simple photo viewer that targets multiple platforms

Tutorial: Build a simple photo viewer that targets multiple platforms

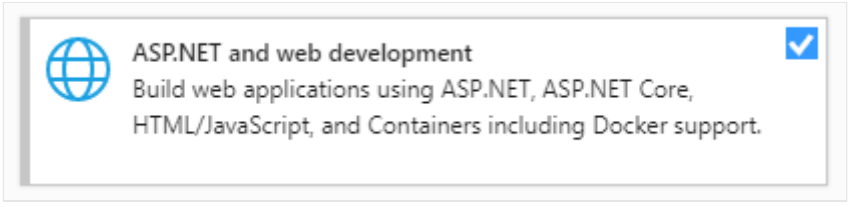
Article • 08/19/2024

After you've created a starter simple photo viewer WinUI 3 app, you might be wondering how to reach more users without having to rewrite your app. This tutorial uses Uno Platform to expand the reach of your existing C# WinUI 3 application enabling reuse of the business logic and UI layer across native mobile, web, and desktop. With only minimal changes to the simple photo viewer app, we can run a pixel-perfect copy of the app ported to these platforms.

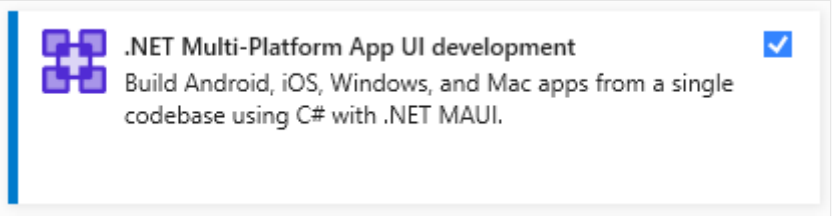


Prerequisites

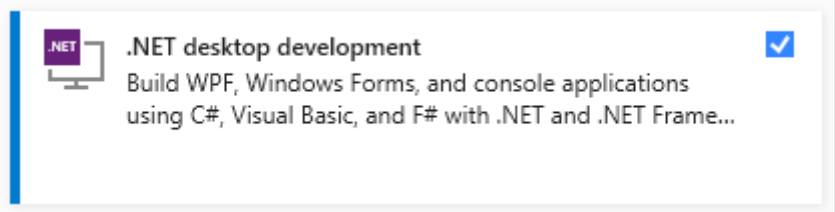
- Visual Studio 2022 17.4 or later
- Set up your development computer (see Get started with WinUI)
- ASP.NET and web development workload (for WebAssembly development)



- .NET Multi-platform App UI development installed (for iOS, Android, Mac Catalyst development)



- .NET desktop development installed (for Gtk, Wpf, and Linux Framebuffer development)



Finalize your environment

1. Open a command-line prompt, Windows Terminal if you have it installed, or else Command Prompt or Windows Powershell from the Start Menu.
2. Install or update the `uno-check` tool:
    - Use the following command:

      .NET CLI

      ```
      dotnet tool install -g uno.check
      ```

    - To update the tool, if you already have previously installed an older version:

      .NET CLI

      ```
      dotnet tool update -g uno.check
      ```

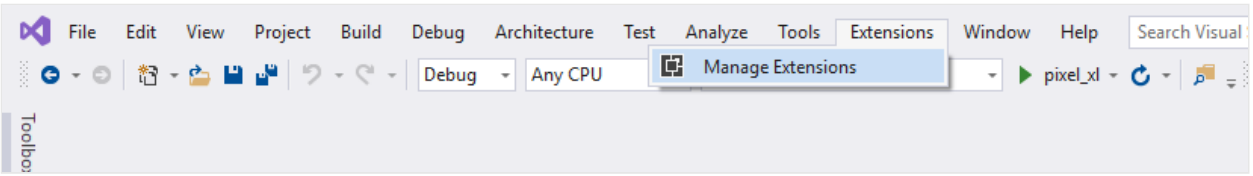
3. Run the tool with the following command:

   ```
   uno-check
   ```

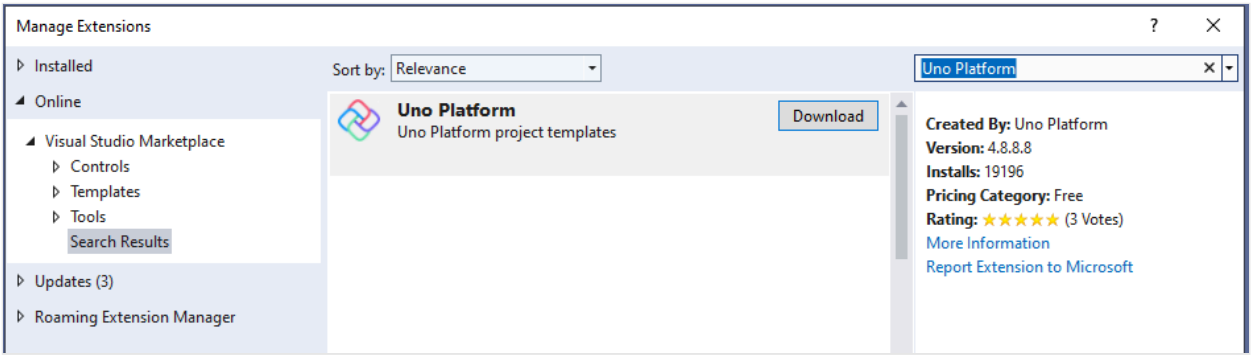
4. Follow the instructions indicated by the tool. Because it needs to modify your system, you may be prompted for elevated permissions.

Install the Uno Platform solution templates

Launch Visual Studio, then click `Continue without code`. Click `Extensions` -> `Manage Extensions` from the Menu Bar.



In the Extension Manager expand the **Online** node and search for `Uno`, install the `Uno Platform` extension, or download and install it from the Visual Studio Marketplace, then restart Visual Studio.

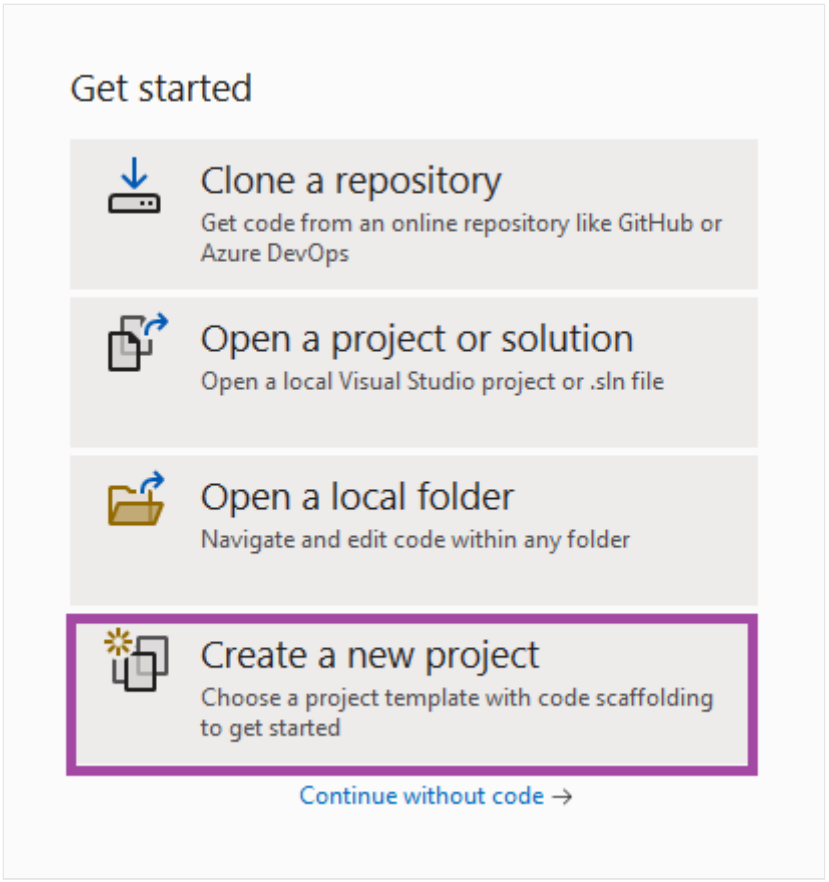


Create an application

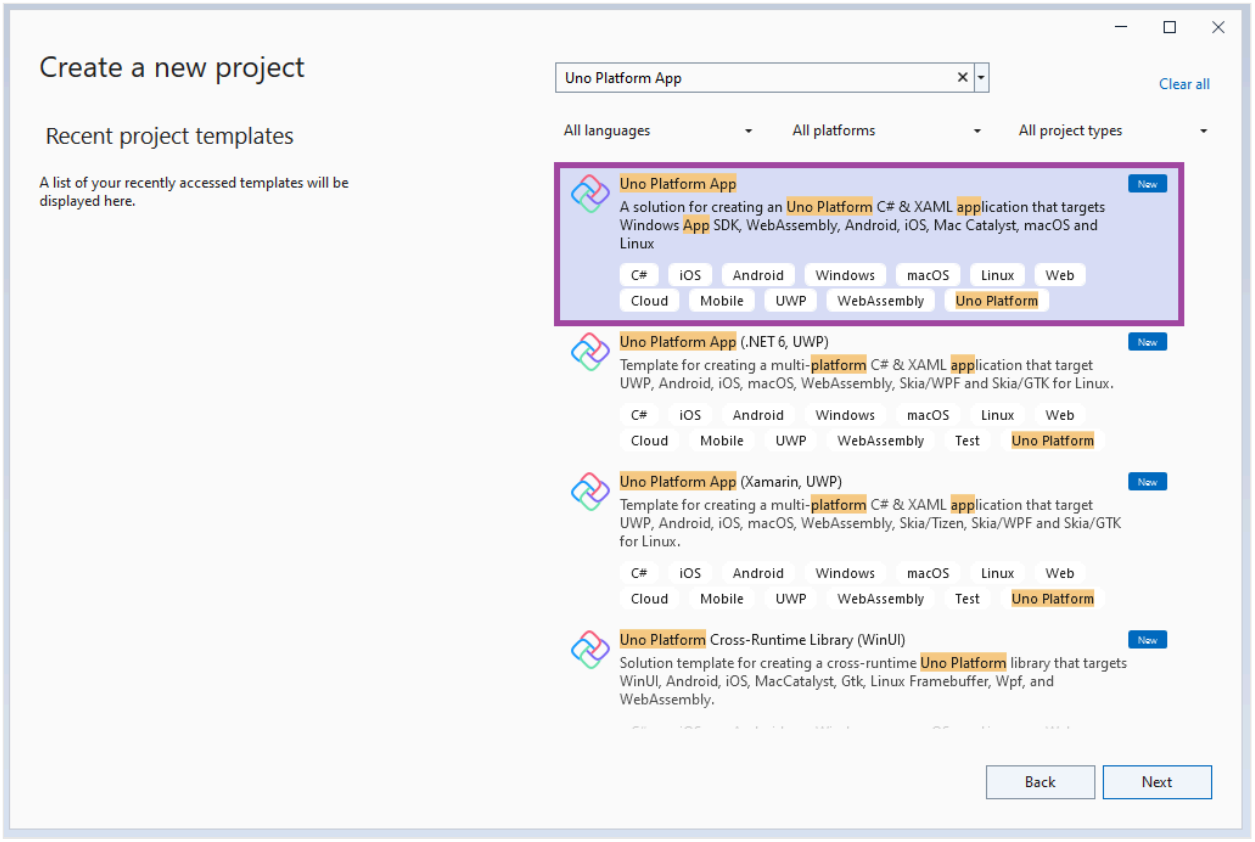
Now that we're ready to create a multi-platform application, the approach we'll take is to create a new Uno Platform application. We'll copy code from the previous tutorial's **SimplePhotos** WinUI 3 project into our multi-platform project. This is possible because Uno Platform lets you reuse your existing codebase. For features dependent on OS APIs provided by each platform, you can easily make them work over time. This approach is especially useful if you have an existing application that you want to port to other platforms.

Soon enough, you will be able to reap the benefits of this approach, as you can target more platforms with a familiar XAML flavor and the codebase you already have.

Open Visual Studio and create a new project via `File` > `New` > `Project`:

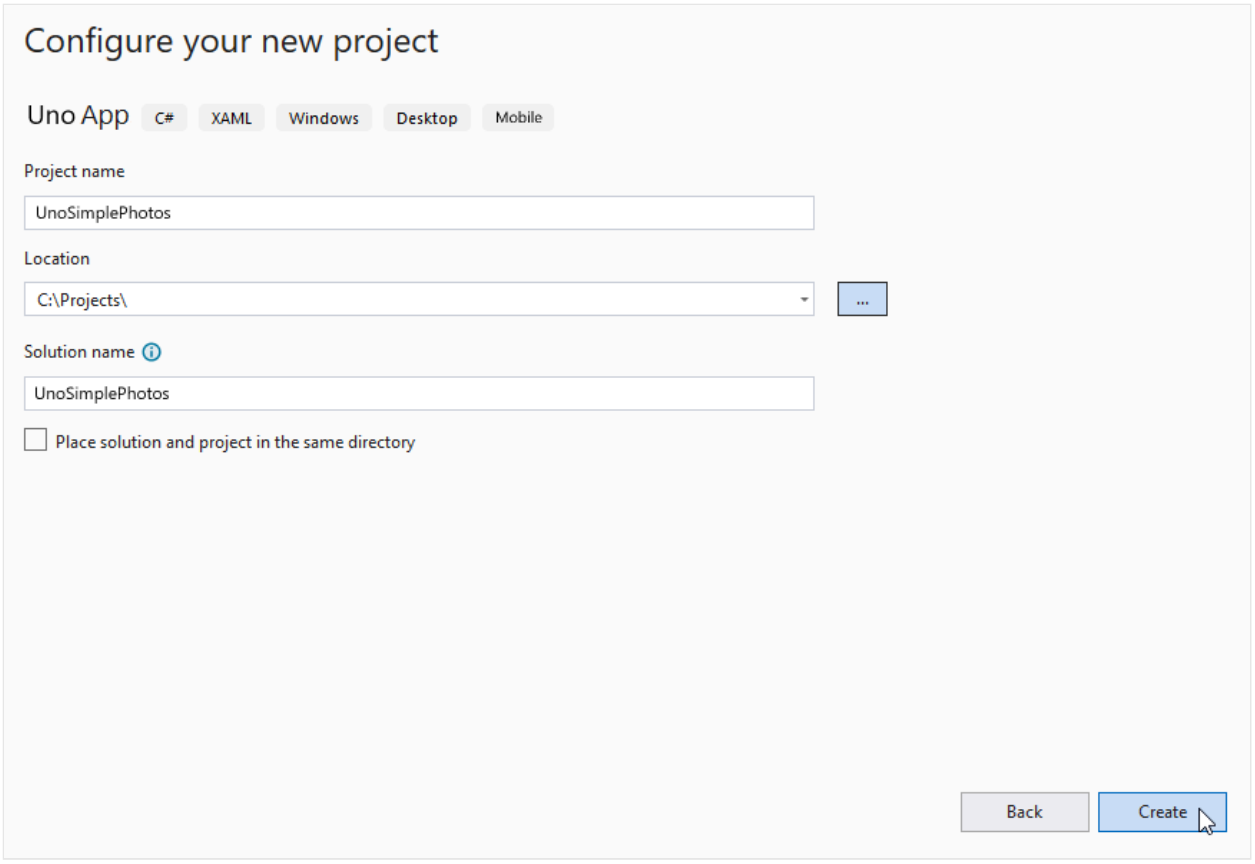


Search for Uno and select the Uno Platform App project template:



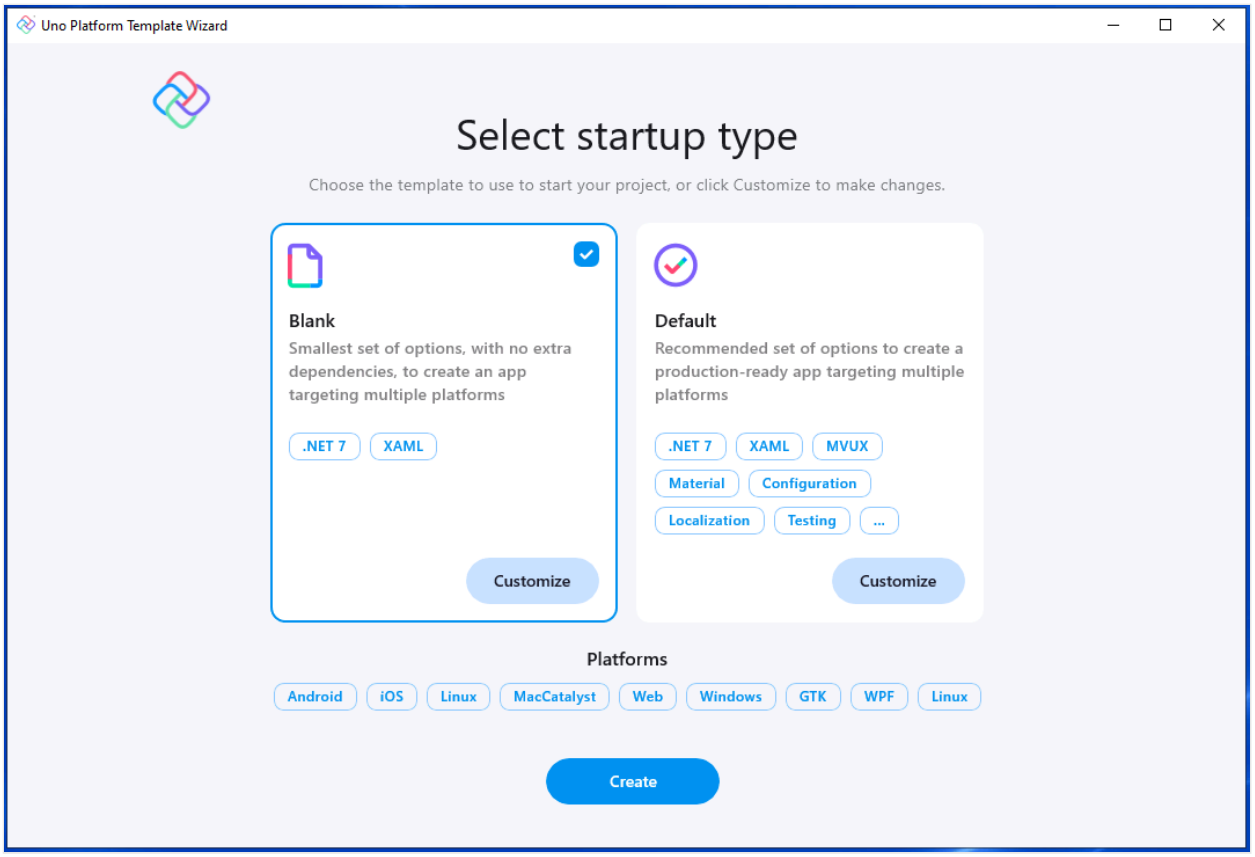
Create a new C# solution using the **Uno Platform App** type from Visual Studio's **Start Page**. To avoid conflicting with the code from the previous tutorial, we'll give this solution a different name, "UnoSimplePhotos". Specify the project name, solution name,

and directory. In this example, our `UnoSimplePhotos` multi-platform project belongs to a `UnoSimplePhotos` solution, which will live in C:\Projects:



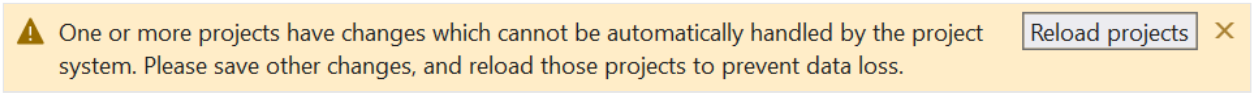
Now you'll choose a base template to take your Simple Photo gallery application multi-platform.

The Uno Platform App template comes with two preset options that allow you to quickly get started with either a **Blank** solution or the **Default** configuration which includes references to the Uno.Material and Uno.Toolkit libraries. The Default configuration also includes Uno.Extensions which is used for dependency injection, configuration, navigation, and logging. In addition, it uses MVUX in place of MVVM, making it a great starting point for rapidly building real-world applications.

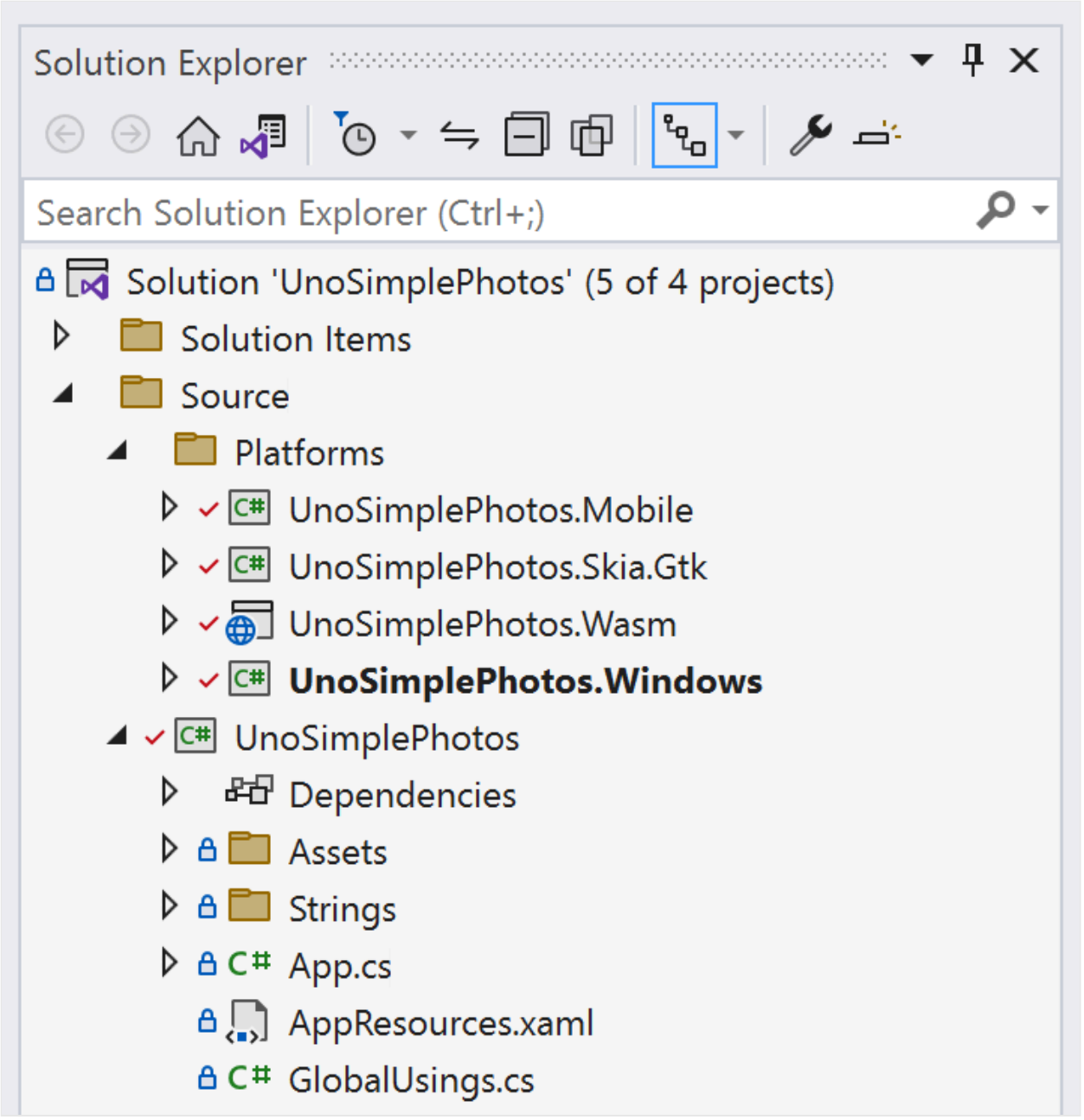


To keep things simple, select the **Blank** preset. Then, click the **Create** button. Wait for the projects to be created and their dependencies to be restored.

A banner at the top of the editor may ask to reload projects, click **Reload projects**:



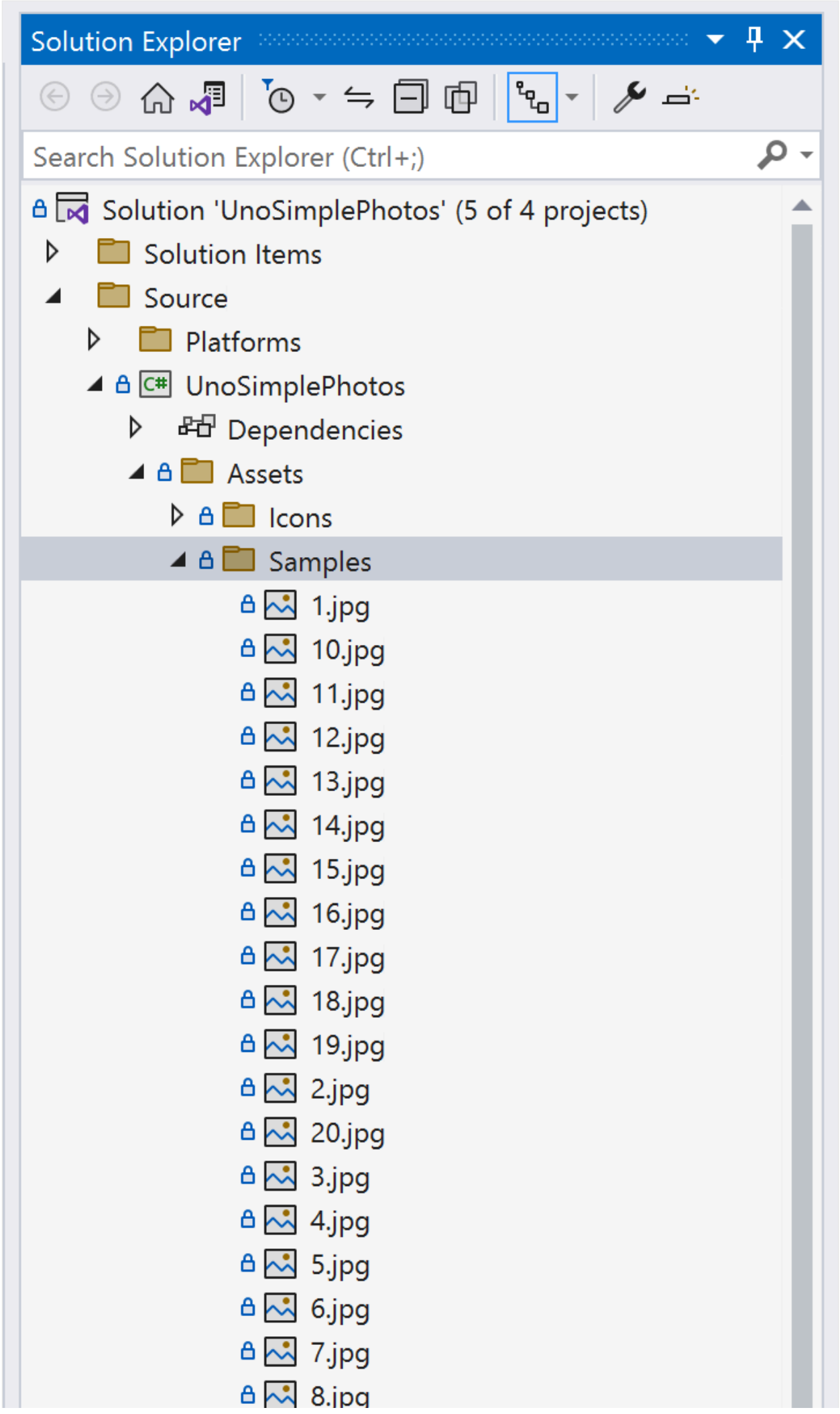
You should see the following default file structure in your **Solution Explorer**:

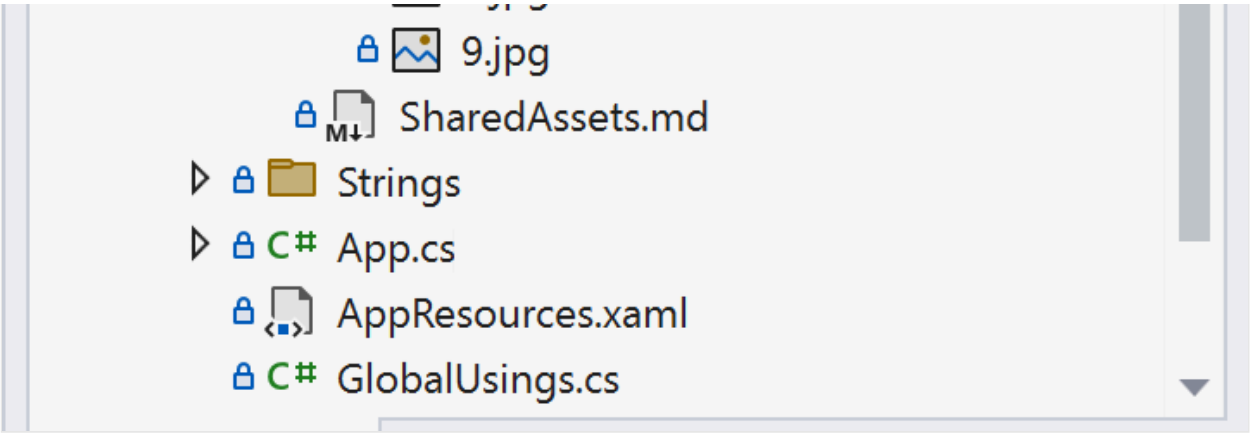


Add image assets to the project

Your app will need some images to display. You can use the same images from the previous tutorial.

In the `UnoSimplePhotos` project, create a new folder named `Assets` and copy the JPG image files to a `Samples` subfolder. The `Assets` folder structure should now look like this:


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'UnoSimplePhotos' (5 of 4 projects)
Solution Items
Source
Platforms
UnoSimplePhotos
Dependencies
Assets
Icons
Samples
1.jpg
10.jpg
11.jpg
12.jpg
13.jpg
14.jpg
15.jpg
16.jpg
17.jpg
18.jpg
19.jpg
2.jpg
20.jpg
3.jpg
4.jpg
5.jpg
6.jpg
7.jpg
8.jpg



For more information on creating the `Assets` folder and adding images to it, see the Uno Platform documentation about Assets and image display.

Preparing your app

Now that you've generated the functional starting point of your multi-platform WinUI application, you can copy code into it from the desktop project.

Copy the view

Because Uno Platform allows you to use the XAML flavor you're already familiar with, you can copy the same code over that you created in the previous tutorial.

Return to the **SimplePhotos** project from the previous tutorial. In the **Solution Explorer**, find the file named `MainWindow.xaml` and open it. Observe that the contents of the view are defined within a `Window` element rather than a `Page`. This is because the desktop project is a WinUI 3 application, which can use `Window` elements to define the contents of the view:

XML

```
<Window x:Class="SimplePhotos.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="using:SimplePhotos"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        mc:Ignorable="d">

    <Grid>
        <Grid.Resources>
            <DataTemplate x:Key="ImageGridView_ItemTemplate"
                          x:DataType="local:ImageFileInfo">
                <Grid Height="300"
```

```
                      Width="300"
                      Margin="8">
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition Height="Auto" />
                    </Grid.RowDefinitions>

                    <Image x:Name="ItemImage"
                           Source="Assets/StoreLogo.png"
                           Stretch="Uniform" />

                    <StackPanel Orientation="Vertical"
                                Grid.Row="1">
                        <TextBlock Text="{x:Bind ImageTitle}"
                                   HorizontalAlignment="Center"
                                   Style="{StaticResource
SubtitleTextBlockStyle}" />
                        <StackPanel Orientation="Horizontal"
                                    HorizontalAlignment="Center">
                            <TextBlock Text="{x:Bind ImageFileType}"
                                       HorizontalAlignment="Center"
                                       Style="{StaticResource
CaptionTextBlockStyle}" />
                            <TextBlock Text="{x:Bind ImageDimensions}"
                                       HorizontalAlignment="Center"
                                       Style="{StaticResource
CaptionTextBlockStyle}"
                                       Margin="8,0,0,0" />
                        </StackPanel>

                        <RatingControl Value="{x:Bind ImageRating}"
                                       IsReadOnly="True"/>
                    </StackPanel>
                </Grid>
            </DataTemplate>

            <Style x:Key="ImageGridView_ItemContainerStyle"
                   TargetType="GridViewItem">
                <Setter Property="Background"
                        Value="Gray"/>
                <Setter Property="Margin"
                        Value="8"/>
            </Style>

            <ItemsPanelTemplate x:Key="ImageGridView_ItemsPanelTemplate">
                    <ItemsWrapGrid Orientation="Horizontal"
                                   HorizontalAlignment="Center"/>
                </ItemsPanelTemplate>
        </Grid.Resources>

        <GridView x:Name="ImageGridView"
                  ItemsSource="{x:Bind Images}"
                  ItemTemplate="{StaticResource ImageGridView_ItemTemplate}"
                  ItemContainerStyle="{StaticResource
ImageGridView_ItemContainerStyle}"
```

```
                  ItemsPanel="{StaticResource
ImageGridView_ItemsPanelTemplate}"

ContainerContentChanging="ImageGridView_ContainerContentChanging" />
    </Grid>
</Window>
```

Uno Platform's multi-platform implementation of the controls found in the `Window` element, such as `GridView`, `Image`, and `RatingControl`, ensure that the view itself will work on all supported platforms with only a trivial amount of effort. Copy the contents of this `Window` and paste them into the `Page` element of the `MainPage.xaml` file in the **UnoSimplePhotos** Uno Platform project. The `MainPage` view XAML should look like this:

XML

```
<Page x:Class="UnoSimplePhotos.MainPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:local="using:UnoSimplePhotos"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      mc:Ignorable="d">

    <Grid>
        <Grid.Resources>
            <DataTemplate x:Key="ImageGridView_ItemTemplate"
                          x:DataType="local:ImageFileInfo">
                <Grid Height="300"
                      Width="300"
                      Margin="8">
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition Height="Auto" />
                    </Grid.RowDefinitions>

                    <Image x:Name="ItemImage"
                           Source="Assets/StoreLogo.png"
                           Stretch="Uniform" />

                    <StackPanel Orientation="Vertical"
                                Grid.Row="1">
                        <TextBlock Text="{x:Bind ImageTitle}"
                                   HorizontalAlignment="Center"
                                   Style="{StaticResource
SubtitleTextBlockStyle}" />
                        <StackPanel Orientation="Horizontal"
                                    HorizontalAlignment="Center">
                            <TextBlock Text="{x:Bind ImageFileType}"
                                       HorizontalAlignment="Center"
                                       Style="{StaticResource
CaptionTextBlockStyle}" />
```

```
                            <TextBlock Text="{x:Bind ImageDimensions}"
                                       HorizontalAlignment="Center"
                                       Style="{StaticResource
CaptionTextBlockStyle}"
                                       Margin="8,0,0,0" />
                        </StackPanel>

                        <RatingControl Value="{x:Bind ImageRating}"
                                       IsReadOnly="True"/>
                    </StackPanel>
                </Grid>
            </DataTemplate>

            <Style x:Key="ImageGridView_ItemContainerStyle"
                   TargetType="GridViewItem">
                <Setter Property="Background"
                        Value="Gray"/>
                <Setter Property="Margin"
                        Value="8"/>
            </Style>

            <ItemsPanelTemplate x:Key="ImageGridView_ItemsPanelTemplate">
                <ItemsWrapGrid Orientation="Horizontal"
                               HorizontalAlignment="Center"/>
            </ItemsPanelTemplate>
        </Grid.Resources>

        <GridView x:Name="ImageGridView"
                  ItemsSource="{x:Bind Images}"
                  ItemTemplate="{StaticResource ImageGridView_ItemTemplate}"
                  ItemContainerStyle="{StaticResource
ImageGridView_ItemContainerStyle}"
                  ItemsPanel="{StaticResource
ImageGridView_ItemsPanelTemplate}"

ContainerContentChanging="ImageGridView_ContainerContentChanging">
        </GridView>
    </Grid>
</Page>
```

You may recall that the desktop solution also had a `MainWindow.xaml.cs` file that contained code-behind which corresponds to the view. In the Uno Platform project, the code-behind for the `MainPage` view we've copied into is contained in the `MainPage.xaml.cs` file.

To bring this code-behind multi-platform, we should first move the following into the `MainPage.xaml.cs` file:

- `Images` property: Provides the `GridView` with an observable collection of image files

- Contents of the constructor: Calls `GetItemsAsync()` to populate the `Images` collection with items representing image files
- Remove the manual modification of the `ImageGridView` control's `ItemsSource` property
- `ImageGridView_ContainerContentChanging` method: Used as part of a strategy to progressively load `GridView` items as they are scrolled into view
- `ShowImage` method: Loads the image files into the `GridView`
- `GetItemsAsync` method: Gets the image asset files from the `Samples` folder
- `LoadImageInfoAsync` method: Constructs an `ImageFileInfo` object from a created `StorageFile`

After moving everything over, `MainPage.xaml.cs` should now look like this:

C#

```
using Microsoft.UI.Xaml.Controls;
using System.Collections.ObjectModel;
using Windows.Storage;
using Windows.Storage.Search;

namespace UnoSimplePhotos;

public sealed partial class MainPage : Page
{
    public ObservableCollection<ImageFileInfo> Images { get; }
    = new ObservableCollection<ImageFileInfo>();

    public MainPage()
    {
        this.InitializeComponent();
        GetItemsAsync();
    }

    private void ImageGridView_ContainerContentChanging(ListViewBase sender,
        ContainerContentChangingEventArgs args)
    {
        if (args.InRecycleQueue)
        {
            var templateRoot = args.ItemContainer.ContentTemplateRoot as
Grid;
            var image = templateRoot.FindName("ItemImage") as Image;
            image.Source = null;
        }

        if (args.Phase == 0)
        {
```

```
            args.RegisterUpdateCallback(ShowImage);
            args.Handled = true;
        }
    }

    private async void ShowImage(ListViewBase sender,
ContainerContentChangingEventArgs args)
    {
        if (args.Phase == 1)
        {
            // It's phase 1, so show this item's image.
            var templateRoot = args.ItemContainer.ContentTemplateRoot as
Grid;
            var image = templateRoot.FindName("ItemImage") as Image;
            var item = args.Item as ImageFileInfo;
            image.Source = await item.GetImageThumbnailAsync();
        }
    }

    private async Task GetItemsAsync()
    {
        StorageFolder appInstalledFolder =
Package.Current.InstalledLocation;
        StorageFolder picturesFolder = await
appInstalledFolder.GetFolderAsync("Assets\\Samples");

        var result = picturesFolder.CreateFileQueryWithOptions(new
QueryOptions());

        IReadOnlyList<StorageFile> imageFiles = await
result.GetFilesAsync();
        foreach (StorageFile file in imageFiles)
        {
            Images.Add(await LoadImageInfoAsync(file));
        }
    }

    public async static Task<ImageFileInfo> LoadImageInfoAsync(StorageFile
file)
    {
        var properties = await file.Properties.GetImagePropertiesAsync();
        ImageFileInfo info = new(properties,
                                 file, file.DisplayName,
file.DisplayType);

        return info;
    }
}
```

> **Note**
>
> The files in your Uno app project should use `UnoSimplePhotos` as the namespace.

So far, the files for the main view we're working with contain all the capabilities of the desktop solution. After we copy over the `ImageFileInfo.cs` model file, we will learn how to modify the desktop-oriented blocks of code for multi-platform compatibility.

Copy `ImageFileInfo` from the desktop project and paste it into the `ImageFileInfo.cs` file. Make the following changes:

- Rename the namespace to be `UnoSimplePhotos` instead of `SimplePhotos`:

  C#

  ```
  // Found towards the top of the file
  namespace UnoSimplePhotos;
  ```

- Change the parameter type of the `OnPropertyChanged` method to be nullable:

  C#

  ```
  // string -> string?
  protected void OnPropertyChanged([CallerMemberName] string?
  propertyName = null)
  ...
  ```

- Make the `PropertyChangedEventHandler` nullable:

  C#

  ```
  // PropertyChangedEventHandler -> PropertyChangedEventHandler?
  public event PropertyChangedEventHandler? PropertyChanged;
  ```

Put together, the file should look like this:

C#

```
using Microsoft.UI.Xaml.Media.Imaging;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using Windows.Storage;
using Windows.Storage.FileProperties;
using Windows.Storage.Streams;
using ThumbnailMode = Windows.Storage.FileProperties.ThumbnailMode;

namespace UnoSimplePhotos;

public class ImageFileInfo : INotifyPropertyChanged
{
    public ImageFileInfo(ImageProperties properties,
```

```
        StorageFile imageFile,
        string name,
        string type)
    {
        ImageProperties = properties;
        ImageName = name;
        ImageFileType = type;
        ImageFile = imageFile;
        var rating = (int)properties.Rating;
        var random = new Random();
        ImageRating = rating == 0 ? random.Next(1, 5) : rating;
    }

    public StorageFile ImageFile { get; }

    public ImageProperties ImageProperties { get; }

    public async Task<BitmapImage> GetImageSourceAsync()
    {
        using IRandomAccessStream fileStream = await
ImageFile.OpenReadAsync();

        // Create a bitmap to be the image source.
        BitmapImage bitmapImage = new();
        bitmapImage.SetSource(fileStream);

        return bitmapImage;
    }

    public async Task<BitmapImage> GetImageThumbnailAsync()
    {
        StorageItemThumbnail thumbnail =
            await ImageFile.GetThumbnailAsync(ThumbnailMode.PicturesView);
        // Create a bitmap to be the image source.
        var bitmapImage = new BitmapImage();
        bitmapImage.SetSource(thumbnail);
        thumbnail.Dispose();

        return bitmapImage;
    }

    public string ImageName { get; }

    public string ImageFileType { get; }

    public string ImageDimensions => $"{ImageProperties.Width} x
{ImageProperties.Height}";

    public string ImageTitle
    {
        get => string.IsNullOrEmpty(ImageProperties.Title) ? ImageName :
ImageProperties.Title;
        set
        {
            if (ImageProperties.Title != value)
```

```
            {
                ImageProperties.Title = value;
                _ = ImageProperties.SavePropertiesAsync();
                OnPropertyChanged();
            }
        }
    }

    public int ImageRating
    {
        get => (int)ImageProperties.Rating;
        set
        {
            if (ImageProperties.Rating != value)
            {
                ImageProperties.Rating = (uint)value;
                _ = ImageProperties.SavePropertiesAsync();
                OnPropertyChanged();
            }
        }
    }

    public event PropertyChangedEventHandler? PropertyChanged;

    protected void OnPropertyChanged([CallerMemberName] string? propertyName
= null) =>
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
}
```

This class will serve as a model to represent the image files in the `GridView`. Although it should technically possible to run the app at this point, it may not render the images correctly or display their properties. In the next sections, we'll make a set of changes to these copied files to make them compatible in a multi-platform context.

Using preprocessor directives

In the desktop project from the previous tutorial, the `MainPage.xaml.cs` file contains a `GetItemsAsync` method that enumerates items from a `StorageFolder` representing the installed package location. Because that location is not available on certain platforms such as WebAssembly, we'll need to make changes to this method to make it compatible with all platforms. We'll accordingly make some changes to the `ImageFileInfo` class to ensure compatibility.

First, make the required changes to the `GetItemsAsync` method. Replace the `GetItemsAsync` method in the `MainPage.xaml.cs` file with the following code:

C#

```
private async Task GetItemsAsync()
{
#if WINDOWS
    StorageFolder appInstalledFolder = Package.Current.InstalledLocation;
    StorageFolder picturesFolder = await
appInstalledFolder.GetFolderAsync("UnoSimplePhotos\\Assets\\Samples");

    var result = picturesFolder.CreateFileQueryWithOptions(new
QueryOptions());

    IReadOnlyList<StorageFile> imageFiles = await result.GetFilesAsync();
#else
    var imageFileNames = Enumerable.Range(1, 20).Select(i => new Uri($"ms-
appx:///UnoSimplePhotos/Assets/Samples/{i}.jpg"));
    var imageFiles = new List<StorageFile>();

    foreach (var file in imageFileNames)
    {
        imageFiles.Add(await
StorageFile.GetFileFromApplicationUriAsync(file));
    }
#endif
    foreach (StorageFile file in imageFiles)
    {
        Images.Add(await LoadImageInfoAsync(file));
    }
}
```

This method now uses a **preprocessor directive** to determine which code to execute based on the platform. On Windows, the method gets the `StorageFolder` representing the installed package location and returns the `Samples` folder from it. On other platforms, the method counts up to 20, getting the image files from the `Samples` folder using a `Uri` to represent the image file.

Next, adjust the `LoadImageInfoAsync` method to accommodate the changes we made to the `GetItemsAsync` method. Replace the `LoadImageInfoAsync` method in the `MainPage.xaml.cs` file with the following code:

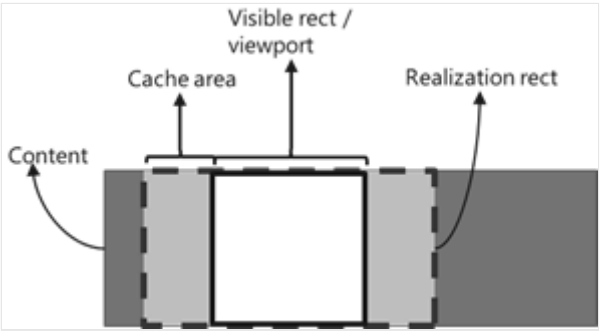
C#

```
public async static Task<ImageFileInfo> LoadImageInfoAsync(StorageFile file)
{
#if WINDOWS
    var properties = await file.Properties.GetImagePropertiesAsync();
    ImageFileInfo info = new(properties,
                                file, file.DisplayName, $"{file.FileType}
file");
#else
    ImageFileInfo info = new(file, file.DisplayName, $"{file.FileType}
```

```
file");
#endif
    return info;
}
```

Similar to the `GetItemsAsync` method, this method now uses a preprocessor directive to determine which code to execute based on the platform. On Windows, the method gets the `ImageProperties` from the `StorageFile` and uses it to create an `ImageFileInfo` object. On other platforms, the method constructs an `ImageFileInfo` object without the `ImageProperties` parameter. Later, modifications will be made to the `ImageFileInfo` class to accommodate this change.

Controls like `GridView` allow for **progressive loading** of updated item container content as they are scrolled into the viewport. This is done by using the `ContainerContentChanging` event. In the desktop project from the previous tutorial, the `ImageGridView_ContainerContentChanging` method uses this event to load the image files into the `GridView`. Because certain aspects of this event are not supported on all platforms, we'll need to make changes to this method to make it compatible with them.



For instance, the `ContainerContentChangingEventArgs.Phase` property is currently unsupported on platforms other than Windows. We'll need to make changes to the `ImageGridView_ContainerContentChanging` method to accommodate this change. Replace the `ImageGridView_ContainerContentChanging` method in the `MainPage.xaml.cs` file with the following code:

C#

```
private void ImageGridView_ContainerContentChanging(
ListViewBase sender,
ContainerContentChangingEventArgs args)
{

    if (args.InRecycleQueue)
    {
        var templateRoot = args.ItemContainer.ContentTemplateRoot as Grid;
        var image = templateRoot?.FindName("ItemImage") as Image;
        if (image is not null)
```

```
        {
            image.Source = null;
        }
    }

#if WINDOWS
        if (args.Phase == 0)
        {
            args.RegisterUpdateCallback(ShowImage);
            args.Handled = true;
        }
#else
    ShowImage(sender, args);
#endif
}
```

The specialized callback is now only registered using `ContainerContentChangingEventArgs.RegisterUpdateCallback()` if the platform is Windows. Otherwise, the `ShowImage` method is called directly. We'll also need to make changes to the `ShowImage` method to work alongside the changes made to the `ImageGridView_ContainerContentChanging` method. Replace the `ShowImage` method in the `MainPage.xaml.cs` file with the following code:

C#

```
private async void ShowImage(ListViewBase sender,
ContainerContentChangingEventArgs args)
{
    if (
#if WINDOWS
            args.Phase == 1
#else
        true
#endif
        )
    {

        // It's phase 1, so show this item's image.
        var templateRoot = args.ItemContainer.ContentTemplateRoot as Grid;
        var image = templateRoot?.FindName("ItemImage") as Image;
        var item = args.Item as ImageFileInfo;
#if WINDOWS
        if (image is not null && item is not null)
        {
            image.Source = await item.GetImageThumbnailAsync();
        }
#else
        if (item is not null)
        {
            await item.GetImageSourceAsync();
        }
```

```
#endif
    }
}
```

Again, preprocessor directives ensure that the `ContainerContentChangingEventArgs.Phase` property is only used on platforms where it is supported. We make use of the previously unused `GetImageSourceAsync()` method to load the image files into the `GridView` on platforms other than Windows. At this point, we'll accommodate the changes made above by editing the `ImageFileInfo` class.

Creating a separate code path for other platforms

Update `ImageFileInfo.cs` to include a new property called `ImageSource` that will be used to load the image file.

C#

```
public BitmapImage? ImageSource { get; private set; }
```

Because platforms like the web do not support advanced image file properties that are readily available on Windows, we'll add a constructor overload that does not require an `ImageProperties` typed parameter. Add the new overload after the existing one using the following code:

C#

```
public ImageFileInfo(StorageFile imageFile,
    string name,
    string type)
{
    ImageName = name;
    ImageFileType = type;
    ImageFile = imageFile;
}
```

This constructor overload is used to construct an `ImageFileInfo` object on platforms other than Windows. Since we did this, it makes sense to make the `ImageProperties` property nullable. Update the `ImageProperties` property to be nullable using the following code:

C#

```
public ImageProperties? ImageProperties { get; }
```

Update the `GetImageSourceAsync` method to use the `ImageSource` property instead of only returning a `BitmapImage` object. Replace the `GetImageSourceAsync` method in the `ImageFileInfo.cs` file with the following code:

C#

```
public async Task<BitmapImage> GetImageSourceAsync()
{
    using IRandomAccessStream fileStream = await ImageFile.OpenReadAsync();

    // Create a bitmap to be the image source.
    BitmapImage bitmapImage = new();
    bitmapImage.SetSource(fileStream);

    ImageSource = bitmapImage;
    PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(ImageSource)));

    return bitmapImage;
}
```

To prevent getting the value of `ImageProperties` when it's null, make the following changes:

- Modify the `ImageDimensions` property to use the null conditional operator:

  C#

  ```
  public string ImageDimensions => $"{ImageProperties?.Width} x
  {ImageProperties?.Height}";
  ```

- Change the `ImageTitle` property to use the null conditional operator:

  C#

  ```
  public string ImageTitle
  {
      get => string.IsNullOrEmpty(ImageProperties?.Title) ? ImageName :
  ImageProperties?.Title;
      set
      {
          if (ImageProperties is not null)
          {
              if (ImageProperties.Title != value)
              {
                  ImageProperties.Title = value;
                  _ = ImageProperties.SavePropertiesAsync();
                  OnPropertyChanged();
              }
  ```

```
        }
    }
}
```

- Change `ImageRating` to not rely on `ImageProperties` by generating a random star rating for demonstration purposes:

```
C#

public int ImageRating
{
    get => (int)((ImageProperties?.Rating == null ||
ImageProperties.Rating == 0) ? (uint)Random.Shared.Next(1, 5) :
ImageProperties.Rating);
    set
    {
        if (ImageProperties is not null)
        {
            if (ImageProperties.Rating != value)
            {
                ImageProperties.Rating = (uint)value;
                _ = ImageProperties.SavePropertiesAsync();
                OnPropertyChanged();
            }
        }
    }
}
```

- Update the constructor that generates a random integer to no longer do this:

```
C#

public ImageFileInfo(ImageProperties properties,
    StorageFile imageFile,
    string name,
    string type)
{
    ImageProperties = properties;
    ImageName = name;
    ImageFileType = type;
    ImageFile = imageFile;
}
```

With these edits, the `ImageFileInfo` class should contain the following code. It now has a newly-separated code path for platforms other than Windows:

```
C#
```

```
using Microsoft.UI.Xaml.Media.Imaging;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using Windows.Storage;
using Windows.Storage.FileProperties;
using Windows.Storage.Streams;
using ThumbnailMode = Windows.Storage.FileProperties.ThumbnailMode;

namespace UnoSimplePhotos;

public class ImageFileInfo : INotifyPropertyChanged
{
    public BitmapImage? ImageSource { get; private set; }

    public ImageFileInfo(ImageProperties properties,
        StorageFile imageFile,
        string name,
        string type)
    {
        ImageProperties = properties;
        ImageName = name;
        ImageFileType = type;
        ImageFile = imageFile;
    }

    public ImageFileInfo(StorageFile imageFile,
        string name,
        string type)
    {
        ImageName = name;
        ImageFileType = type;
        ImageFile = imageFile;
    }

    public StorageFile ImageFile { get; }

    public ImageProperties? ImageProperties { get; }

    public async Task<BitmapImage> GetImageSourceAsync()
    {
        using IRandomAccessStream fileStream = await
ImageFile.OpenReadAsync();

        // Create a bitmap to be the image source.
        BitmapImage bitmapImage = new();
        bitmapImage.SetSource(fileStream);

        ImageSource = bitmapImage;
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(nameof(ImageSource)));

        return bitmapImage;
    }
```

```
    public async Task<BitmapImage> GetImageThumbnailAsync()
    {
        StorageItemThumbnail thumbnail =
            await ImageFile.GetThumbnailAsync(ThumbnailMode.PicturesView);
        // Create a bitmap to be the image source.
        var bitmapImage = new BitmapImage();
        bitmapImage.SetSource(thumbnail);
        thumbnail.Dispose();

        return bitmapImage;
    }

    public string ImageName { get; }

    public string ImageFileType { get; }

    public string ImageDimensions => $"{ImageProperties?.Width} x
{ImageProperties?.Height}";

    public string ImageTitle
    {
        get => string.IsNullOrEmpty(ImageProperties?.Title) ? ImageName :
ImageProperties.Title;
        set
        {
            if (ImageProperties is not null)
            {
                if (ImageProperties.Title != value)
                {
                    ImageProperties.Title = value;
                    _ = ImageProperties.SavePropertiesAsync();
                    OnPropertyChanged();
                }
            }
        }
    }

    public int ImageRating
    {
        get => (int)((ImageProperties?.Rating == null ||
ImageProperties.Rating == 0) ? (uint)Random.Shared.Next(1, 5) :
ImageProperties.Rating);
        set
        {
            if (ImageProperties is not null)
            {
                if (ImageProperties.Rating != value)
                {
                    ImageProperties.Rating = (uint)value;
                    _ = ImageProperties.SavePropertiesAsync();
                    OnPropertyChanged();
                }
            }
        }
    }
```

```
    public event PropertyChangedEventHandler? PropertyChanged;

    protected void OnPropertyChanged([CallerMemberName] string? propertyName
= null) =>
        PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
}
```

This `ImageFileInfo` class is used to represent the image files in the `GridView`. Finally, we'll make changes to the `MainPage.xaml` file to accommodate the changes to the model.

Using platform-specific XAML markup

There are a couple of items in our view markup which should only be evaluated on Windows. Add a new namespace on the `Page` element of the `MainPage.xaml` file like this:

XML

```
...
xmlns:win="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

Now, in `MainPage.xaml`, replace the `ItemsPanel` property setter on the `GridView` element with the following code:

XML

```
win:ItemsPanel="{StaticResource ImageGridView_ItemsPanelTemplate}"
```

Prepending the property name with `win:` ensures that the property is only set on Windows. Do this again within the `ImageGridView_ItemTemplate` resource. We want to only load elements that use the `ImageDimensions` property on Windows. Replace the `TextBlock` element that uses the `ImageDimensions` property with the following code:

XML

```
<win:TextBlock Text="{x:Bind ImageDimensions}"
               HorizontalAlignment="Center"
               Style="{StaticResource CaptionTextBlockStyle}"
               Margin="8,0,0,0" />
```

The `MainPage.xaml` file should now look like this:

XML

```
<Page x:Class="UnoSimplePhotos.MainPage"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:local="using:UnoSimplePhotos"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:win="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      mc:Ignorable="d"
      Background="{ThemeResource ApplicationPageBackgroundThemeBrush}">

    <Grid>
        <Grid.Resources>
            <DataTemplate x:Key="ImageGridView_ItemTemplate"
                          x:DataType="local:ImageFileInfo">
                <Grid Height="300"
                      Width="300"
                      Margin="8">
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition Height="Auto" />
                    </Grid.RowDefinitions>

                    <Image x:Name="ItemImage"
                           Source="{x:Bind ImageSource}"
                           Stretch="Uniform" />

                    <StackPanel Orientation="Vertical"
                                Grid.Row="1">
                        <TextBlock Text="{x:Bind ImageTitle}"
                                   HorizontalAlignment="Center"
                                   Style="{StaticResource
SubtitleTextBlockStyle}" />
                        <StackPanel Orientation="Horizontal"
                                    HorizontalAlignment="Center">
                            <TextBlock Text="{x:Bind ImageFileType}"
                                       HorizontalAlignment="Center"
                                       Style="{StaticResource
CaptionTextBlockStyle}" />
                            <win:TextBlock Text="{x:Bind ImageDimensions}"
                                           HorizontalAlignment="Center"
                                           Style="{StaticResource
CaptionTextBlockStyle}"
                                           Margin="8,0,0,0" />
                        </StackPanel>

                        <RatingControl Value="{x:Bind ImageRating}"
                                       IsReadOnly="True" />
                    </StackPanel>
                </Grid>
            </DataTemplate>

            <Style x:Key="ImageGridView_ItemContainerStyle"
```

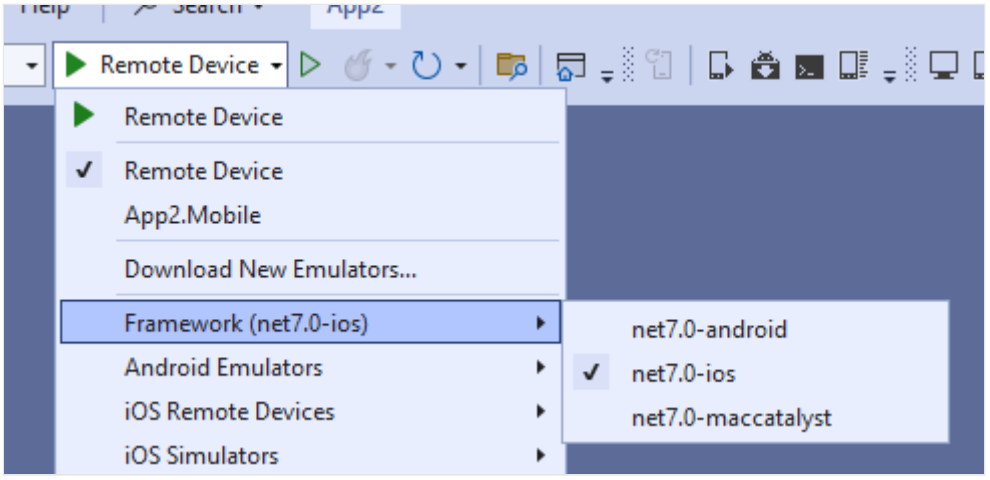
```
                    TargetType="GridViewItem">
                <Setter Property="Background"
                        Value="Gray" />
                <Setter Property="Margin"
                        Value="8"/>
            </Style>

            <ItemsPanelTemplate x:Key="ImageGridView_ItemsPanelTemplate">
                <ItemsWrapGrid Orientation="Horizontal"
                               HorizontalAlignment="Center"/>
            </ItemsPanelTemplate>
        </Grid.Resources>

        <GridView x:Name="ImageGridView"
                  ItemsSource="{x:Bind Images, Mode=OneWay}"
                  win:ItemsPanel="{StaticResource
ImageGridView_ItemsPanelTemplate}"

ContainerContentChanging="ImageGridView_ContainerContentChanging"
                  ItemContainerStyle="{StaticResource
ImageGridView_ItemContainerStyle}"
                  ItemTemplate="{StaticResource ImageGridView_ItemTemplate}"
/>
    </Grid>
</Page>
```

Running the app

Launch the `UnoSimplePhotos.Windows` target. Observe that this WinUI app is very similar to the previous tutorial.

You can now build and run your app on any of the supported platforms. To do so, you can use the debug toolbar drop-down to select a target platform to deploy:

- To run the **WebAssembly** (Wasm) head:
  - Right-click on the `UnoSimplePhotos.Wasm` project, select **Set as startup project**
  - Press the `UnoSimplePhotos.Wasm` button to deploy the app
  - If desired, you can add and use the `UnoSimplePhotos.Server` project as an alternative
- To debug for **iOS**:
  - Right-click on the `UnoSimplePhotos.Mobile` project, and select **Set as startup project**
  - In the debug toolbar drop-down, select an active iOS device or the simulator. You'll need to be paired with a Mac for this to work.



- To debug for **Mac Catalyst**:
  - Right-click on the `UnoSimplePhotos.Mobile` project, and select **Set as startup project**
  - In the debug toolbar drop-down, select a remote macOS device. You'll need to be paired with one for this to work.

- To debug the **Android** platform:
  - Right-click on the `UnoSimplePhotos.Mobile` project, and select **Set as startup project**
  - In the debug toolbar drop-down, select either an active Android device or the emulator
    - Select an active device in the "Device" sub-menu

- To debug on **Linux** with **Skia GTK**:
  - Right-click on the `UnoSimplePhotos.Skia.Gtk` project, and select **Set as startup project**
  - Press the `UnoSimplePhotos.Skia.Gtk` button to deploy the app

See also

- Uno Platform documentation
- Update ListView and GridView items progressively
- Simple photo viewer tutorial

WinUI

Article • 11/08/2024

Important

At this time, there are two generations of WinUI: **WinUI 2 for UWP** and **WinUI in the Windows App SDK (WinUI 3)**. While both can be used in production-ready apps on Windows 10 and later, each have different development targets.

See **Comparison of WinUI 3 and WinUI 2**.

WinUI is a native user experience (UX) framework for both Windows desktop and UWP applications.

By incorporating the Fluent Design System into all experiences, controls, and styles, WinUI provides consistent, intuitive, and accessible experiences using the latest user interface (UI) patterns.

With support for both desktop and UWP apps, you can build with WinUI from the ground up, or gradually migrate your existing MFC, WinForms, or WPF apps using familiar languages such as C++, C#, Visual Basic, and JavaScript (using React Native for Desktop).

> The WinUI libraries are hosted in the WinUI GitHub repo where you can file feature requests or bugs, and interact with the WinUI team.

Comparison of WinUI 3 and WinUI 2

The following table highlights some of most significant differences between WinUI 3 in the Windows App SDK and WinUI 2 for UWP.

Expand table

WinUI 3 in the Windows App SDK	WinUI 2 for UWP
WinUI 3 ships with the Windows App SDK.	**WinUI 2**, the 2nd generation of WinUI, ships as a standalone NuGet package, and is integrated with Windows SDKs.
UX stack and control library completely decoupled from the OS and Windows SDKs, including the core framework, composition, and input layers of the UX stack.	UX stack and control library tightly coupled to the OS and Windows SDKs.
WinUI 3 can be used to build production-ready **desktop/Win32** Windows apps.	WinUI 2 can be used in UWP applications and incorporated into desktop applications using XAML Islands (for installation instructions, see Getting started with the WinUI 2 Library).
WinUI 3 ships as a component of the Windows App SDK framework package, with Visual Studio project templates in the Windows App SDK Visual Studio Extension (VSIX).	Part of WinUI 2 ships within the operating system itself (the Windows.UI.* family of UWP WinRT APIs) and part of it ships as a library ("WinUI 2") with additional controls, elements and the latest styles on top of what's already included in the operating system itself. With WinUI 2, these features ship in a downloadable NuGet package. However, other significant parts of the UI stack are still built-in to the OS, such as the core XAML framework, input, and composition layers.
WinUI 3 supports C# (.NET 6 and later) and C++ for desktop apps.	WinUI 2 supports C# and Visual Basic (.NET Native), and C++ apps.
WinUI 3 is supported only in desktop-based projects. To use WinUI 3, UWP projects can migrate their project type to desktop (see how to migrate your UWP app to the Windows App SDK).	WinUI 2 can be incorporated into production UWP apps by installing a NuGet package into a new or existing UWP project. WinUI controls and styles can then be referenced directly in new apps, or by updating "Windows.UI." namespace references to "Microsoft.UI." in existing apps.
WinUI 3 supports the Chromium-based WebView2 control	WinUI 2 supports the WebView control on all devices, and starting with WinUI 2.8, the WebView2 control on Desktop.
WinUI 3 works downlevel to Windows 10 October 2018	WinUI 2.0 - 2.7 works downlevel to Windows 10 Creators Update (Version 1703, OS build 15063). WinUI 2.8 and later

WinUI 3 in the Windows App SDK	WinUI 2 for UWP
Update (Version 1809, OS build 17763).	works downlevel to Windows 10 October 2018 Update (Version 1809, OS build 17763).

See also

- Windows App SDK
- Stable channel release notes for the Windows App SDK
- Windows App SDK and supported Windows releases
- Windows App SDK API docs
- WinUI 2 API docs
- Windows App SDK Samples

WinUI in the Windows App SDK (WinUI 3)

Article • 07/15/2024

WinUI 3 is the native UI platform component that ships with the Windows App SDK (completely decoupled from Windows SDKs). The Windows App SDK provides a unified set of APIs and tools that can be used to create production desktop apps that target Windows 10 and later, and can be published to the Microsoft Store.

> **Important**
>
> The **Windows App SDK** was previously known as **Project Reunion**. Some assets, such as the VSIX extension and NuGet packages, still use this code name (these will be renamed in a future release).
>
> **Windows App SDK** is used in all documentation except where a specific release or asset still refers to **Project Reunion**.

Create your first WinUI 3 project

WinUI resources

Release notes: For more details on the Windows App SDK, see Latest stable channel release notes for the Windows App SDK.

Github: WinUI is hosted on Github. Use the WinUI repo, to file feature requests or bugs, and interact with the WinUI team.

Examples

> **Tip**
>
> For more info, design guidance, and code examples, see **Design and code Windows apps**.
>
> The **WinUI 3 Gallery** and **WinUI 2 Gallery** apps include interactive examples of most WinUI 3 and WinUI 2 controls, features, and functionality.

If installed already, open them by clicking the following links: **WinUI 3 Gallery** or **WinUI 2 Gallery**.

If they are not installed, you can download the **WinUI 3 Gallery** and the **WinUI 2 Gallery** from the Microsoft Store.

You can also get the source code for both from **GitHub** (use the *main* branch for WinUI 3 and the *winui2* branch for WinUI 2).

See also

- Windows App SDK
- Stable release channel for the Windows App SDK
- Windows App SDK API docs
- WinUI 2 API docs
- Windows App SDK Samples

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Build a C# .NET app with WinUI 3 and Win32 interop

Article • 03/06/2025

In this topic, we step through how to build a basic **C# .NET** application with WinUI 3 and Win32 interop capabilities using Platform Invocation Services (PInvoke).

Prerequisites

1. Start developing Windows apps

Basic managed C#/.NET app

For this example, we'll specify the location and size of the app window, convert and scale it for the appropriate DPI, disable the window minimize and maximize buttons, and finally query the current process to show a list of the modules that are loaded into the current process.

We're going to build our example app from the initial template application (see Prerequisites). Also see WinUI 3 templates in Visual Studio.

The MainWindow.xaml file

With WinUI 3, you can create instances of the Window class in XAML markup.

The XAML Window class has been extended to support desktop windows, turning it into an abstraction of each of the low-level window implementations used by the UWP and desktop app models. Specifically, CoreWindow for UWP and window handles (or HWNDs) for Win32.

The following code shows the MainWindow.xaml file from the initial template app, which uses the Window class as the root element for the app.

XAML

```
<Window
    x:Class="WinUI_3_basic_win32_interop.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:WinUI_3_basic_win32_interop"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
VerticalAlignment="Center">
        <Button x:Name="myButton" Click="myButton_Click">Click Me</Button>
    </StackPanel>
</Window>
```

Configuration

1. To call Win32 APIs exported from `User32.dll`, you can use the **C#/Win32 P/Invoke Source Generator** in your Visual Studio project. Click **Tools** > **NuGet Package Manager** > **Manage NuGet Packages for Solution...**, and (on the **Browse** tab) search for *Microsoft.Windows.CsWin32*. For more details, see Calling Native Functions from Managed Code.

   You can optionally confirm that installation was successful by confirming that *Microsoft.Windows.CsWin32* is listed under the **Dependencies** > **Packages** node in Solution Explorer.

   You can also optionally double-click the application project file (or right click and select **Edit project file**) to open the file in a text editor, and confirm that the project file now includes a NuGet `PackageReference` for "Microsoft.Windows.CsWin32".

   XML

   ```
   <Project Sdk="Microsoft.NET.Sdk">
     <PropertyGroup>
       <OutputType>WinExe</OutputType>
       <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
       <TargetPlatformMinVersion>10.0.17763.0</TargetPlatformMinVersion>
       <RootNamespace>WinUI_3_basic_win32_interop</RootNamespace>
       <ApplicationManifest>app.manifest</ApplicationManifest>
       <Platforms>x86;x64;ARM64</Platforms>
       <RuntimeIdentifiers>win-x86;win-x64;win-arm64</RuntimeIdentifiers>
       <PublishProfile>win-$(Platform).pubxml</PublishProfile>
       <UseWinUI>true</UseWinUI>
       <EnableMsixTooling>true</EnableMsixTooling>
       <Nullable>enable</Nullable>
     </PropertyGroup>

     <ItemGroup>
       <Content Include="Assets\SplashScreen.scale-200.png" />
       <Content Include="Assets\LockScreenLogo.scale-200.png" />
       <Content Include="Assets\Square150x150Logo.scale-200.png" />
       <Content Include="Assets\Square44x44Logo.scale-200.png" />
       <Content Include="Assets\Square44x44Logo.targetsize-24_altform-
   unplated.png" />
   ```

```
    <Content Include="Assets\StoreLogo.png" />
    <Content Include="Assets\Wide310x150Logo.scale-200.png" />
  </ItemGroup>

  <ItemGroup>
    <Manifest Include="$(ApplicationManifest)" />
  </ItemGroup>

  <!--
    Defining the "Msix" ProjectCapability here allows the Single-
project MSIX Packaging
    Tools extension to be activated for this project even if the
Windows App SDK Nuget
    package has not yet been restored.
  -->
  <ItemGroup
Condition="'$(DisableMsixProjectCapabilityAddedByProject)'!='true' and
'$(EnableMsixTooling)'=='true'">
    <ProjectCapability Include="Msix" />
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.Windows.CsWin32"
Version="0.3.183">
      <PrivateAssets>all</PrivateAssets>
      <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.Windows.SDK.BuildTools"
Version="10.0.26100.1742" />
    <PackageReference Include="Microsoft.WindowsAppSDK"
Version="1.6.250205002" />
  </ItemGroup>

  <!--
    Defining the "HasPackageAndPublishMenuAddedByProject" property here
allows the Solution
    Explorer "Package and Publish" context menu entry to be enabled for
this project even if
    the Windows App SDK Nuget package has not yet been restored.
  -->
  <PropertyGroup
Condition="'$(DisableHasPackageAndPublishMenuAddedByProject)'!='true'
and '$(EnableMsixTooling)'=='true'">
    <HasPackageAndPublishMenu>true</HasPackageAndPublishMenu>
  </PropertyGroup>

  <!-- Publish Properties -->
  <PropertyGroup>
    <PublishReadyToRun Condition="'$(Configuration)' ==
'Debug'">False</PublishReadyToRun>
    <PublishReadyToRun Condition="'$(Configuration)' !=
'Debug'">True</PublishReadyToRun>
    <PublishTrimmed Condition="'$(Configuration)' ==
'Debug'">False</PublishTrimmed>
    <PublishTrimmed Condition="'$(Configuration)' !=
```

```
'Debug'">True</PublishTrimmed>
  </PropertyGroup>
</Project>
```

2. Add a text file to your project, and name it `NativeMethods.txt`. The contents of this file inform the C#/Win32 P/Invoke Source Generator the functions and types for which you want P/Invoke source code generated. In other words, which functions and types you'll be calling and using in your C# code.

```
C#

GetDpiForWindow
GetWindowLong
SetWindowPos
SetWindowLong
HWND_TOP
WINDOW_STYLE
```

Code

1. In the `App.xaml.cs` code-behind file, we get a handle to the **Window** by using the **WindowNative.GetWindowHandle** WinRT COM interop method (see Retrieve a window handle (HWND)).

   That method is called from the app's **OnLaunched** handler, as shown here:

```
C#

/// <summary>
/// Invoked when the application is launched.
/// </summary>
/// <param name="args">Details about the launch request and process.
</param>
protected override void
OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
{
    m_window = new MainWindow();

    var hwnd = WinRT.Interop.WindowNative.GetWindowHandle(m_window);

    SetWindowDetails(hwnd, 800, 600);

    m_window.Activate();
}
```

2. We then call a `SetWindowDetails` method, passing the Window handle and preferred dimensions.

In this method:

- We call GetDpiForWindow to get the dots per inch (dpi) value for the window (Win32 uses physical pixels, while WinUI 3 uses effective pixels). This dpi value is used to calculate the scale factor, and apply it to the width and height specified for the window.
- We then call SetWindowPos to specify the desired location of the window.
- Finally, we call SetWindowLong to disable the *Minimize* and *Maximize* buttons.

```
C#

private static void SetWindowDetails(IntPtr hwnd, int width, int
height)
{
    var dpi =
Windows.Win32.PInvoke.GetDpiForWindow((Windows.Win32.Foundation.HWND)hw
nd);
    float scalingFactor = (float)dpi / 96;
    width = (int)(width * scalingFactor);
    height = (int)(height * scalingFactor);

    _ =
Windows.Win32.PInvoke.SetWindowPos((Windows.Win32.Foundation.HWND)hwnd,
                                   Windows.Win32.Foundation.HWND.HWND_TOP,
                                   0, 0, width, height,

Windows.Win32.UI.WindowsAndMessaging.SET_WINDOW_POS_FLAGS.SWP_NOMOVE);

    var nIndex =
Windows.Win32.PInvoke.GetWindowLong((Windows.Win32.Foundation.HWND)hwnd
,

Windows.Win32.UI.WindowsAndMessaging.WINDOW_LONG_PTR_INDEX.GWL_STYLE) &
                ~
(int)Windows.Win32.UI.WindowsAndMessaging.WINDOW_STYLE.WS_MINIMIZEBOX &
                ~
(int)Windows.Win32.UI.WindowsAndMessaging.WINDOW_STYLE.WS_MAXIMIZEBOX;

    _ =
Windows.Win32.PInvoke.SetWindowLong((Windows.Win32.Foundation.HWND)hwnd
,

Windows.Win32.UI.WindowsAndMessaging.WINDOW_LONG_PTR_INDEX.GWL_STYLE,
            nIndex);
}
```

3. In the MainWindow.xaml file, we use a ContentDialog with a ScrollViewer to display a list of all the modules loaded for the current process.

XAML

```
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center"
VerticalAlignment="Center">
    <Button x:Name="myButton" Click="myButton_Click">Display loaded
modules</Button>

    <ContentDialog x:Name="contentDialog" CloseButtonText="Close">
        <ScrollViewer>
            <TextBlock x:Name="cdTextBlock" TextWrapping="Wrap" />
        </ScrollViewer>
    </ContentDialog>

</StackPanel>
```

4. We then replace the `MyButton_Click` event handler with the following code.

   Here, we get a reference to the current process by calling GetCurrentProcess. We then iterate through the collection of Modules and append the filename of each ProcessModule to our display string.

   C#

   ```
   private async void myButton_Click(object sender, RoutedEventArgs e)
   {
       myButton.Content = "Clicked";

       var description = new System.Text.StringBuilder();
       var process = System.Diagnostics.Process.GetCurrentProcess();
       foreach (System.Diagnostics.ProcessModule module in
   process.Modules)
       {
           description.AppendLine(module.FileName);
       }

       cdTextBlock.Text = description.ToString();
       await contentDialog.ShowAsync();
   }
   ```

5. Compile and run the app.

6. After the window appears, select the "Display loaded modules" button.

The basic Win32 interop application described in this topic.

Summary

In this topic we covered accessing the underlying window implementation (in this case Win32 and HWNDs) and using Win32 APIs along with the WinRT APIs. This demonstrates how you can use existing desktop application code when creating new WinUI 3 desktop apps.

For a more extensive sample, see the AppWindow gallery sample in the Windows App SDK Samples GitHub repo.

An example to customize the window title bar

In this second example, we show how to customize the window's title bar and its content. Before following along with it, review these topics:

- Install tools for the Windows App SDK.
- Create your first WinUI 3 project.

Create a new project

1. In Visual Studio, create a new C# or C++/WinRT project from the **Blank App, Packaged (WinUI 3 in Desktop)** project template.

Configuration

1. Again, reference the *Microsoft.Windows.CsWin32* NuGet package just like we did in the first example.

2. Add a `NativeMethods.txt` text file to your project.

```
C#

LoadImage
SendMessage
SetWindowText
WM_SETICON
```

MainWindow.xaml

> **Note**
>
> If you need an icon file to use with this walkthrough, then you can download the computer.ico file from the **WirelessHostednetwork** sample app. Place that file in your `Assets` folder, and add the file to your project as content. You'll then be able to refer to the file using the url `Assets/computer.ico`.
>
> Otherwise, feel free to use an icon file that you already have, and change the two references to it in the code listings below.

1. In the code listing below, you'll see that in `MainWindow.xaml` we've added two buttons, and specified **Click** handlers for each. In the **Click** handler for the first button (**basicButton_Click**), we set the title bar icon and text. In the second (**customButton_Click**), we demonstrate more significant customization by replacing the title bar with the content of the **StackPanel** named *customTitleBarPanel*.

```
XAML

<?xml version="1.0" encoding="utf-8"?>
<Window
    x:Class="window_titlebar.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
    xmlns:local="using:window_titlebar"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    Title="Basic WinUI 3 Window title bar sample">

    <Grid x:Name="rootElement" RowDefinitions="100, *, 100, *">

        <StackPanel x:Name="customTitleBarPanel" Grid.Row="0"
Orientation="Horizontal" HorizontalAlignment="Stretch"
VerticalAlignment="Top" Visibility="Collapsed">
            <Image Source="Images/windowIcon.gif" />
            <TextBlock VerticalAlignment="Center" Text="Full customization
of title bar"/>
        </StackPanel>

        <StackPanel x:Name="buttonPanel"  Grid.Row="2"
Orientation="Horizontal" HorizontalAlignment="Center">
            <Button x:Name="basicButton" Click="basicButton_Click"
Margin="25">Set the Window title and icon</Button>
            <Button x:Name="customButton" Click="customButton_Click"
Margin="25">Customize the window title bar</Button>
        </StackPanel>

    </Grid>
</Window>
```

MainWindow.xaml.cs/cpp

1. In the code listing below for the **basicButton_Click** handler—in order to keep the custom title bar hidden—we collapse the *customTitleBarPanel* **StackPanel**, and we set the ExtendsContentIntoTitleBar property to `false`.
2. We then call **IWindowNative::get_WindowHandle** (for C#, using the interop helper method **GetWindowHandle**) to retrieve the window handle (**HWND**) of the main window.
3. Next, we set the application icon (for C#, using the PInvoke.User32 NuGet package) by calling the LoadImage and SendMessage functions.
4. Finally, we call SetWindowText to update the title bar string.

C#

```
private void basicButton_Click(object sender, RoutedEventArgs e)
{
    // Ensure the custom title bar content is not displayed.
    customTitleBarPanel.Visibility = Visibility.Collapsed;

    // Disable custom title bar content.
    ExtendsContentIntoTitleBar = false;
```

```
    //Get the Window's HWND
    var hwnd = WinRT.Interop.WindowNative.GetWindowHandle(this);

    var hIcon = Windows.Win32.PInvoke.LoadImage(
        null,
        "Images/windowIcon.ico",
        Windows.Win32.UI.WindowsAndMessaging.GDI_IMAGE_TYPE.IMAGE_ICON,
        20, 20,
        Windows.Win32.UI.WindowsAndMessaging.IMAGE_FLAGS.LR_LOADFROMFILE);

    Windows.Win32.PInvoke.SendMessage(
        (Windows.Win32.Foundation.HWND)hwnd,
        Windows.Win32.PInvoke.WM_SETICON,
        (Windows.Win32.Foundation.WPARAM)0,
        (Windows.Win32.Foundation.LPARAM)hIcon.DangerousGetHandle());

    Windows.Win32.PInvoke.SetWindowText((Windows.Win32.Foundation.HWND)hwnd,
"Basic customization of title bar");
}
```

C++/WinRT

```
// pch.h
...
#include <microsoft.ui.xaml.window.h>
...

// MainWindow.xaml.h
...
void basicButton_Click(Windows::Foundation::IInspectable const& sender,
Microsoft::UI::Xaml::RoutedEventArgs const& args);
...

// MainWindow.xaml.cpp
void MainWindow::basicButton_Click(IInspectable const&, RoutedEventArgs
const&)
{
    // Ensure the that custom title bar content is not displayed.
    customTitleBarPanel().Visibility(Visibility::Collapsed);

    // Disable custom title bar content.
    ExtendsContentIntoTitleBar(false);

    // Get the window's HWND
    auto windowNative{ this->m_inner.as<::IWindowNative>() };
    HWND hWnd{ 0 };
    windowNative->get_WindowHandle(&hWnd);

    HICON icon{ reinterpret_cast<HICON>(::LoadImage(nullptr,
L"Assets/computer.ico", IMAGE_ICON, 0, 0, LR_DEFAULTSIZE | LR_LOADFROMFILE))
};
    ::SendMessage(hWnd, WM_SETICON, 0, (LPARAM)icon);
```

```
    this->Title(L"Basic customization of title bar");
}
```

5. In the **customButton_Click** handler, we set the visibility of the *customTitleBarPanel* **StackPanel** to **Visible**.
6. We then set the ExtendsContentIntoTitleBar property to `true`, and call SetTitleBar to display the *customTitleBarPanel* **StackPanel** as our custom title bar.

C#

```
private void customButton_Click(object sender, RoutedEventArgs e)
{
    customTitleBarPanel.Visibility = Visibility.Visible;

    // Enable custom title bar content.
    ExtendsContentIntoTitleBar = true;
    // Set the content of the custom title bar.
    SetTitleBar(customTitleBarPanel);
}
```

C++/WinRT

```
// MainWindow.xaml.h
...
void customButton_Click(Windows::Foundation::IInspectable const& sender,
Microsoft::UI::Xaml::RoutedEventArgs const& args);
...

// MainWindow.xaml.cpp
void MainWindow::customButton_Click(IInspectable const&, RoutedEventArgs
const&)
{
    customTitleBarPanel().Visibility(Visibility::Visible);

    // Enable custom title bar content.
    ExtendsContentIntoTitleBar(true);

    // Set the content of the custom title bar.
    SetTitleBar(customTitleBarPanel());
}
```

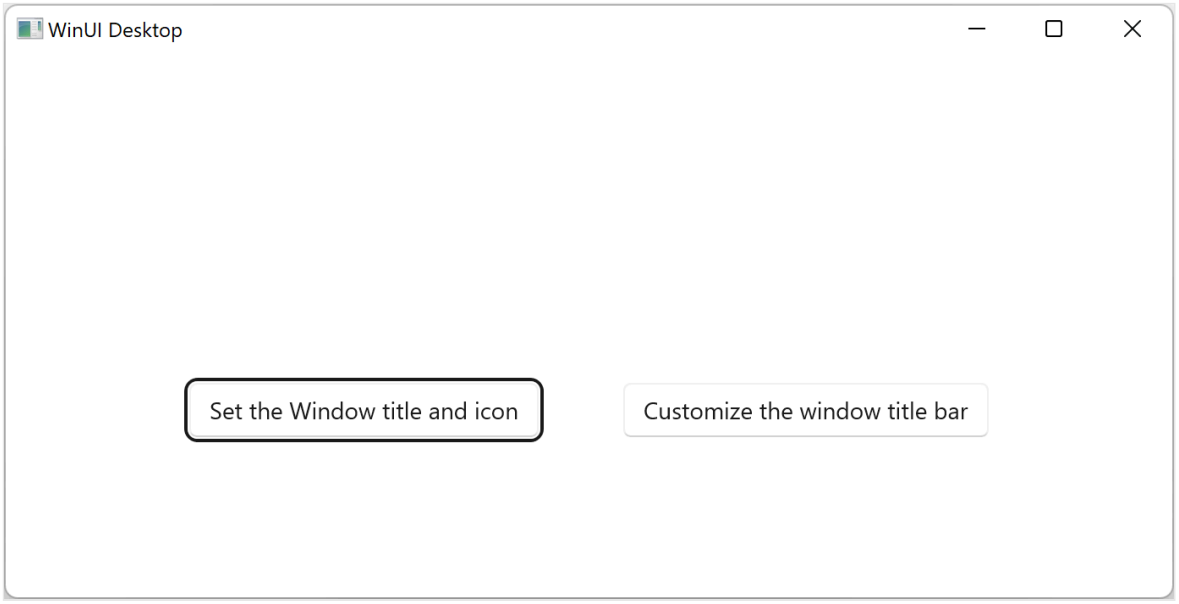
App.xaml

1. In the `App.xaml` file, immediately after the `<!-- Other app resources here -->` comment, we've added some custom-colored brushes for the title bar, as shown below.

C#

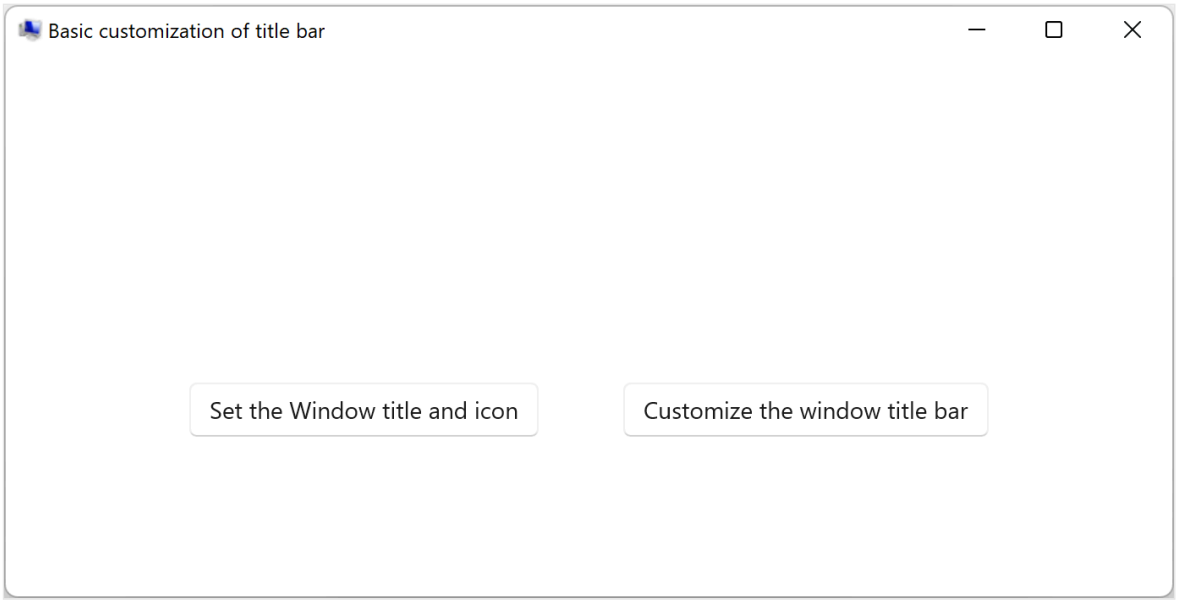
```
<?xml version="1.0" encoding="utf-8"?>
<Application
    x:Class="window_titlebar.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:window_titlebar">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <XamlControlsResources
xmlns="using:Microsoft.UI.Xaml.Controls" />
                <!-- Other merged dictionaries here -->
            </ResourceDictionary.MergedDictionaries>
            <!-- Other app resources here -->
            <SolidColorBrush
x:Key="WindowCaptionBackground">Green</SolidColorBrush>
            <SolidColorBrush
x:Key="WindowCaptionBackgroundDisabled">LightGreen</SolidColorBrush>
            <SolidColorBrush
x:Key="WindowCaptionForeground">Red</SolidColorBrush>
            <SolidColorBrush
x:Key="WindowCaptionForegroundDisabled">Pink</SolidColorBrush>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

1. If you've been following along with these steps in your own app, then you can build your project now, and run the app. You'll see an application window similar to the following (with the custom app icon):



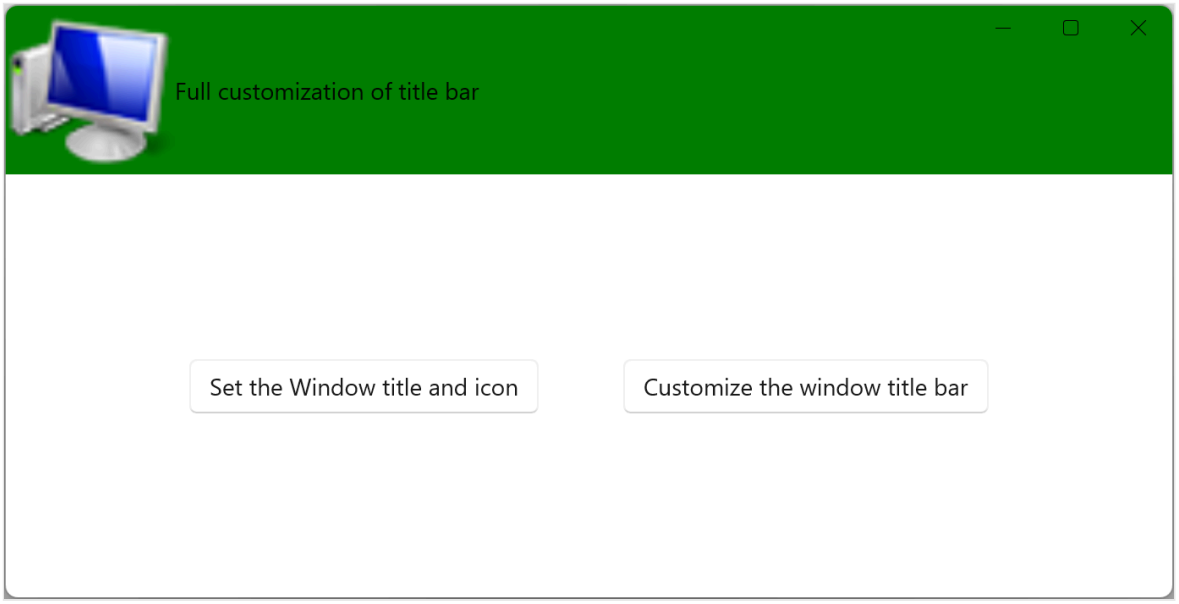
Template app.

- Here's the basic custom title bar:



Template app with custom application icon.

- Here's the fully custom title bar:



Template app with custom title bar.

See also

- Windows App SDK
- Stable release channel for the Windows App SDK
- Manage app windows
- Windows App SDK Samples

Build XAML controls with C++/WinRT

Article • 07/09/2024

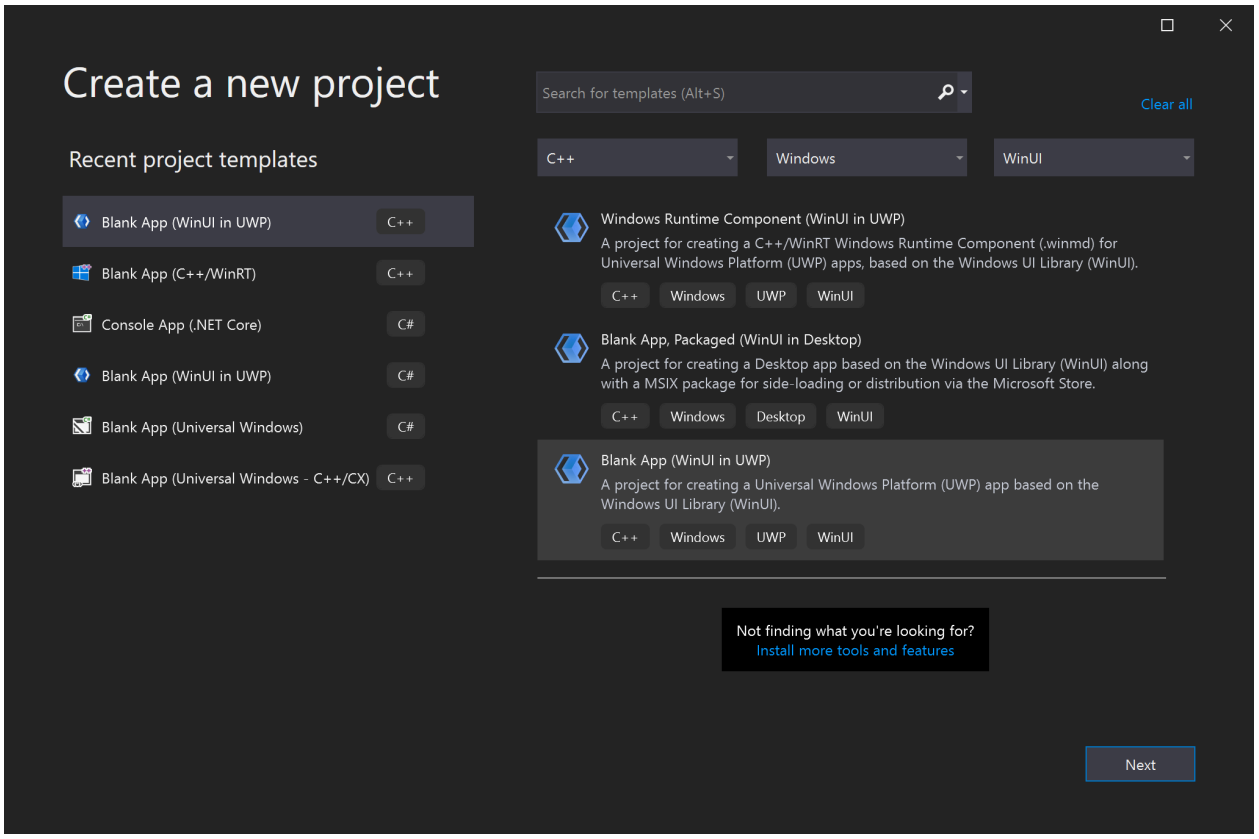
This article walks you through creating a templated XAML control for WinUI 3 with C++/WinRT. Templated controls inherit from **Microsoft.UI.Xaml.Controls.Control** and have visual structure and visual behavior that can be customized using XAML control templates. This article describes the same scenario as the article XAML custom (templated) controls with C++/WinRT but has been adapted to use WinUI 3.

Prerequisites

1. Start developing Windows apps
2. Download and install the latest version of the C++/WinRT Visual Studio Extension (VSIX)

Create a Blank App (BgLabelControlApp)

Begin by creating a new project in Microsoft Visual Studio. In the `Create a new project` dialog, select the **Blank App (WinUI in UWP)** project template, making sure to select the C++ language version. Set the project name to "BgLabelControlApp" so that the file names align with the code in the examples below. Set the **Target version** to Windows 10, version 1903 (build 18362) and **Minimum version** to Windows 10, version 1803 (build 17134). This walkthrough will also work for desktop apps created with the **Blank App, Packaged (WinUI in Desktop)** project template, just make sure to perform all of the steps in the **BgLabelControlApp (Desktop)** project.



Add a templated control to your app

To add a templated control, click the **Project** menu in the toolbar or right-click your project in **Solution Explorer** and select **Add New Item** . Under **Visual C++->WinUI** select the **Custom Control (WinUI)** template. Name the new control "BgLabelControl" and click *Add*. This will add three new files to your project. `BgLabelControl.h` is the header containing the control declarations and `BgLabelControl.cpp` contains the C++/WinRT implementation of the control. `BgLabelControl.idl` is the Interface Definition file that allows the control to be instantiated as a runtime class.

Implement the BgLabelControl custom control class

In the following steps you will update the code in the `BgLabelControl.idl`, `BgLabelControl.h`, and `BgLabelControl.cpp` files in the project directory to implement the runtime class.

The templated control class will be instantiated from XAML markup, and for that reason it's going to be a runtime class. When you build the finished project, the MIDL compiler (midl.exe) will use the `BgLabelControl.idl` file to a generate the Windows Runtime metadata file (.winmd) for your control, which consumers of your component will

reference. For more information on creating runtime classes, see Author APIs with C++/WinRT.

The templated control we are creating will expose a single property which is a string that will be used as a label for the control. Replace the contents of `BgLabelControl.idl` with the following code..

C++/WinRT

```
// BgLabelControl.idl
namespace BgLabelControlApp
{
    runtimeclass BgLabelControl : Microsoft.UI.Xaml.Controls.Control
    {
        BgLabelControl();
        static Microsoft.UI.Xaml.DependencyProperty LabelProperty{ get; };
        String Label;
    }
}
```

The listing above shows the pattern that you follow when declaring a dependency property (DP). There are two pieces to each DP. First, you declare a read-only static property of type DependencyProperty. It has the name of your DP plus Property. You'll use this static property in your implementation. Second, you declare a read-write instance property with the type and name of your DP. If you wish to author an attached property (rather than a DP), then see the code examples in Custom attached properties.

Note that the XAML classes referenced in the code above are in Microsoft.UI.Xaml namespaces. This is what distinguishes them as WinUI controls as opposed to UWP XAML controls, which are defined in Windows.UI.XAML namespaces.

Replace the contents of BgLabelControl.h with the following code.

C++/WinRT

```
// BgLabelControl.h
#pragma once
#include "BgLabelControl.g.h"

namespace winrt::BgLabelControlApp::implementation
{
    struct BgLabelControl : BgLabelControlT<BgLabelControl>
    {
        BgLabelControl() {
DefaultStyleKey(winrt::box_value(L"BgLabelControlApp.BgLabelControl")); }

        winrt::hstring Label()
        {
```

```
            return winrt::unbox_value<winrt::hstring>
(GetValue(m_labelProperty));
        }

        void Label(winrt::hstring const& value)
        {
            SetValue(m_labelProperty, winrt::box_value(value));
        }

        static Microsoft::UI::Xaml::DependencyProperty LabelProperty() {
return m_labelProperty; }

        static void OnLabelChanged(Microsoft::UI::Xaml::DependencyObject
const&, Microsoft::UI::Xaml::DependencyPropertyChangedEventArgs const&);

    private:
        static Microsoft::UI::Xaml::DependencyProperty m_labelProperty;
    };
}
namespace winrt::BgLabelControlApp::factory_implementation
{
    struct BgLabelControl : BgLabelControlT<BgLabelControl,
implementation::BgLabelControl>
    {
    };
}
```

The code shown above implements the **Label** and **LabelProperty** properties, add a static event handler named **OnLabelChanged** to process changes to the value of the dependency property, and adds a private member to store the backing field for **LabelProperty**. Again, note that the XAML classes referenced in the header file are in the Microsoft.UI.Xaml namespaces that belong to the WinUI 3 framework instead of the Windows.UI.Xaml namespaces used by the UWP UI framework.

Next, replace the contents of BgLabelControl.cpp with the following code.

C++/WinRT

```
// BgLabelControl.cpp
#include "pch.h"
#include "BgLabelControl.h"
#if __has_include("BgLabelControl.g.cpp")
#include "BgLabelControl.g.cpp"
#endif

namespace winrt::BgLabelControlApp::implementation
{
    Microsoft::UI::Xaml::DependencyProperty BgLabelControl::m_labelProperty
=
        Microsoft::UI::Xaml::DependencyProperty::Register(
            L"Label",
```

```
            winrt::xaml_typename<winrt::hstring>(),
            winrt::xaml_typename<BgLabelControlApp::BgLabelControl>(),
            Microsoft::UI::Xaml::PropertyMetadata{
winrt::box_value(L"default label"),
Microsoft::UI::Xaml::PropertyChangedCallback{
&BgLabelControl::OnLabelChanged } }
    );

    void
BgLabelControl::OnLabelChanged(Microsoft::UI::Xaml::DependencyObject const&
d, Microsoft::UI::Xaml::DependencyPropertyChangedEventArgs const& /* e */)
    {
        if (BgLabelControlApp::BgLabelControl theControl{
d.try_as<BgLabelControlApp::BgLabelControl>() })
        {
            // Call members of the projected type via theControl.

            BgLabelControlApp::implementation::BgLabelControl* ptr{
winrt::get_self<BgLabelControlApp::implementation::BgLabelControl>
(theControl) };
            // Call members of the implementation type via ptr.
        }
    }
}
```

This walkthrough won't use the **OnLabelChanged** callback, but it's provided so that you can see how to register a dependency property with a property-changed callback. The implementation of **OnLabelChanged** also shows how to obtain a derived projected type from a base projected type (the base projected type is DependencyObject, in this case). And it shows how to then obtain a pointer to the type that implements the projected type. That second operation will naturally only be possible in the project that implements the projected type (that is, the project that implements the runtime class).

The xaml_typename function is provided by the Windows.UI.Xaml.Interop namespace that is not included by default in the WinUI 3 project template. Add a line to the precompiled header file for your project, `pch.h`, to include the header file associated with this namespace.

C++/WinRT

```
// pch.h
...
#include <winrt/Windows.UI.Xaml.Interop.h>
...
```

Define the default style for BgLabelControl

In its constructor, **BgLabelControl** sets a default style key for itself. A templated control needs to have a default style — containing a default control template — which it can use to render itself with in case the consumer of the control doesn't set a style and/or template. In this section we'll add a markup file to the project containing our default style.

Make sure that **Show All Files** is still toggled on (in **Solution Explorer**). Under your project node, create a new folder (not a filter, but a folder) and name it "Themes". Under `Themes`, add a new item of type **Visual C++ > WinUI > Resource Dictionary (WinUI)**, and name it "Generic.xaml". The folder and file names have to be like this in order for the XAML framework to find the default style for a templated control. Delete the default contents of Generic.xaml, and paste in the markup below.

XAML

```
<!-- \Themes\Generic.xaml -->
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:BgLabelControlApp">

    <Style TargetType="local:BgLabelControl" >
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="local:BgLabelControl">
                    <Grid Width="100" Height="100" Background="
{TemplateBinding Background}">
                        <TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center" Text="{TemplateBinding Label}"/>
                    </Grid>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</ResourceDictionary>
```

In this case, the only property that the default style sets is the control template. The template consists of a square (whose background is bound to the **Background** property that all instances of the XAML **Control** type have), and a text element (whose text is bound to the **BgLabelControl::Label** dependency property).

Add an instance of BgLabelControl to the main UI page

Open `MainWindow.xaml`, which contains the XAML markup for our main UI page. Immediately after the **Button** element (inside the **StackPanel**), add the following markup.

XAML

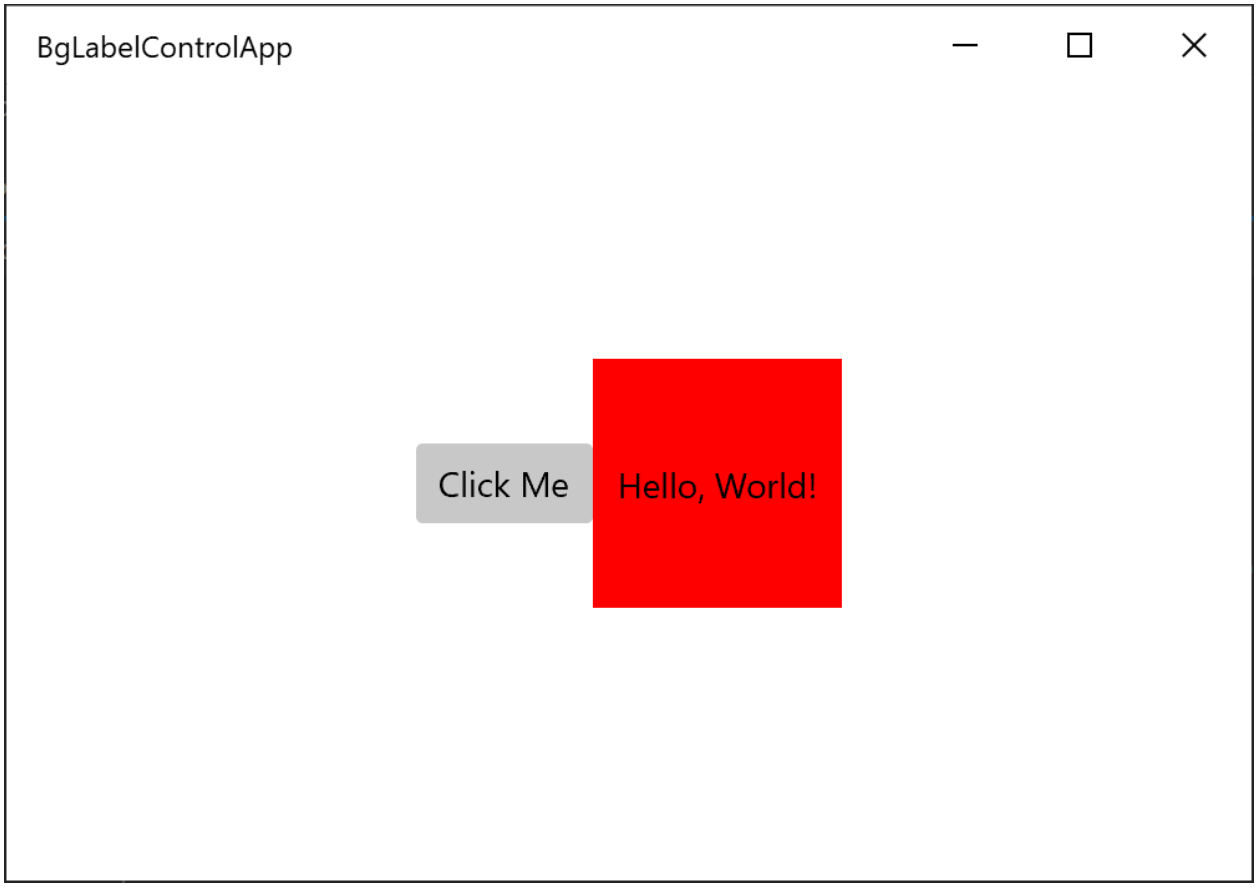
```
<local:BgLabelControl Background="Red" Label="Hello, World!"/>
```

Also, add the following include directive to `MainWindow.h` so that the **MainWindow** type (a combination of compiling XAML markup and imperative code) is aware of the **BgLabelControl** templated control type. If you want to use **BgLabelControl** from another XAML page, then add this same include directive to the header file for that page, too. Or, alternatively, just put a single include directive in your precompiled header file.

C++/WinRT

```
//MainWindow.h
...
#include "BgLabelControl.h"
...
```

Now build and run the project. You'll see that the default control template is binding to the background brush, and to the label, of the **BgLabelControl** instance in the markup.



Implementing overridable functions, such as MeasureOverride and OnApplyTemplate

You derive a templated control from the **Control** runtime class, which itself further derives from base runtime classes. And there are overridable methods of **Control**, **FrameworkElement**, and **UIElement** that you can override in your derived class. Here's a code example showing you how to do that.

C++/WinRT

```
// Control overrides.
void OnPointerPressed(Microsoft::UI::Xaml::Input::PointerRoutedEventArgs
const& /* e */) const { ... };

// FrameworkElement overrides.
Windows::Foundation::Size MeasureOverride(Windows::Foundation::Size const&
/* availableSize */) const { ... };
void OnApplyTemplate() const { ... };

// UIElement overrides.
Microsoft::UI::Xaml::Automation::Peers::AutomationPeer
OnCreateAutomationPeer() const { ... };
```

Overridable functions present themselves differently in different language projections. In C#, for example, overridable functions typically appear as protected virtual functions. In C++/WinRT, they're neither virtual nor protected, but you can still override them and provide your own implementation, as shown above.

Generating the control source files without using a template.

This section shows how you can generate the necessary source files for creating your custom control without using the **Custom Control** item template.

First, add a new Midl File (.idl) item to the project. From the **Project** menu, select **Add New Item...** and type "MIDL" in the search box to find the .idl file item. Name the new file `BgLabelControl.idl` so that the name will be consistent with the steps in this article. Delete the default contents of `BgLabelControl.idl`, and paste in the runtime class declaration shown in the steps above.

After saving the new .idl file, the next step is to generate the Windows Runtime metadata file (.winmd) and stubs for the .cpp and .h implementation files that you will use to implement the templated control. Generate these files by building the solution,

which will cause the MIDL compiler (midl.exe) to compile the .idl file you created. Note that the Solution will not build successfully and Visual Studio will show build errors in the output window, but the necessary files will be generated.

Copy the stub files BgLabelControl.h and BgLabelControl.cpp from \BgLabelControlApp\BgLabelControlApp\Generated Files\sources\ into the project folder. In **Solution Explorer**, make sure Show All Files is toggled on. Right-click the stub files that you copied, and click **Include In Project**.

The compiler places a static_assert line at the top of BgLabelControl.h and BgLabelControl.cpp to prevent the generated files from being compiled. When implementing your control, you should remove these lines from the files you have placed in your project directory. For this walkthrough, you can just overwrite the entire contents of the files with the code provided above.

See also

- Windows App SDK
- Stable release channel for the Windows App SDK
- Windows App SDK Samples

Build XAML controls with C#

Article • 07/12/2024

This article walks you through creating a templated XAML control for WinUI 3 with C#. Templated controls inherit from **Microsoft.UI.Xaml.Controls.Control** and have visual structure and visual behavior that can be customized using XAML control templates.

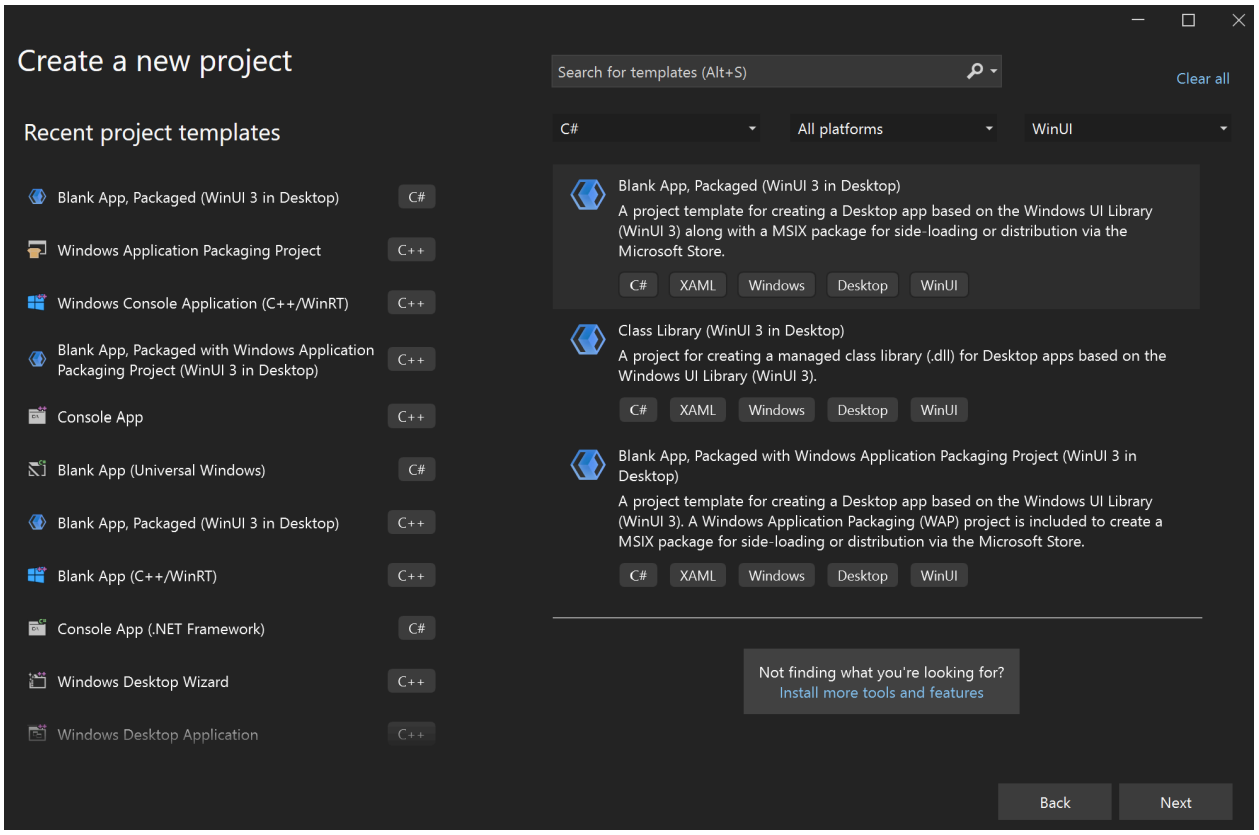
To create standalone WinUI 3 components in C# for consumption from both C# and C++/WinRT apps, see the article Walkthrough: Create a C# component with WinUI 3 controls, and consume it from a C++ Windows App SDK application.

Prerequisites

1. Set up your development environment—see Install tools for the Windows App SDK.
2. Follow the instructions on how to Create your first WinUI 3 project.

Create a Blank App (BgLabelControlApp)

Begin by creating a new project in Microsoft Visual Studio. In the **Create a new project** dialog, select the **Blank App, Packaged (WinUI 3 in Desktop)** project template, making sure to select the C# language version. Set the project name to "BgLabelControlApp" so that the file names align with the code in the examples below.



Add a templated control to your app

To add a templated control, click the **Project** menu in the toolbar or right-click your project in **Solution Explorer** and select **Add New Item** . Under **Visual C#->WinUI** select the **Custom Control (WinUI 3)** template. Name the new control "BgLabelControl" and click *Add*.

Update the custom control C# file

In the C# file, BgLabelControl.cs, note that the constructor defines the **DefaultStyleKey** property for our control. This key identifies the default template that will be used if the consumer of the control doesn't explicitly specify a template. The key value is the *type* of our control. We will see this key in use later when we implement our generic template file.

C#

```
public BgLabelControl()
{
    this.DefaultStyleKey = typeof(BgLabelControl);
}
```

Our templated control will have a text label that can be set programmatically in code, in XAML, or via data binding. In order for the system to keep the text of our control's label

up to date, it needs to be implemented as a DependencyPropety. To do this, first we declare a string property and call it **Label**. Instead of using a backing variable, we set and get the value of our dependency property by calling GetValue and SetValue. These methods are provided by the DependencyObject, which **Microsoft.UI.Xaml.Controls.Control** inherits.

C#

```
public string Label
{
    get => (string)GetValue(LabelProperty);
    set => SetValue(LabelProperty, value);
}
```

Next, declare the dependency property and register it with the system by calling DependencyProperty.Register. This method specifies the name and type of our **Label** property, the type of the owner of the property, our **BgLabelControl** class, and the default value for the property.

C#

```
DependencyProperty LabelProperty = DependencyProperty.Register(
    nameof(Label),
    typeof(string),
    typeof(BgLabelControl),
    new PropertyMetadata(default(string), new
PropertyChangedCallback(OnLabelChanged)));
```

These two steps are all that's required to implement a dependency property, but for this example, we'll add an optional handler for the **OnLabelChanged** event. This event is raised by the system whenever the property value is updated. In this case we check to see if the new label text is an empty string or not and update a class variable accordingly.

C#

```
public bool HasLabelValue { get; set; }

private static void OnLabelChanged(DependencyObject d,
DependencyPropertyChangedEventArgs e)
{
    BgLabelControl labelControl = d as BgLabelControl; //null checks omitted
    String s = e.NewValue as String; //null checks omitted
    if (s == String.Empty)
    {
        labelControl.HasLabelValue = false;
    }
```

```
    else
    {
        labelControl.HasLabelValue = true;
    }
}
```

For more information on how dependency properties work, see Dependency properties overview.

Define the default style for BgLabelControl

A templated control must provide a default style template that is used if the user of the control doesn't explicitly set a style. In this step, we will modify the generic template file for our control.

The generic template file is generated when you add the **Custom Control (WinUI)** to your app. The file is named "Generic.xaml" and is generated in the **Themes** folder in solution explorer. The folder and file names are required in order for the XAML framework to find the default style for a templated control. Delete the default contents of Generic.xaml, and paste in the markup below.

XAML

```
<!-- \Themes\Generic.xaml -->
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:BgLabelControlApp">

    <Style TargetType="local:BgLabelControl" >
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="local:BgLabelControl">
                    <Grid Width="100" Height="100" Background="
{TemplateBinding Background}">
                        <TextBlock HorizontalAlignment="Center"
VerticalAlignment="Center" Text="{TemplateBinding Label}"/>
                    </Grid>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
</ResourceDictionary>
```

In this example you can see that the **TargetType** attribute of the **Style** element is set to our **BgLabelControl** type within the **BgLabelControlApp** namespace. This type is the

same value as we specified above for the **DefaultStyleKey** property in the control's constructor which identifies this as the default style for the control.

The **Text** property of the **TextBlock** in the control template is bound to our control's **Label** dependency property. The property is bound using the TemplateBinding markup extension. This example also binds the **Grid** background to the **Background** dependency property which is inherited from the **Control** class.

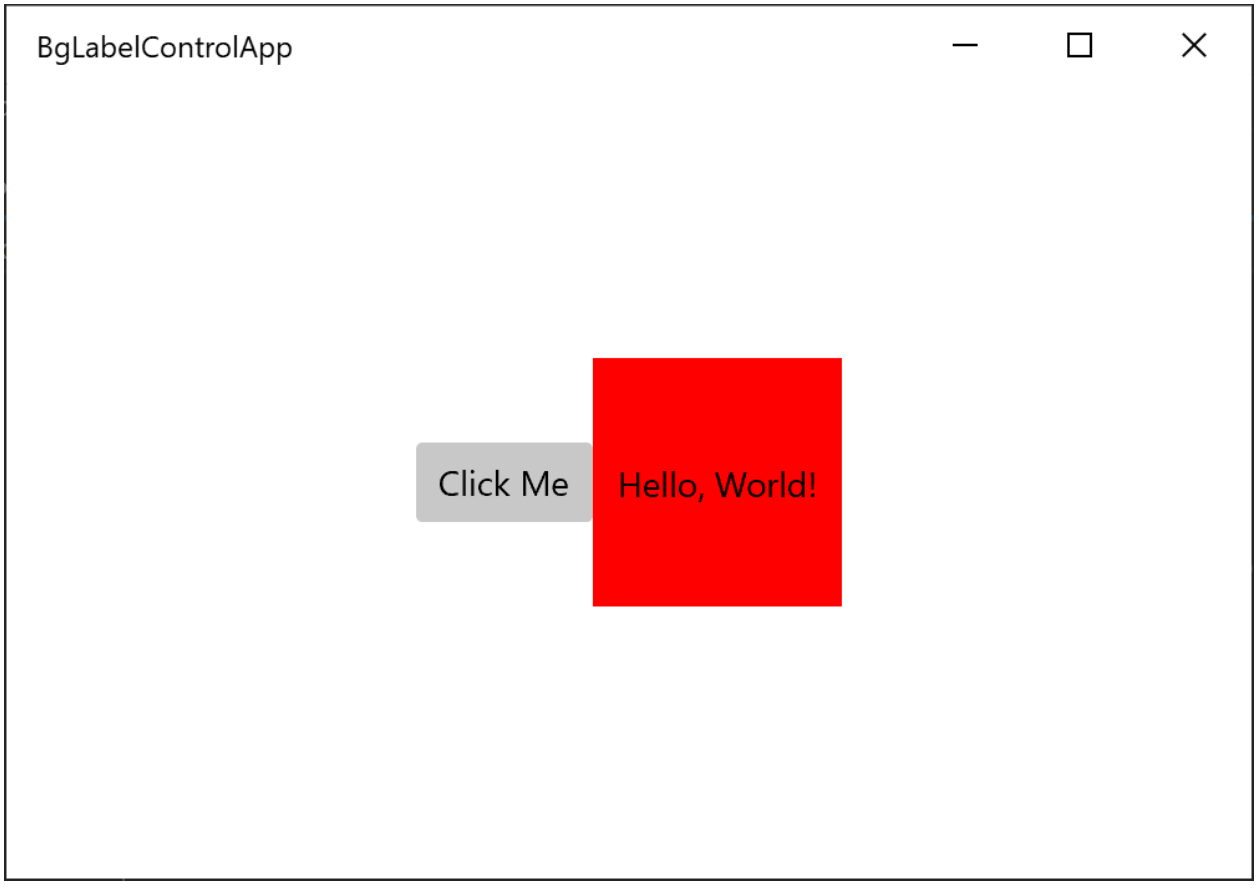
Add an instance of BgLabelControl to the main UI page

Open `MainWindow.xaml`, which contains the XAML markup for our main UI page. Immediately after the **Button** element (inside the **StackPanel**), add the following markup.

XAML

```
<local:BgLabelControl Background="Red" Label="Hello, World!"/>
```

Build and run the app and you will see the templated control, with the background color and label we specified.



See also

- Windows App SDK
- Stable release channel for the Windows App SDK
- Windows App SDK Samples

Localize your WinUI 3 app

Article • 10/24/2022

For more info about the value proposition of localizing your app, see Globalization and localization.

To support multiple languages in a WinUI desktop app, and ensure proper localization of your packaged project, add the appropriate resources to the project (see App resources and the Resource Management System) and declare each supported language in the **package.appxmanifest** file of your project. When you build the project, the specified languages are added to the generated app manifest (**AppxManifest.xml**) and the corresponding resources are used.

> **Note**
>
> As an unpackaged WinUI 3 app (see Create your first WinUI 3 project) doesn't contain a `package.appxmanifest` file, no further action is needed after adding the appropriate resources to the project.

1. Open the .wapproj's `package.appxmanifest` in a text editor and locate the following section:

   XML

   ```
   <Resources>
       <Resource Language="x-generate"/>
   </Resources>
   ```

2. Replace the `<Resource Language="x-generate">` with `<Resource />` elements for each of your supported languages. For example, the following markup specifies that "en-US" and "es-ES" localized resources are available:

   XML

   ```
   <Resources>
       <Resource Language="en-US"/>
       <Resource Language="es-ES"/>
   </Resources>
   ```

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Test apps built with the Windows App SDK and WinUI

Article • 05/31/2024

In this topic we provide some recommendations for how to test and validate functionality in apps created with the Windows App SDK using Windows UI Library (WinUI) user interface (UI) features.

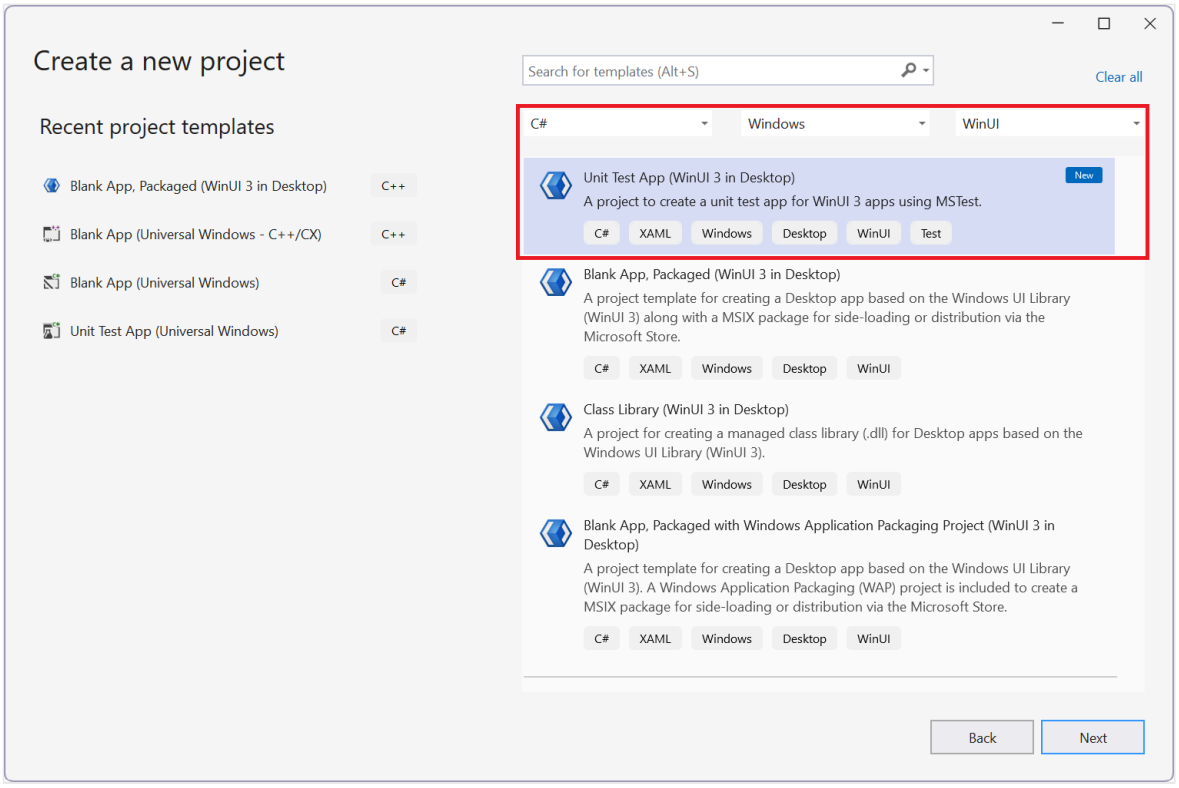
How do I test WinUI functionality in my app?

Most object types under the Microsoft.UI.Xaml namespaces must be used from a UI thread in a Xaml application process. (For details on testing apps created with Windows App SDK that don't use WinUI, see the following section, How do I test non-WinUI functionality in my app?.)

The following steps describe how to use Visual Studio to test code that depends on Microsoft.UI.Xaml types and executes in the context of a Xaml application:

1. Create a unit test project in the same solution as the app you want to test. (This uses MSTest to execute the test code and will initialize a Xaml Window and a Xaml UI Thread.)

   Right click your solution in Solution Explorer, select **Add -> New Project** from the context menu, and choose **Unit Test App (WinUI 3 in Desktop)** for C# or **Unit Test App (WinUI 3)** for C++.



2. Mark your test methods with the `[UITestMethod]` attribute instead of the standard `[TestMethod]` attribute to ensure they execute on the UI thread.

> **Note**
>
> We recommend that you refactor any code to be tested by pulling it out of the main app project and placing it into a library project. Both your app project and your unit test project can then reference that library project.

See Unit tests for Windows UI Library (WinUI) apps in Visual Studio for an example of using this test project.

How do I test non-WinUI functionality in my app?

In many cases, an app includes functionality that does not depend on Microsoft.UI.Xaml types but still needs testing. Various tools are available for testing .NET code, including MSTest, NUnit and xUnit. For more details on testing .NET apps, see Testing in .NET.

In Visual Studio, you can create a new project for any of these testing tools by right clicking your solution in Solution Explorer, selecting **Add -> New Project** from the context menu, choosing **C#** from the **All languages** selector/**Windows** from the **All languages** selector/**Test** from the **All project types** selector, and then picking the

appropriate testing tool from the list (**MSTest Test Project**, **NUnit Test Project** or **xUnit Test Project**).

When creating a new MSTest, NUnit or xUnit project that references a WinUI project, you must:

1. Update the `TargetFramework` in the .csproj file of your testing project. This value must match the `TargetFramework` in the WinUI project (). By default MSTest, NUnit and xUnit projects target the full range of platforms supported by .NET, but a WinUI project only supports Windows and has a more specific TargetFramework.

   For example, if targeting .NET 8, update the TargetFramework of the unit test project from `<TargetFramework>net8.0</TargetFramework>` to `<TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>`.

2. Update the RuntimeIdentifiers in your test project.

   ```
   <RuntimeIdentifiers
   Condition="$([MSBuild]::GetTargetFrameworkVersion('$(TargetFramework)')) &gt;=
   8">win-x86;win-x64;win-arm64</RuntimeIdentifiers>
   ```

   ```
   <RuntimeIdentifiers
   Condition="$([MSBuild]::GetTargetFrameworkVersion('$(TargetFramework)')) &lt;
   8">win10-x86;win10-x64;win10-arm64</RuntimeIdentifiers>
   ```

3. Add the following property to the `PropertyGroup` in .csproj file of your test project to ensure that the test loads the WinAppSDK runtime:
   `<WindowsAppSdkBootstrapInitialize>true</WindowsAppSdkBootstrapInitialize>`

4. Ensure that the Windows App SDK runtime is installed on the machine running the test. For more information on Windows App SDK deployment, see Windows App SDK deployment guide for framework-dependent apps packaged with external location (or unpackaged).

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Unit tests for Windows UI Library (WinUI) apps in Visual Studio

Article • 05/31/2024

This article describes how to create unit tests for WinUI apps in Visual Studio using the built-in unit test project templates.

For a general overview of testing Windows App SDK apps, see Test apps built with the Windows App SDK and WinUI 3.

> **Note**
>
> The unit test app described here is written in the context of a WinUI application. This is required for any tests that execute code requiring the Xaml runtime. This project will create a Xaml UI Thread and execute the tests.

In this tutorial, you learn how to:

- ✓ Create a **Unit Test App (WinUI 3 in Desktop)** for C# or **Unit Test App (WinUI 3)** for C++ project in Visual Studio.
- ✓ Use the Visual Studio **Text Explorer**.
- ✓ Add a **Class Library (WinUI 3 in Desktop)** project for testing.
- ✓ Run tests with the Visual Studio Test Explorer.

Prerequisites

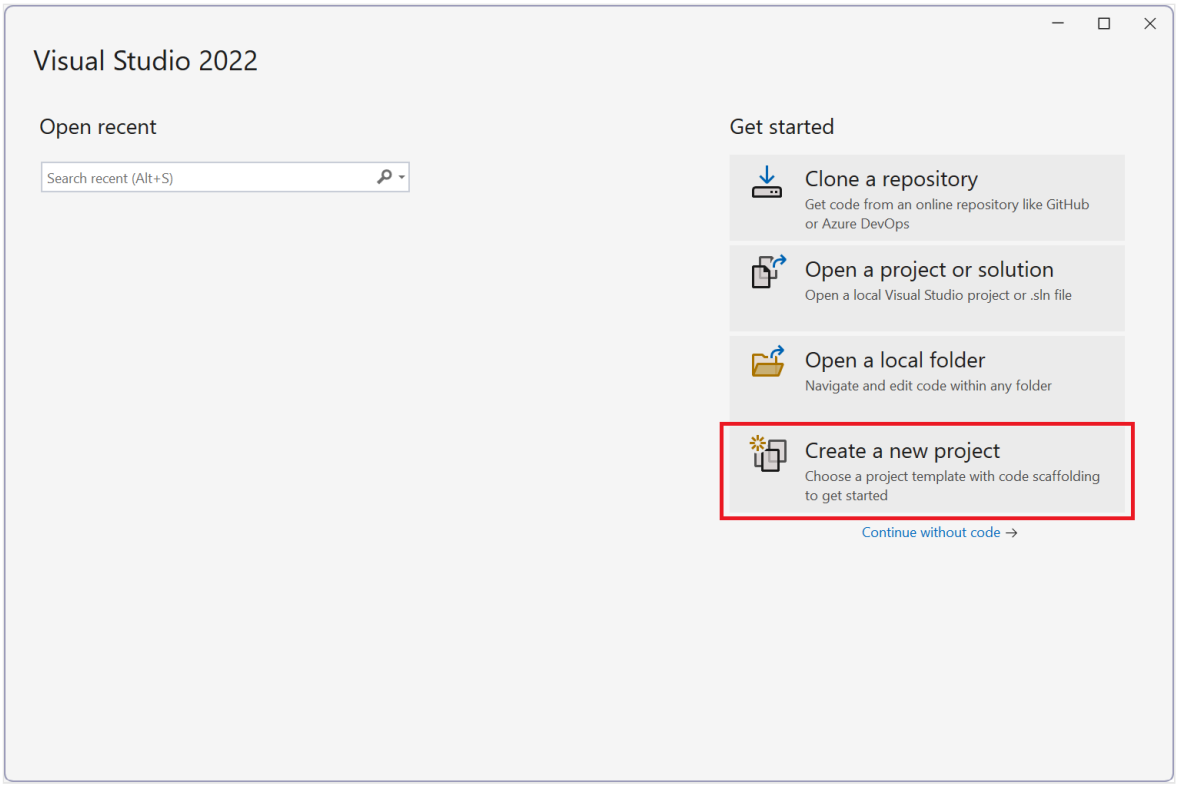
You must have Visual Studio installed.

If you haven't already installed Visual Studio, go to the Visual Studio downloads page to install it for free.

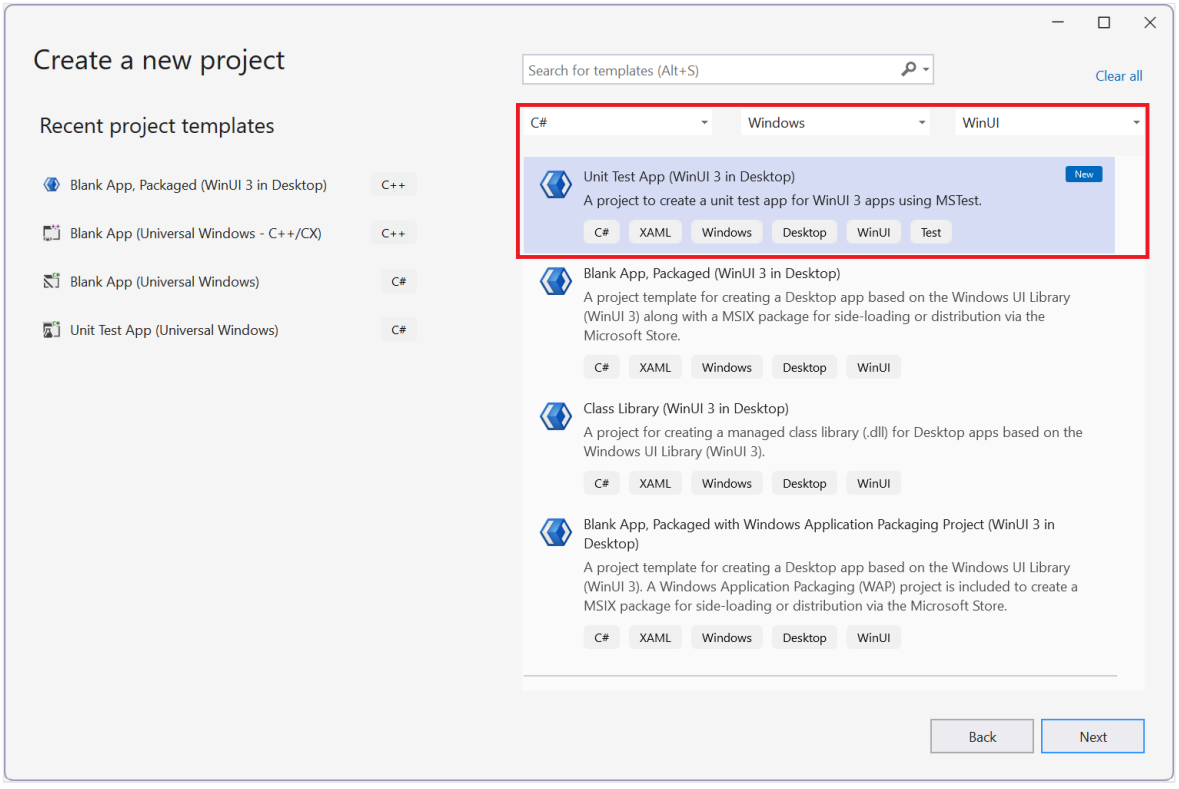
How do I create a Test App (WinUI 3 in Desktop) project

To start, create a unit test project. The project type comes with all the template files you need.

1. Open Visual Studio and select **Create a new project** in the Start window.



2. In the **Create a new project** window, filter projects to **C#**, **Windows**, and **WinUI**, select the **Unit Test App in Desktop (WinUI 3)** template (or **Unit Test App (WinUI 3)** for C++), and then select **Next**



3. [Optional] On the **Configure your new project** window, change the **Project name**, **Solution name** (uncheck **Place solution and project in the same directory**) and the **Location** of your project.

4. Select **Create**.

How do I run tests with Test Explorer

When you create the test project, your tests appear in **Test Explorer**, which is used to run your unit tests. You can also group tests into categories, filter the test list, create, save, and run playlists of tests, debug unit tests and (in Visual Studio Enterprise) analyze code coverage.

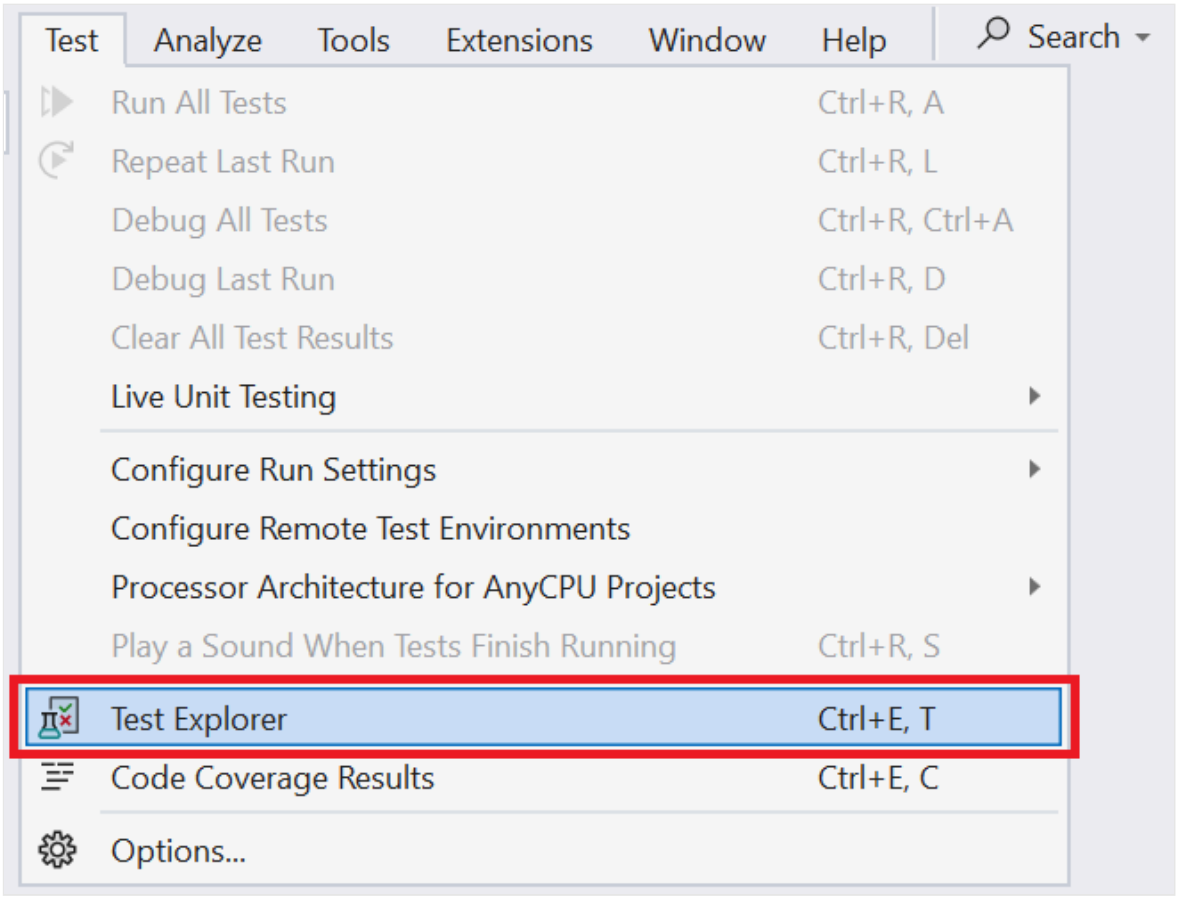
The UnitTests.cs file contains the source code for the unit tests used by **Test Explorer**. By default, the basic sample tests shown here are created automatically:

C#

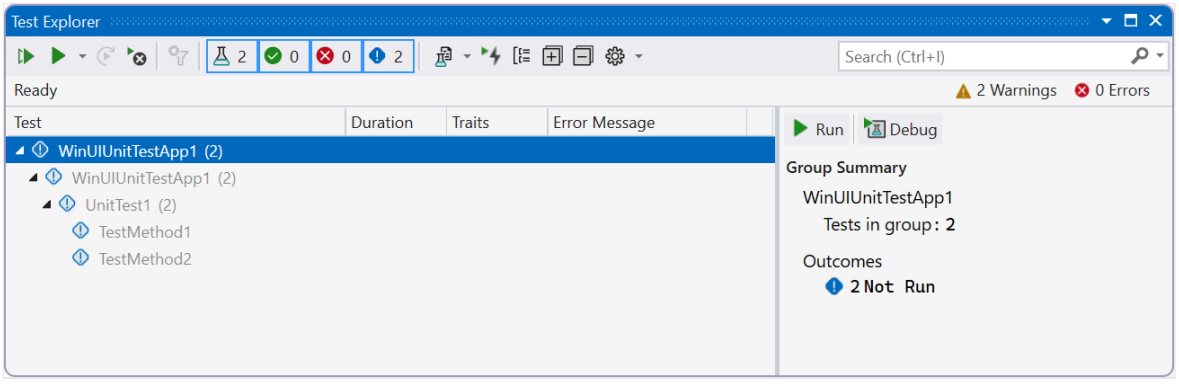
```
namespace WinUITest1
{
   [TestClass]
   public class UnitTest1
   {
      [TestMethod]
      public void TestMethod1()
      {
         Assert.AreEqual(0, 0);
      }

      // Use the UITestMethod attribute for tests that need to run on the UI
thread.
      [UITestMethod]
      public void TestMethod2()
      {
         var grid = new Grid();
         Assert.AreEqual(0, grid.MinWidth);
      }
   }
}
```

1. If you haven't done so already, build your solution. This will allow Visual Studio to "discover" all available tests.

2. Open Test Explorer. If it's not visible, open the **Test** menu, and then choose **Test Explorer** (or press **Ctrl** + **E, T**).

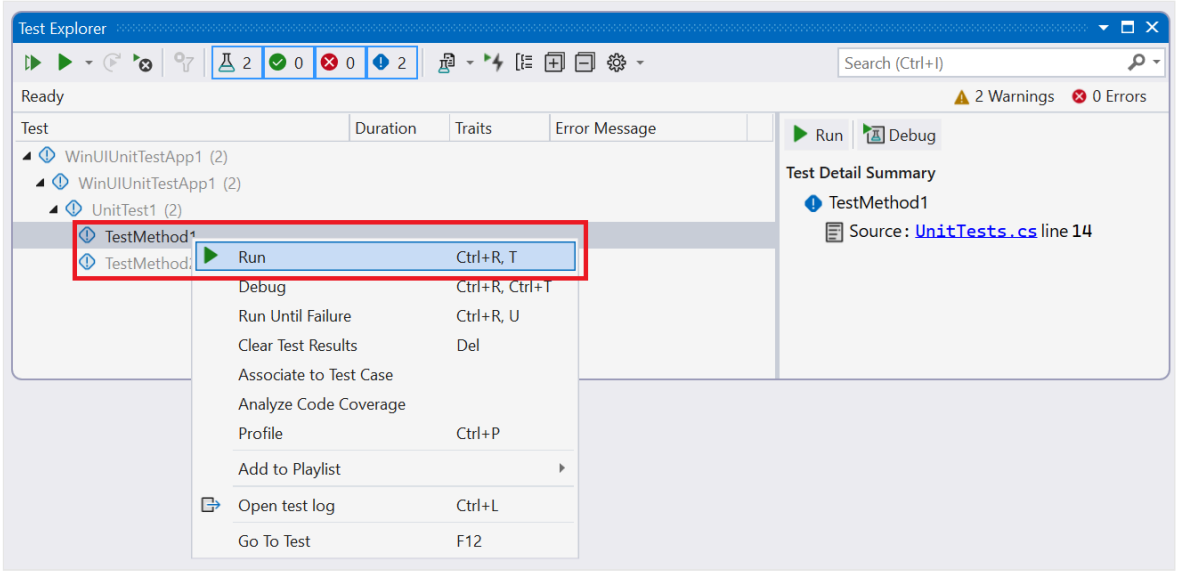


3. View tests. In the **Test Explorer** window, expand all nodes (only the sample tests will be present at this point).

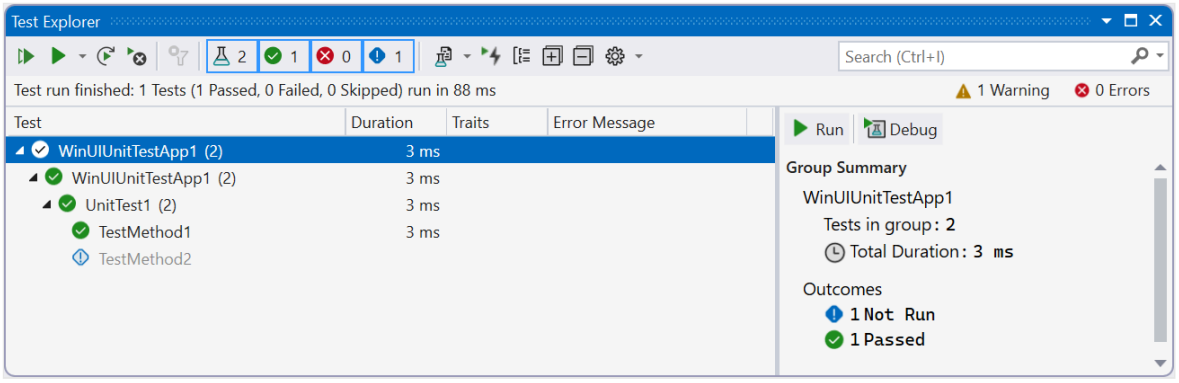


4. Run tests.

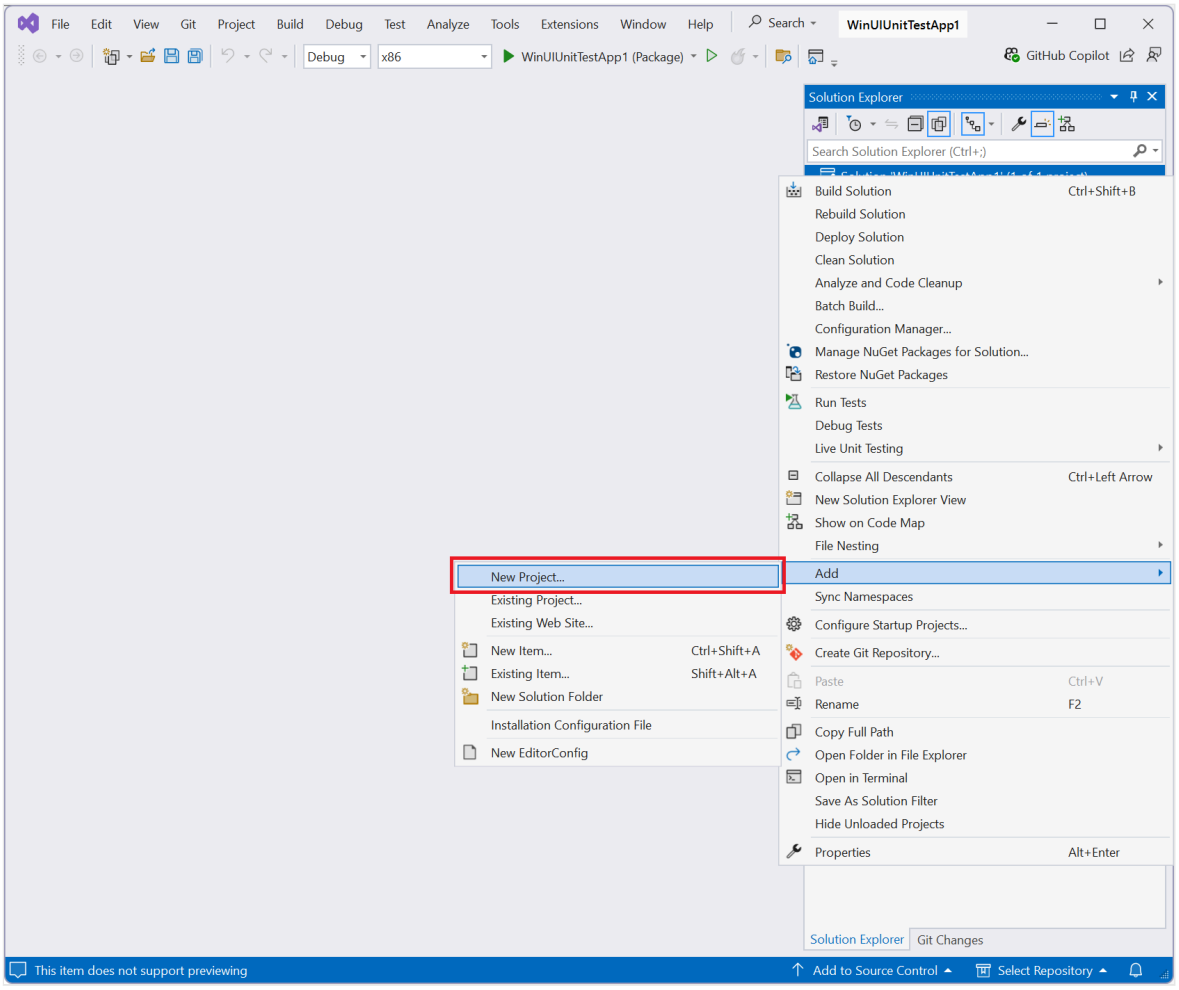
   - Right click on individual test nodes and select **Run**.
   - Select a test and either press the **Play** button or press **Ctrl + R, T**.
   - Press the **Run All Tests In View** button or press **Ctrl + R, V**.



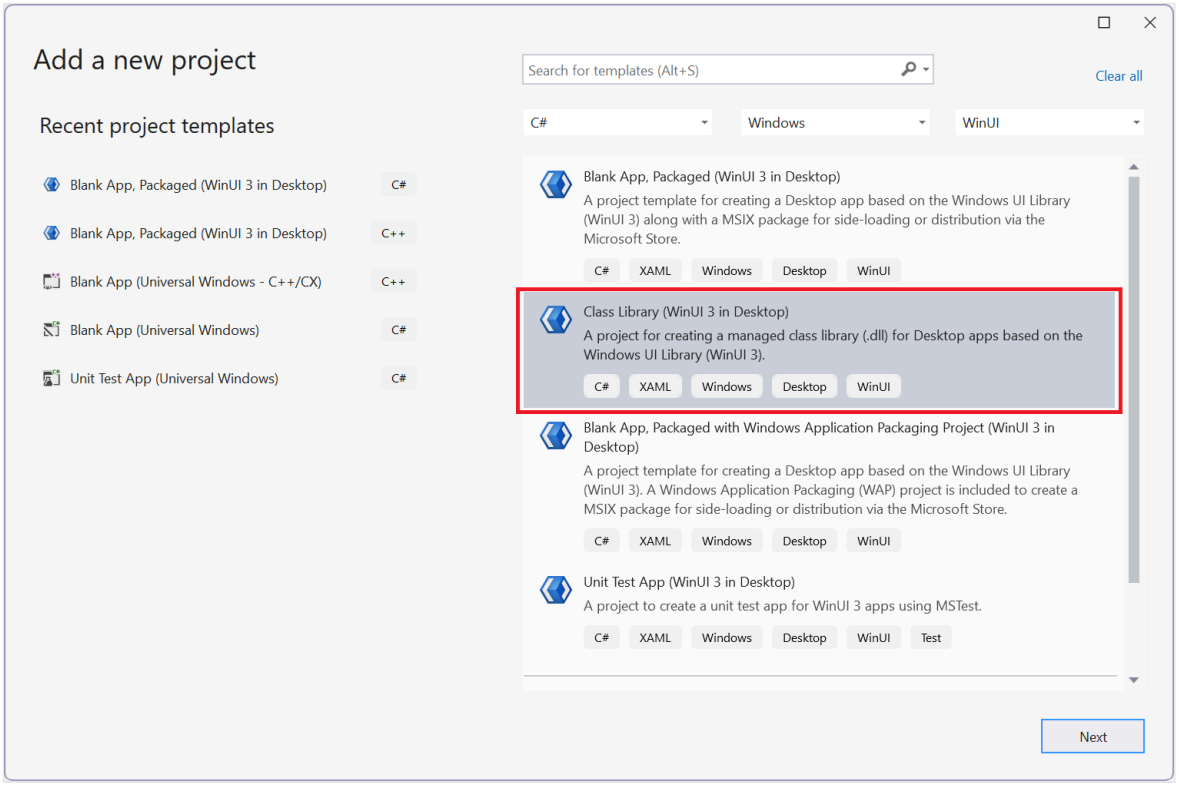
5. Review results. After tests are complete, results are shown in the **Test Explorer** window.



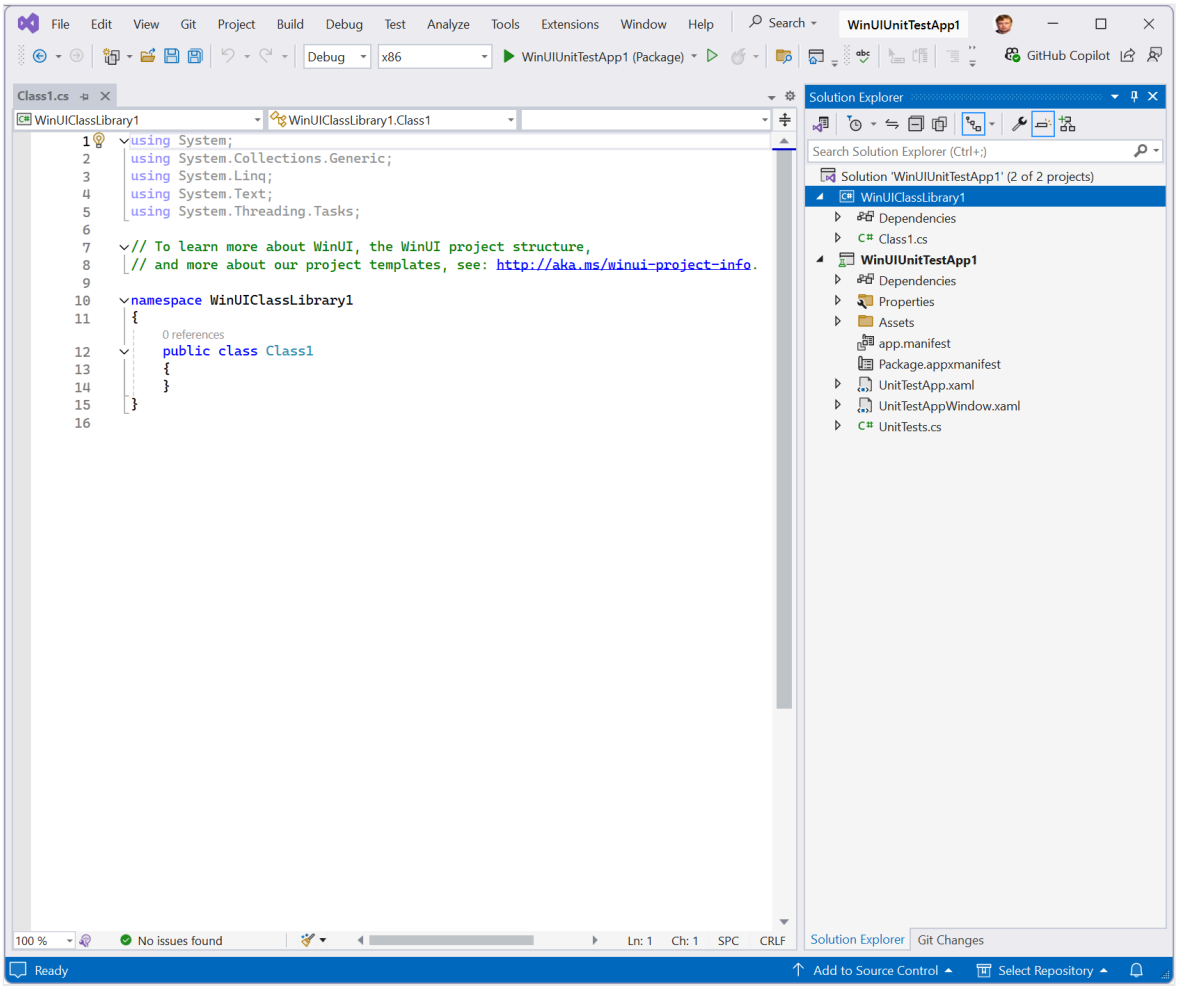
6. Add a new project to the unit test solution. In the Solution Explorer, right click the solution and select **Add -> New Project...**.



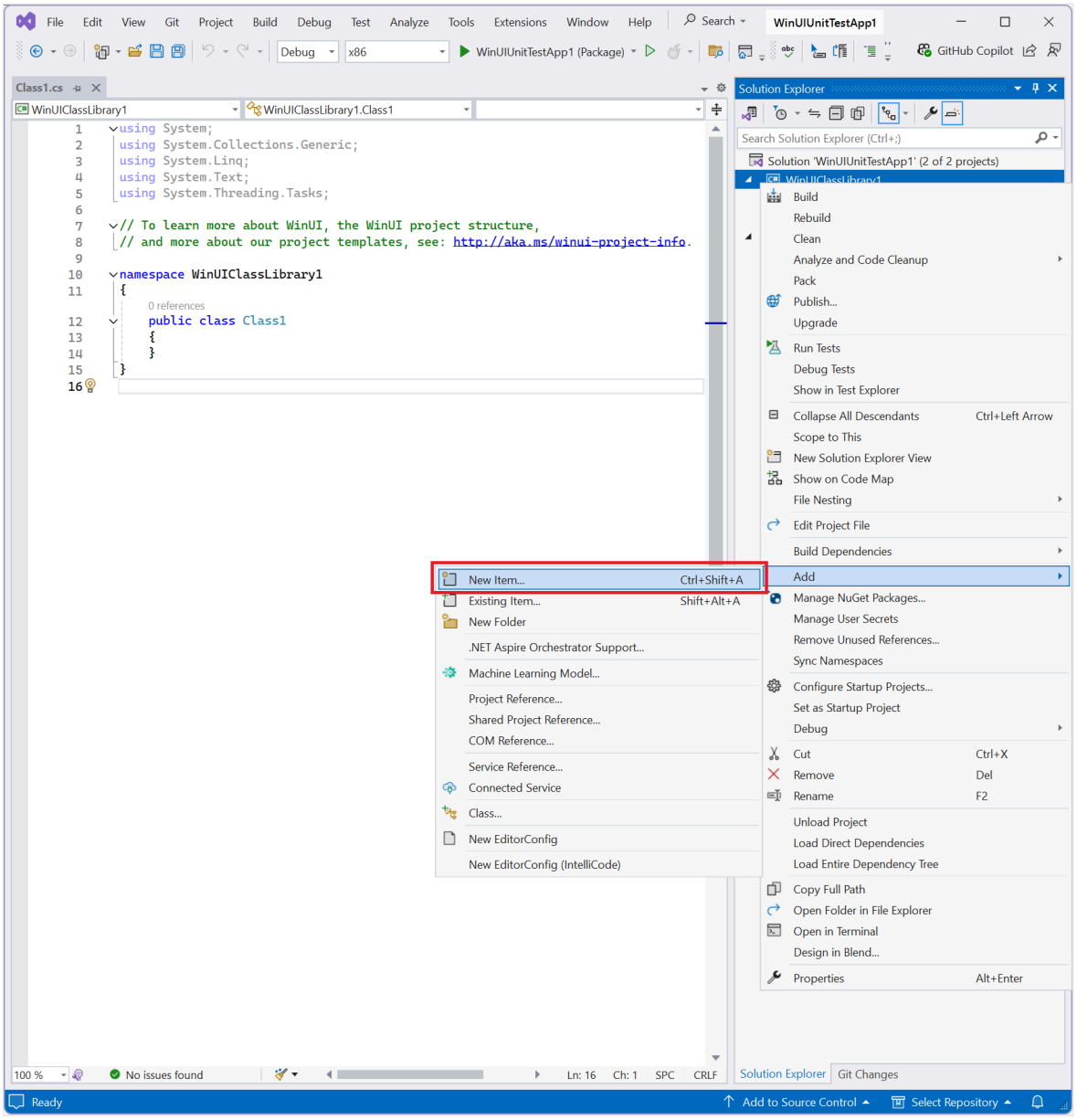
7. For this example, add a WinUI class library project. From the New Project window, filter on C#/Windows/WinUI and select **Class Library (WinUI 3 in Desktop)**.



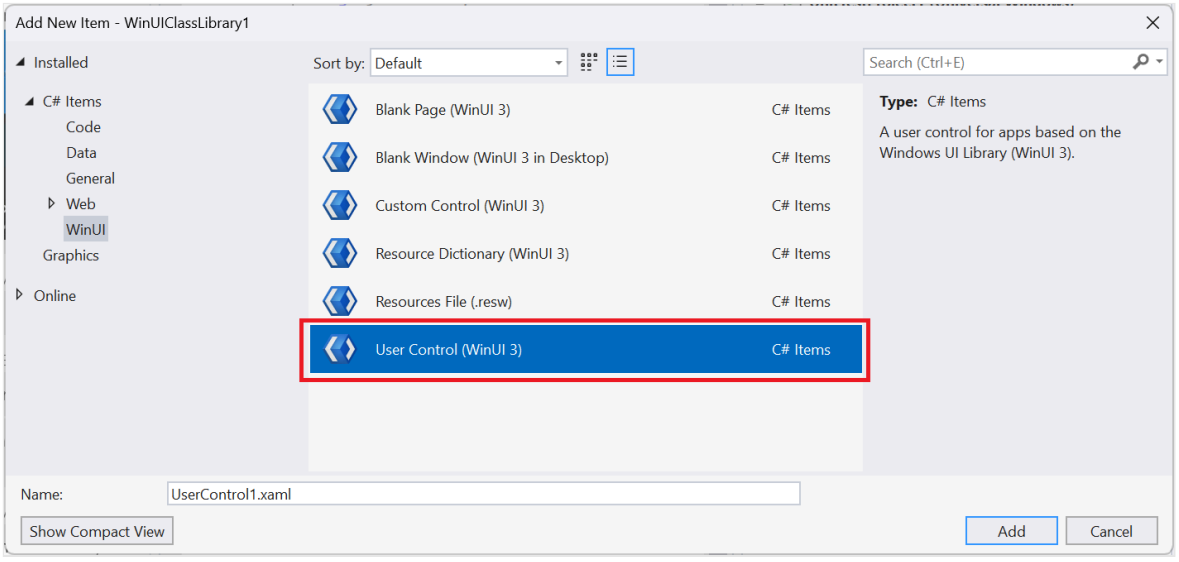
8. Select Next and enter a name for the project (for this example we use "WinUIClassLibrary1") and press **Create**.



9. Add a new UserControl to the project. In the Solution Explorer, right click on the WinUI class library project you just added and select **Add** -> **New Item** from the context menu.



10. In the **Add New Item** window, select the **WinUI** node in the **Installed** items list and then choose **User Control (WinUI 3)** from the results. Name the control "UserControl1".

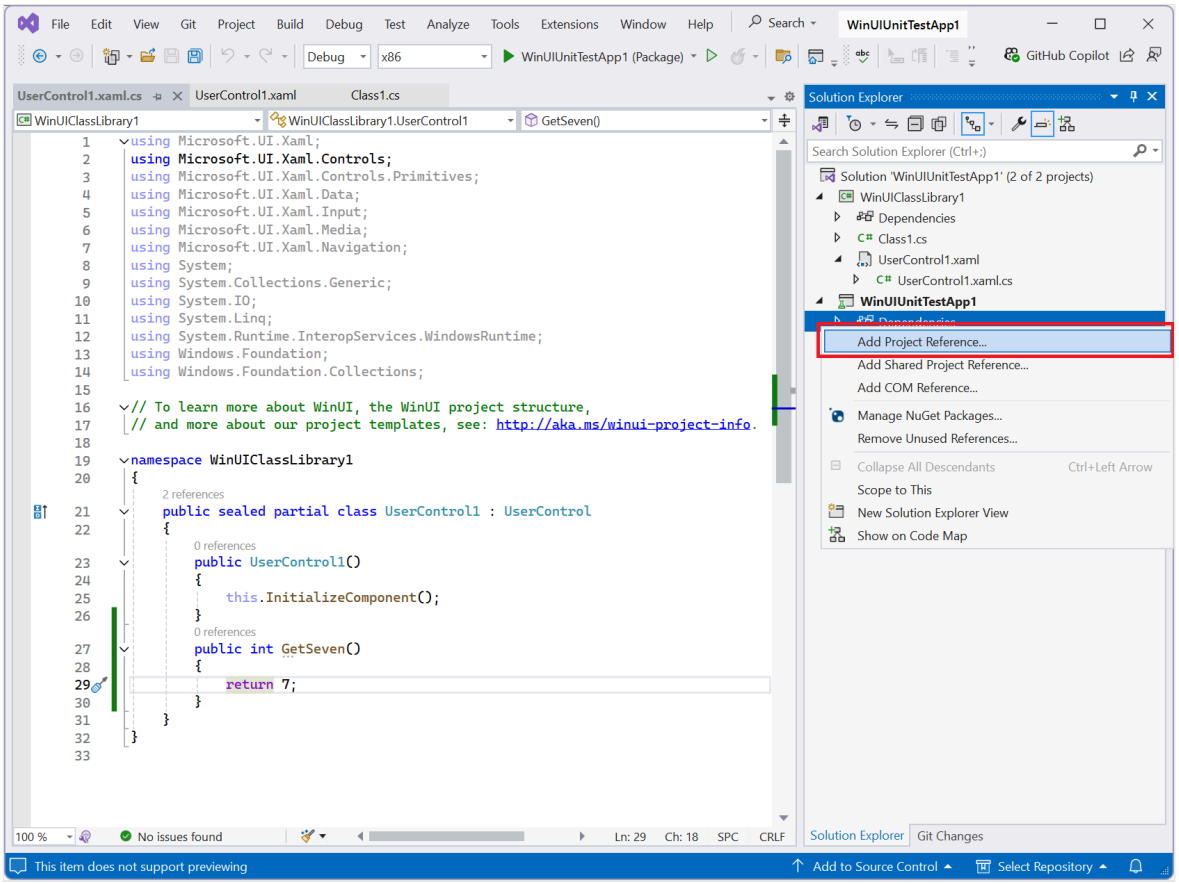


11. Open the UserControl1.xaml.cs code-behind file. For this example, we add a new public method called "GetSeven()" that simply returns an integer.

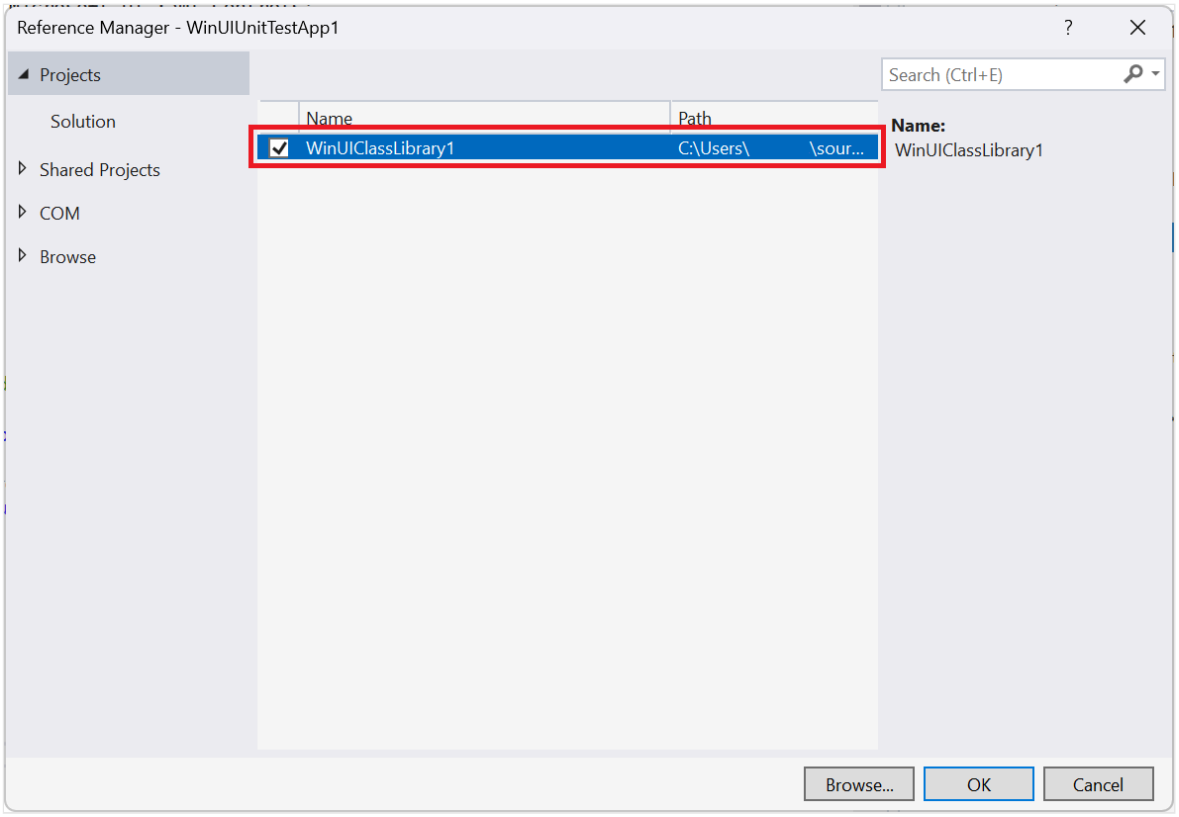
C#

```
namespace WinUIClassLibrary1
{
  public sealed partial class UserControll : UserControl
  {
      public UserControl1()
      {
          this.InitializeComponent();
      }

      public int GetSeven()
      {
          return 7;
      }
  }
}
```

12. Set the WinUI Class Library project as a dependency of the unit test project to enable the use of types from the WinUI class library project. In Solution Explorer, under the class library project, right click on **Dependencies** and select **Add Project Reference**.



Select the "WinUIClassLibrary1" item from the **Projects** list.

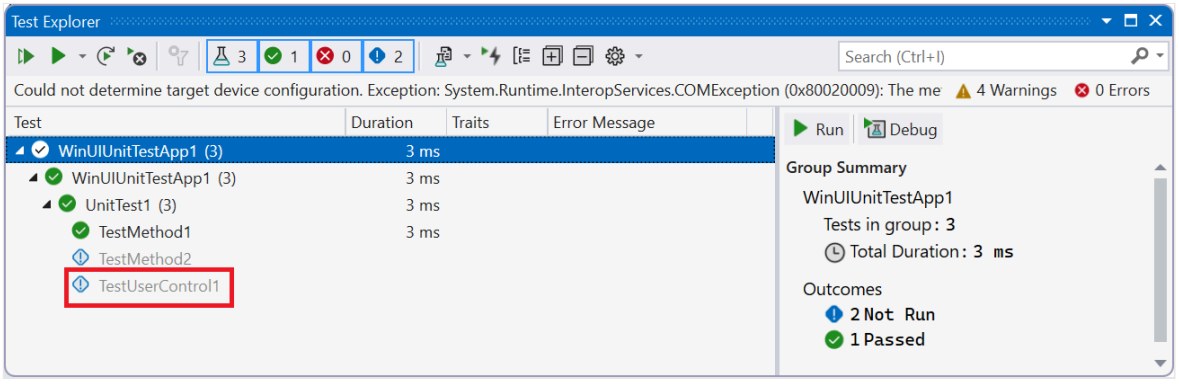


13. Create a new test method in UnitTests.cs. As this test case requires a XAML UI Thread to run, mark it with the [UITestMethod] attribute instead of the standard [TestMethod] attribute.

C#

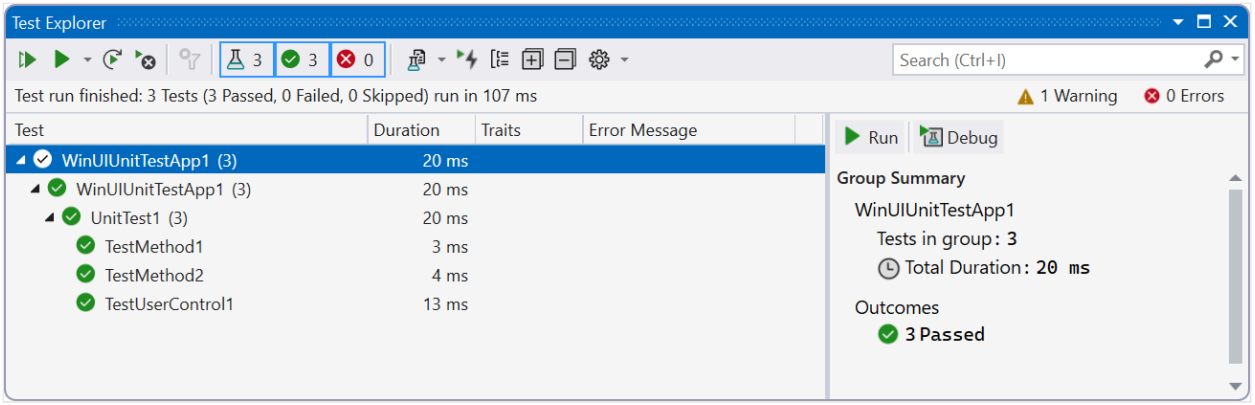
```
[UITestMethod]
public void TestUserControl1()
{
   WinUIClassLibrary1.UserControl1 userControl1 = new
WinUIClassLibrary1.UserControl1();
   Assert.AreEqual(7, userControl1.GetSeven());
}
```

This new test method now appears in **Test Explorer** as one of your unit tests.



14. Run tests.

- Right click the new test node and select **Run**.
- Select the new test and either press the **Play** button or press **Ctrl + R, T**.
- Press the **Run All Tests In View** button or press **Ctrl + R, V**.



Additional resources

Unit test basics

Next steps

In this tutorial, you learned how to:

- ✓ Create a **Test App (WinUI 3 in Desktop)** project in Visual Studio.
- ✓ Use the Visual Studio **Text Explorer**.
- ✓ Add a **Class Library (WinUI 3 in Desktop)** project for testing.
- ✓ Run tests with the Visual Studio Test Explorer.

More comprehensive coverage of the testing tools included with Visual Studio:

Testing tools in Visual Studio

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Build Windows apps with .NET MAUI

Article • 01/23/2025

.NET Multi-platform App UI (.NET MAUI) is a cross-platform framework for creating native mobile and desktop apps with C# and *optionally* XAML. Using .NET MAUI, you can develop apps that can run on Windows, Android, iOS, macOS, and Samsung Tizen from a single shared code-base. If you build a Windows app with .NET MAUI, it will use WinUI 3 as its native platform, and therefore run on Windows 10 version 1809 or later and Windows 11.

Why use .NET MAUI on Windows?

Building apps for Windows with .NET MAUI provides several benefits:

- **Native on Windows**: .NET MAUI creates a WinUI app when targeting Windows. This means that your .NET MAUI app will provide the same user experience on Windows as your Windows App SDK applications.
- **User interface in code**: The .NET MAUI Community Toolkit includes C# Markup, a set of fluent helper methods for building user interfaces in C# code. This enables a separation of concerns between the UI and the business logic of your app, just as with XAML. C# Markup is available on all platforms supported by .NET MAUI.
- **Cross-platform**: Take your Windows apps to all supported platforms, including Android, iOS, macOS, and Samsung Tizen devices.
- **Simplicity**: Develop in a single shared project that can target every platform supported by .NET MAUI.
- **Hot Reload**: Save time while debugging with .NET Hot Reload and XAML Hot Reload support in .NET MAUI. Make edits while the app is running and the changes are automatically applied.
- **Native APIs**: .NET MAUI provides cross-platform APIs for native features on each platform. For native APIs that are not available in .NET MAUI's cross-platform APIs, you can invoke platform-specific code.

If you are planning to build a new app for Windows and want to target additional platforms, you should consider using .NET MAUI. If you are only targeting Windows with your app, there are some good reasons to continue using the Windows App SDK:

- **Familiarity**: .NET MAUI XAML and Windows App SDK XAML have some differences. If you are comfortable with XAML in UWP and Windows App SDK, you will have a bit of a learning curve with the .NET MAUI controls and XAML syntax.

- **Native Controls**: .NET MAUI does not currently support using Windows App SDK controls. If you have existing controls from other Windows App SDK projects you intend to re-use or rely on 3rd Party or open source controls, you will need to find alternatives for .NET MAUI projects.
- **Closer to Windows**: When writing .NET MAUI apps, it outputs a Windows App SDK app, but there is some translation to get from your code to the native Windows app. With Windows App SDK, you are eliminating that translation step and are less likely to encounter issues with styles, API compatibility, or layout.

> **Tip**
>
> Are you still deciding between .NET MAUI, WinUI, and other UI framework options? Check out the **Windows developer FAQ** to learn about the different options for building native Windows apps. Then go watch the **Windows Dev Chat** episode on **Choosing your ideal dev platform**. The segment about **.NET MAUI** hosted by Beth Massi walks through examples and reasons for choosing .NET MAUI for your next app.

.NET MAUI resources for Windows developers

.NET MAUI documentation

The .NET MAUI docs include resources for learning about .NET MAUI development for every platform, including Windows.

WinUI documentation

Explore the WinUI docs to learn about the features of WinUI.

.NET Conf: Focus on MAUI

This one-day live stream from August 2022 featured speakers from Microsoft and the .NET MAUI developer community. Learn how to build apps and hear from the team building .NET MAUI.

.NET MAUI blog posts

- 5 .NET MAUI Features for Building Great Desktop Apps
- Build Your Own Podcast App with .NET, Blazor, and .NET MAUI

GitHub

.NET MAUI is open source and hosted on GitHub. Use the .NET MAUI repository to file feature requests or bugs, interact with the development team, and explore the wiki.

Code samples

Explore the .NET MAUI code samples in the samples browser or on GitHub at dotnet/maui-samples.

Samples that demonstrate specific tasks

Expand table

Sample Name/Link	Task
Animations	**Use animations** to bring your app to life.
Behaviors	**Add functionality without subclassing** using behaviors, instead attaching behavior classes to your controls.
Control Templates	**Define the visual structure** of `ContentView` derived custom controls, and `ContentPage` derived pages.
Data Binding	**Bind objects** to the actual UI elements, responsively updating one another when one changes.
Shell App	**Use a .NET Shell app** to reduce complexity, reuse code, and integrate existing Windows functionality (such as URI navigation and integrated search bars) into your app.
Triggers	**Use triggers** to selectively update, hide, or display specific controls when an event or data changes.

Get started with .NET MAUI on Windows

To get started with .NET MAUI on Windows, install Visual Studio 2022 version 17.3 or later.

- Download 2022 Community
- Download 2022 Professional
- Download 2022 Enterprise

When installing or modifying Visual Studio, select the **.NET Multi-platform App UI development** workload with the default optional installation options selected. For more information about managing workloads in Visual Studio, see Modify Visual Studio workloads, components, and language packs. If you are using Visual Studio Code and the .NET CLI, you will use the dotnet workload command for workload management.

If you haven't enabled development mode on your PC, see Enable your device for development. If it isn't enabled, Visual Studio will prompt you to enable development mode when you try to run your first .NET MAUI project on Windows.

Next steps

Check out a walk-through of creating your first .NET MAUI app on Windows.

Related content

What is .NET MAUI

Windows developer FAQ

.NET MAUI Community Toolkit documentation

Build your first .NET MAUI app for Windows

Article • 01/23/2025

Get hands-on with .NET MAUI by building your first cross-platform app on Windows.

Introduction

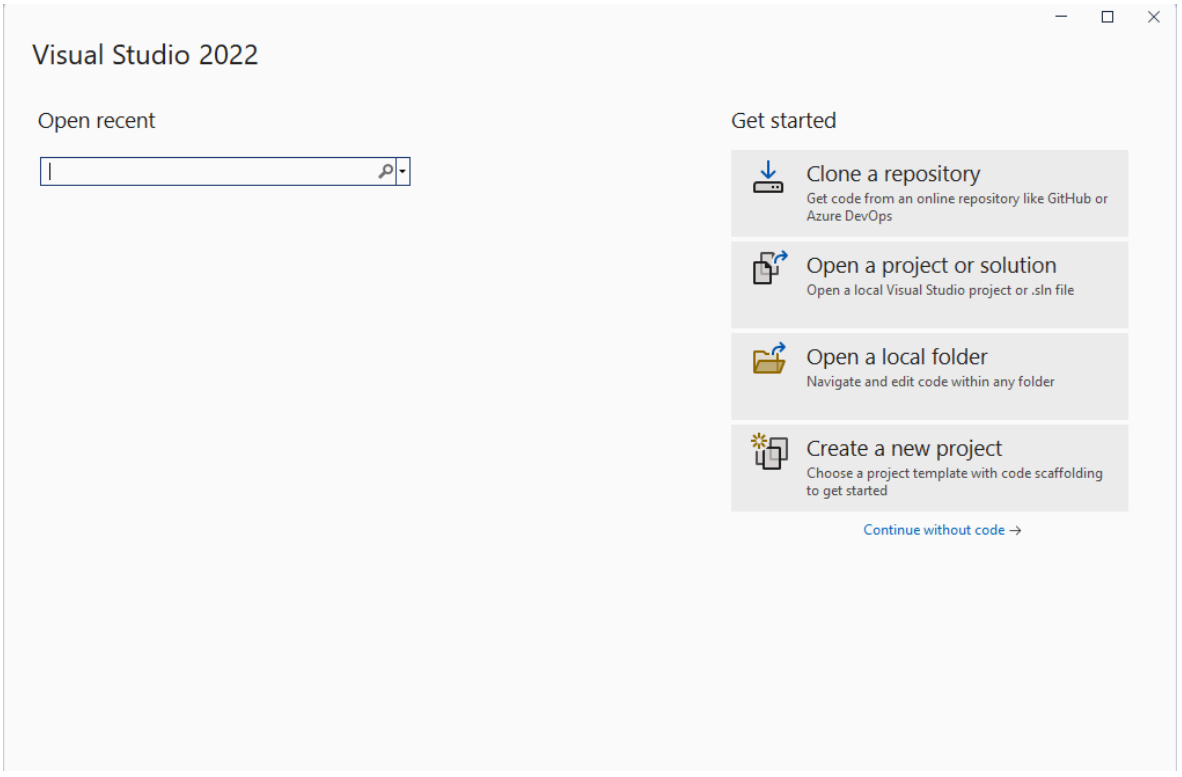
In this tutorial, you'll learn how to create and run your first .NET MAUI app for Windows in Visual Studio 2022 (17.3 or later). We will add some MVVM Toolkit features from the .NET Community Toolkit to improve the design the default project.

Setting up the environment

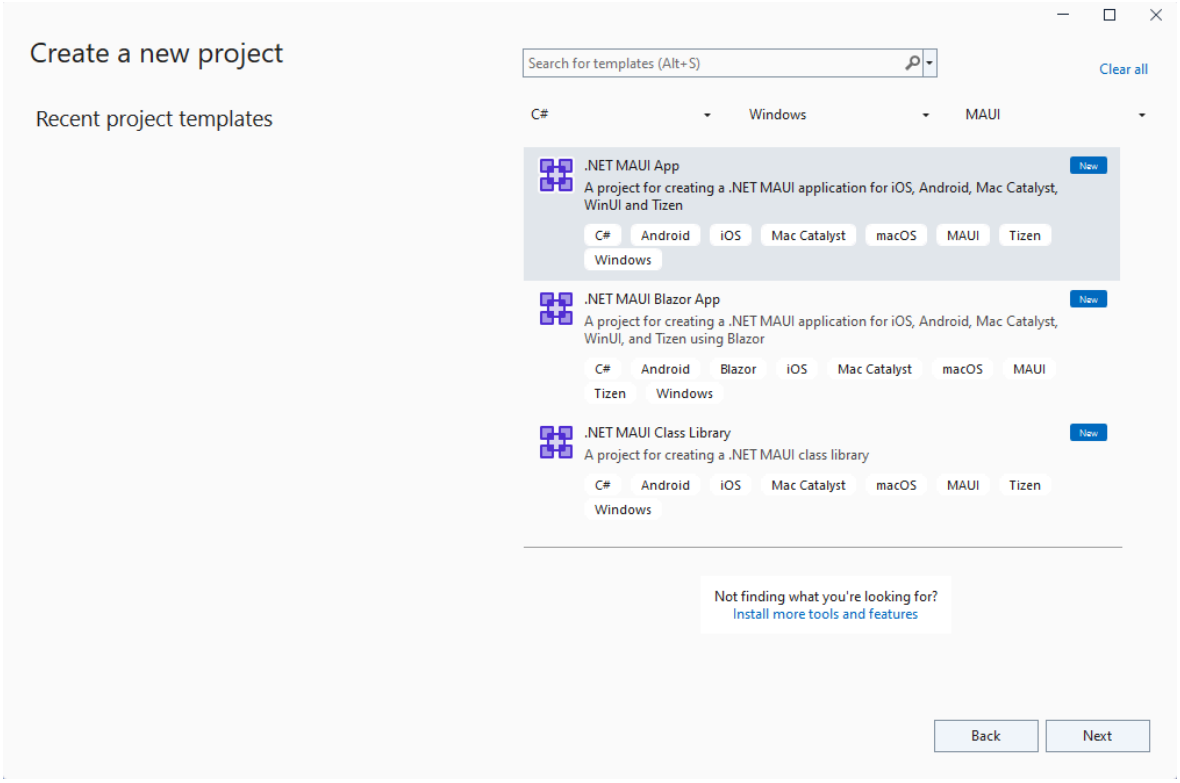
If you haven't already set up your environment for .NET MAUI development, please follow the steps to Get started with .NET MAUI on Windows.

Creating the .NET MAUI project

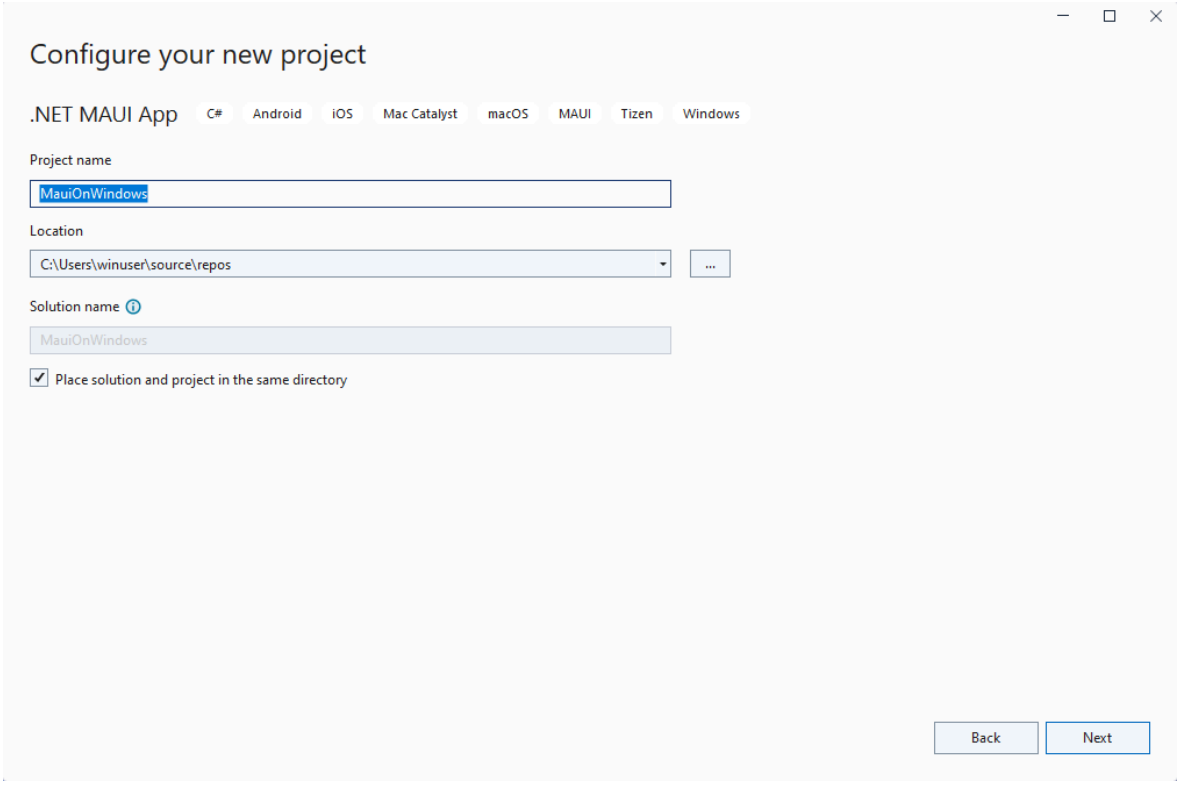
1. Launch Visual Studio, and in the start window click **Create a new project** to create a new project:



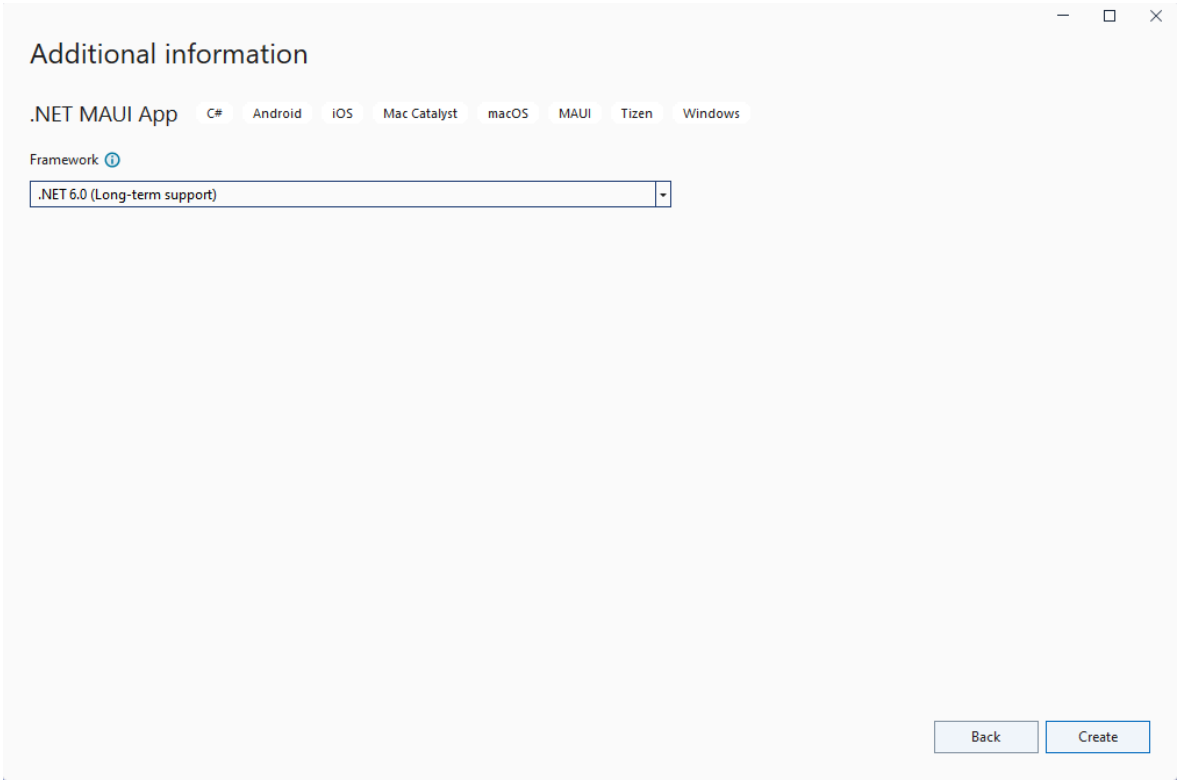
2. In the **Create a new project** window, select **MAUI** in the All project types drop-down, select the **.NET MAUI App** template, and click the **Next** button:



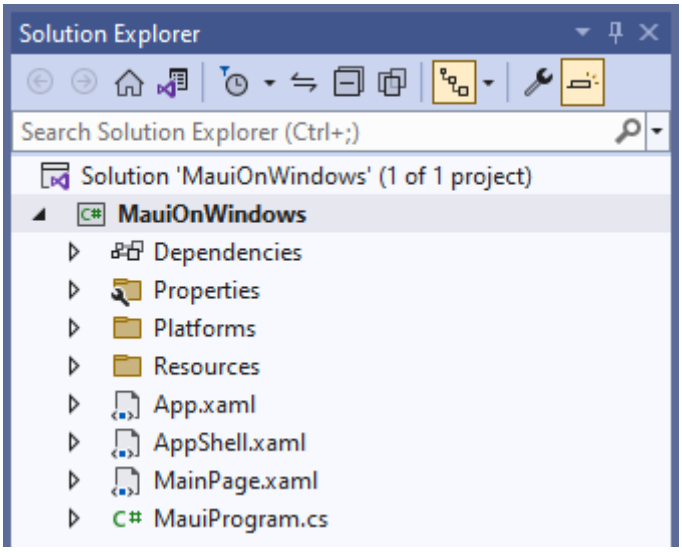
3. In the **Configure your new project** window, give your project a name, choose a location for it, and click the **Next** button:



4. In the **Additional information** window, click the **Create** button:

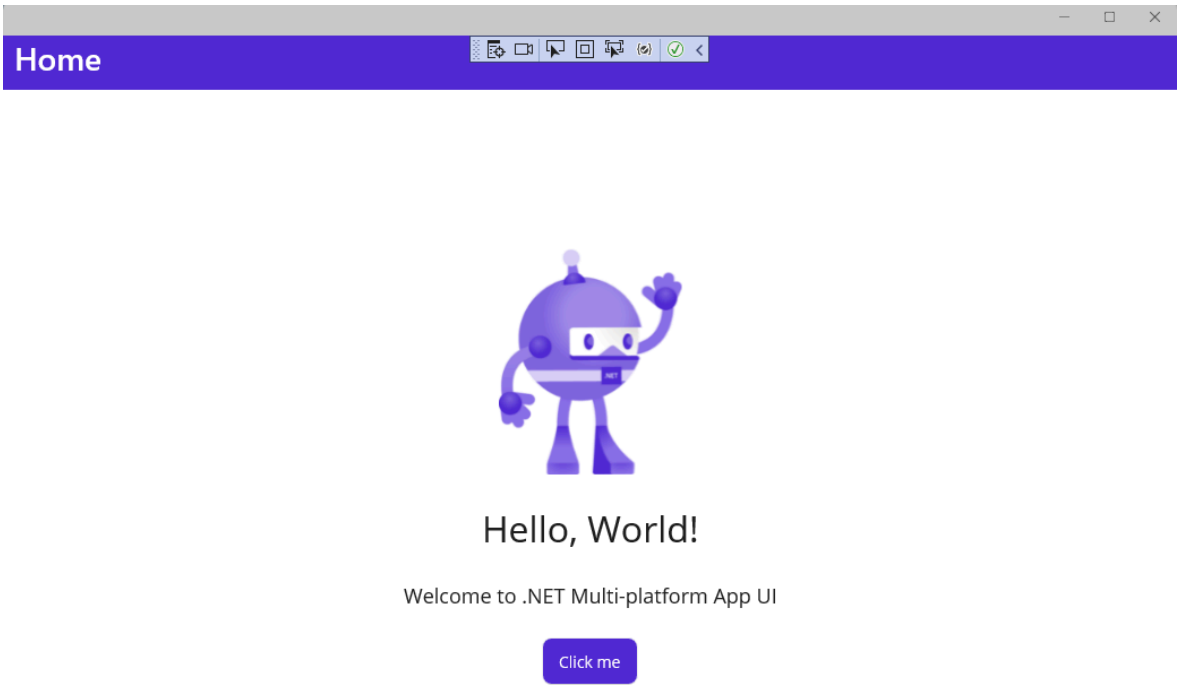


5. Wait for the project to be created, and for its dependencies to be restored:



6. In the Visual Studio toolbar, press the Windows Machine button to build and run the app.

7. In the running app, press the **Click me** button several times and observe that the count of the number of button clicks is incremented:



You just ran your first .NET MAUI app on Windows. In the next section, you'll learn how to add data binding and messaging features from the **MVVM Toolkit** to your app.

Troubleshooting

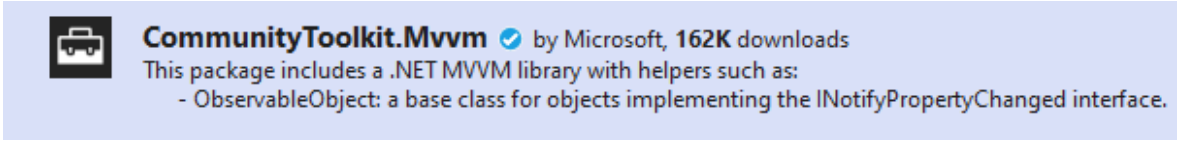
If your app fails to compile, review Troubleshooting known issues, which may have a solution to your problem.

Adding the MVVM Toolkit

Now that you have your first .NET MAUI app running on Windows, let's add some MVVM Toolkit features to the project to improve the app's design.

1. Right-click the project in **Solution Explorer** and select **Manage NuGet Packages...** from the context menu.

2. In the **NuGet Package Manager** window, select the **Browse** tab and search for **CommunityToolkit.MVVM**:

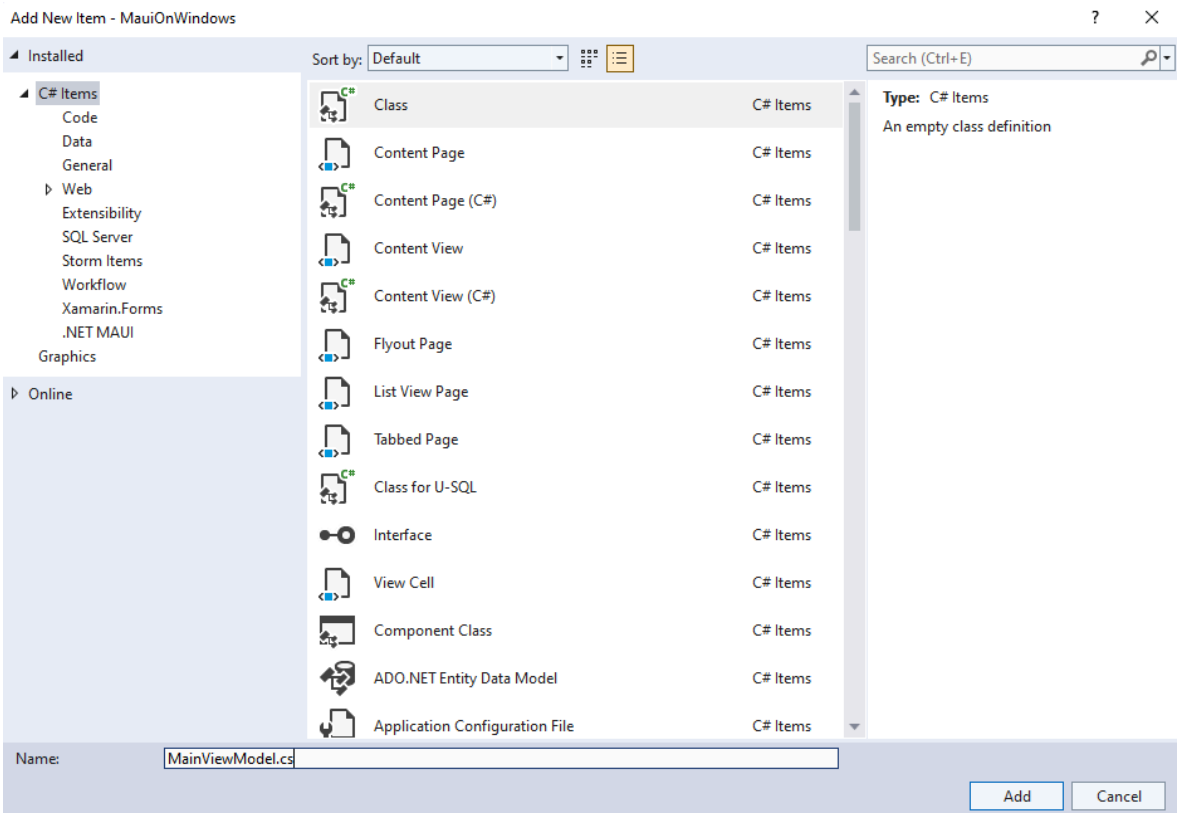


3. Add the latest stable version of the **CommunityToolkit.MVVM** package (version 8.0.0 or later) to the project by clicking **Install**.

4. Close the **NuGet Package Manager** window after the new package has finished installing.

5. Right-click the project again and select **Add | Class** from the context menu.

6. In the **Add New Item** window that appears, name the class `MainViewModel` and click **Add**:



7. The `MainViewModel` class will be the data binding target for the `MainPage`. Update it to inherit from `ObservableObject` in the `CommunityToolkit.Mvvm.ComponentModel` namespace This will also require updating the class to be `public` and `partial`.

8. The `MainViewModel` class will contain the following code. The `CountChangedMessage` record defines a message that is sent each time the Click me button is clicked, notifying the view of the change. The ObservableProperty and RelayCommand attributes added to the `message` and `IncrementCounter` members are source generators provided by the MVVM Toolkit to create the MVVM boilerplate code for `INotifyPropertyChanged` and `IRelayCommand` implementations. The `IncrementCounter` method's implementation contains the logic from `OnCounterClicked` in MainPage.xaml.cs, with a change to send a message with the new counter message. We will be removing that code-behind code later.

C#

```
using CommunityToolkit.Mvvm.ComponentModel;
using CommunityToolkit.Mvvm.Input;
using CommunityToolkit.Mvvm.Messaging;

namespace MauiOnWindows
{
    public sealed record CountChangedMessage(string Text);

    public partial class MainViewModel : ObservableObject
    {
        [ObservableProperty]
        private string message = "Click me";

        private int count;

        [RelayCommand]
        private void IncrementCounter()
        {
            count++;

            // Pluralize the message if the count is greater than 1
            message = count == 1 ? $"Clicked {count} time" : $"Clicked
{count} times";

            WeakReferenceMessenger.Default.Send(new
CountChangedMessage(message));
        }
    }
}
```

> **Note**
>
> You will need to update the namespace in the previous code to match the namespace in your project.

9. Open the **MainPage.xaml.cs** file for editing and remove the `OnCounterClicked` method and the `count` field.

10. Add the following code to the `MainPage` constructor after the call to `InitializeComponent()`. This code will receive the message sent by `IncrementCounter()` in the `MainViewModel` and will update the `CounterBtn.Text` property with the new message and announce the new text with the `SemanticScreenReader`:

C#

```
WeakReferenceMessenger.Default.Register<CountChangedMessage>(this, (r,
m) =>
{
    CounterBtn.Text = m.Text;
    SemanticScreenReader.Announce(m.Text);
});
```

11. You will also need to add a `using` statement to the class:

```
C#

using CommunityToolkit.Mvvm.Messaging;
```

12. In `MainPage.xaml`, add a namespace declaration to the `ContentPage` so the `MainViewModel` class can be found:

```
XAML

xmlns:local="clr-namespace:MauiOnWindows"
```

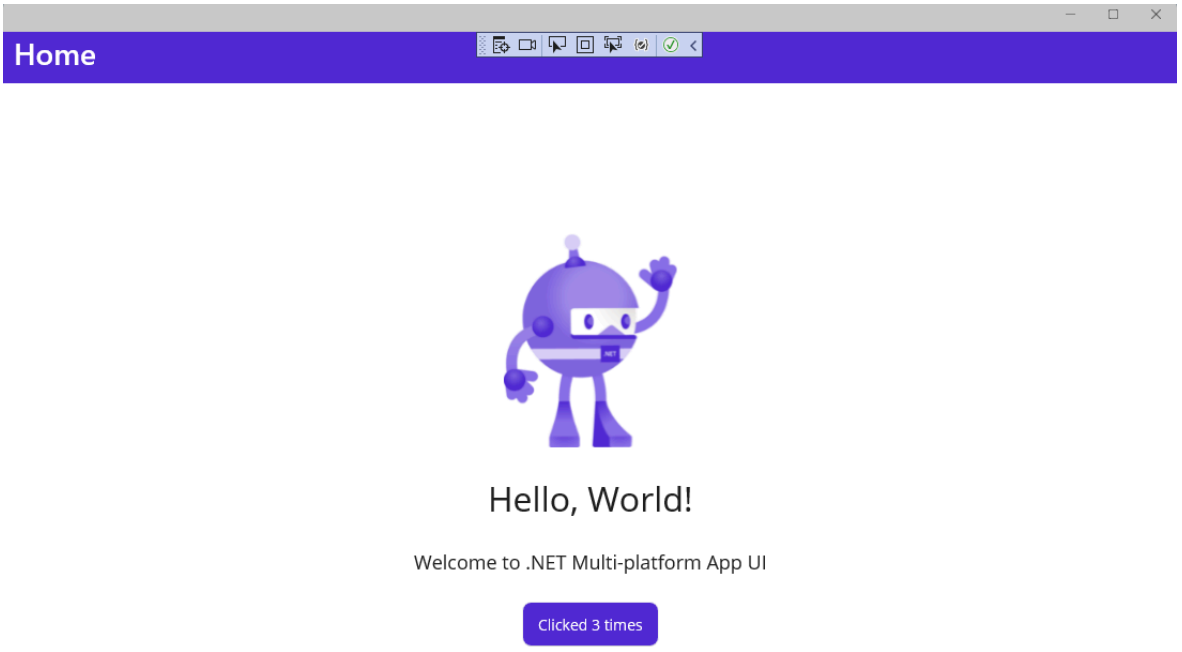
13. Add `MainViewModel` as the `BindingContext` for the `ContentPage`:

```
XAML

<ContentPage.BindingContext>
    <local:MainViewModel/>
</ContentPage.BindingContext>
```

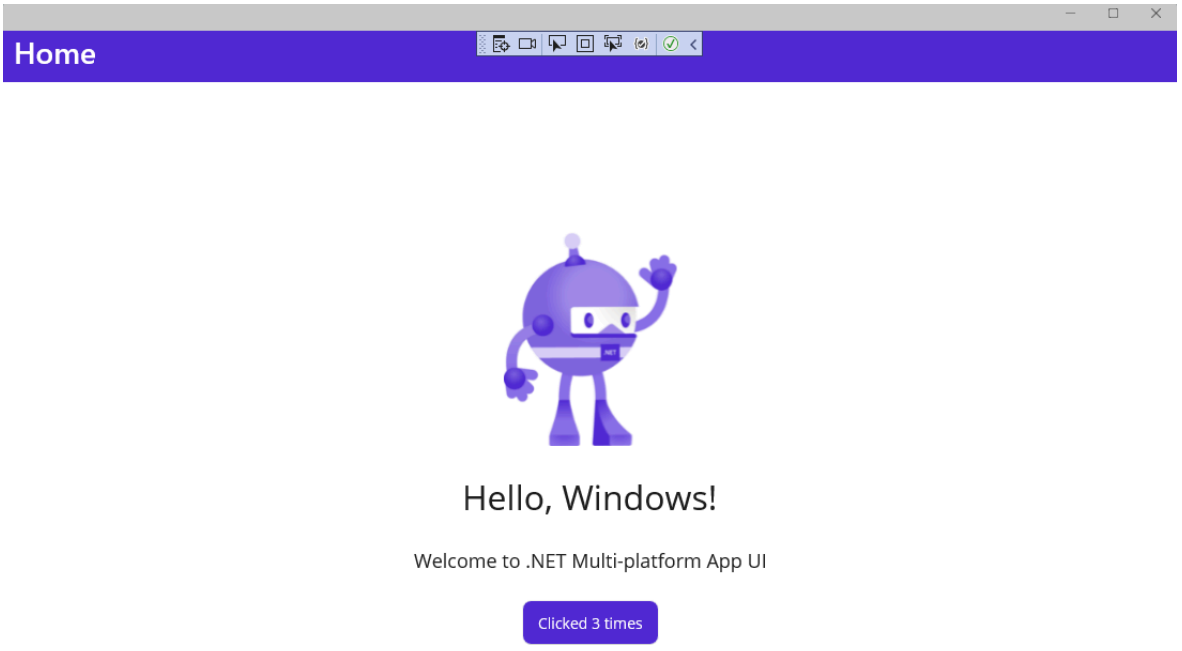
14. Update the `Button` on `MainPage` to use a `Command` instead of handling the `Clicked` event. The command will bind to the `IncrementCounterCommand` public property that is generated by the MVVM Toolkit's source generators:

```
XAML

<Button
    x:Name="CounterBtn"
    Text="Click me"
    SemanticProperties.Hint="Counts the number of times you click"
    Command="{Binding Path=IncrementCounterCommand}"
    HorizontalOptions="Center" />
```

15. Run the project again and observe that the counter is still incremented when you click the button:



16. While the project is running, try updating the "Hello, World!" message in the first Label to read "Hello, Windows!" in **MainPage.xaml**. Save the file and notice that XAML Hot Reload updates the UI while the app is still running:



Next steps

Learn to build an app that displays Microsoft Graph data for a user by leveraging the Graph SDK in a .NET MAUI for Windows tutorial.

Related content

Resources for learning .NET MAUI

Using the Microsoft Graph API with .NET MAUI on Windows

Tutorial: Create a .NET MAUI app using the Microsoft Graph SDK

Article • 07/13/2023

Get hands-on with .NET MAUI by building a cross-platform app on Windows that leverages the Microsoft Graph SDK to display user data.

In this tutorial, you learn how to:

- ✓ Set up your environment for .NET MAUI development and create a .NET MAUI project
- ✓ Register your client app in Azure
- ✓ Integrate with Azure Identity and the Microsoft Graph SDK
- ✓ Update the user interface to display user information from Microsoft Graph

Prerequisites

- An Azure account with an active subscription. Create an account for free.
- The .NET MAUI installation requirements.
- If you are new to .NET MAUI on Windows, you should start with the Build your first .NET MAUI app for Windows tutorial.

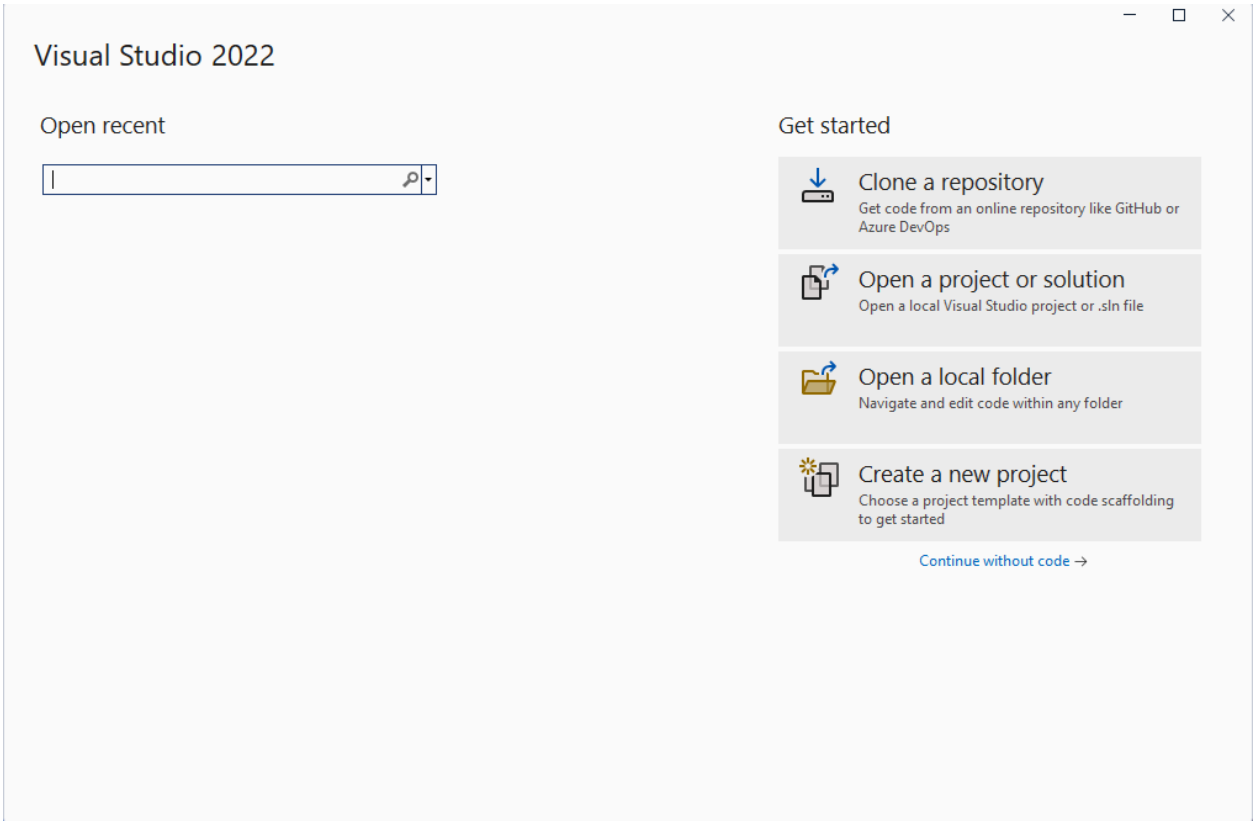
Setting up the environment

If you haven't already set up your environment for .NET MAUI development, please follow the steps to Get started with .NET MAUI on Windows.

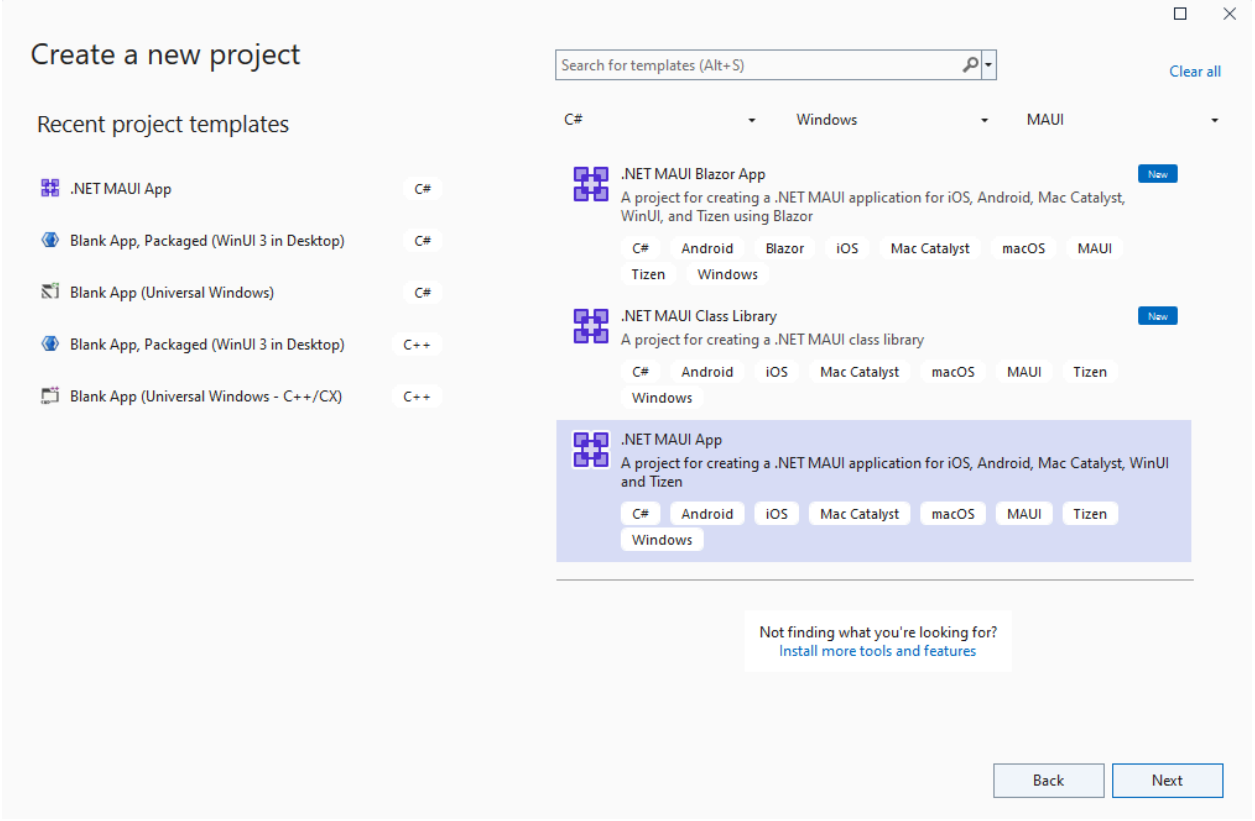
Creating the .NET MAUI project

> ⓘ **Note**
>
> If you are already familiar with setting up a .NET MAUI project, you can skip to the Register your client in Azure section.

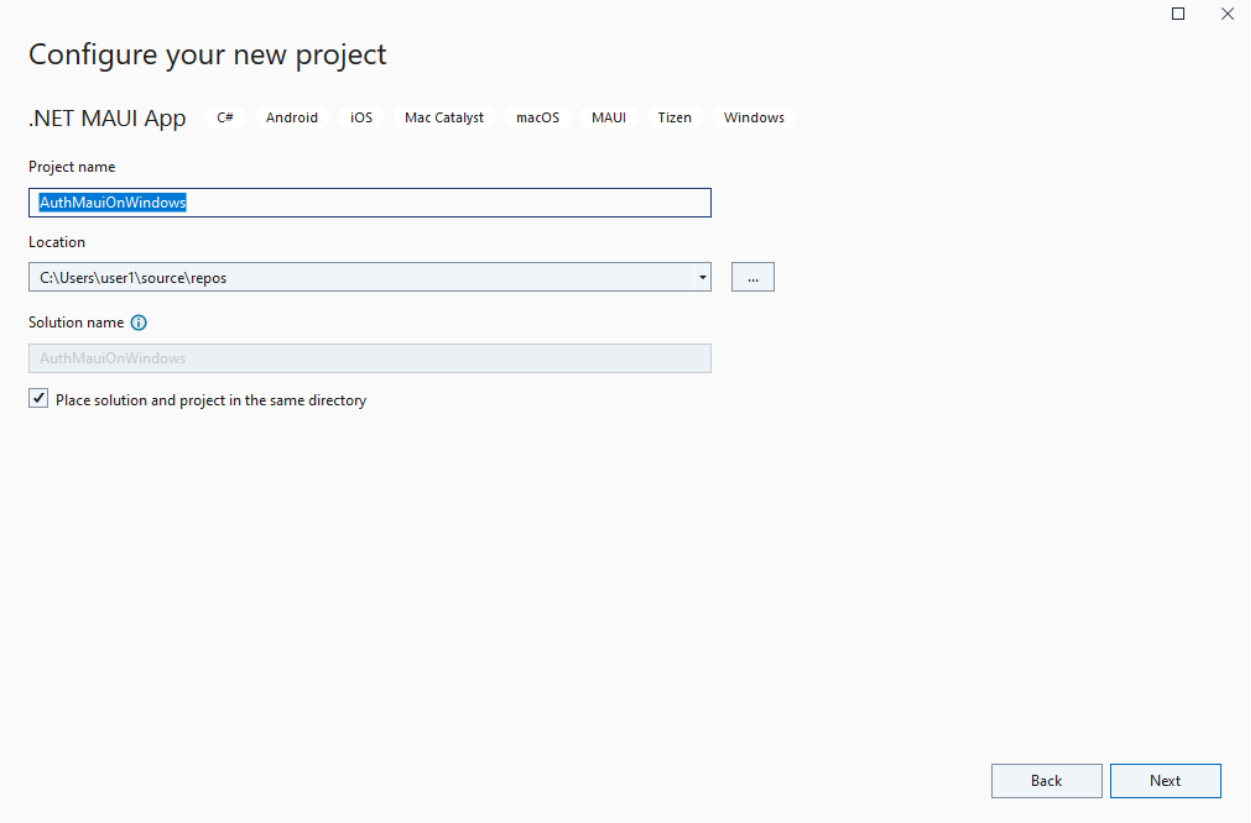
Launch Visual Studio, and in the start window click **Create a new project** to create a new project:



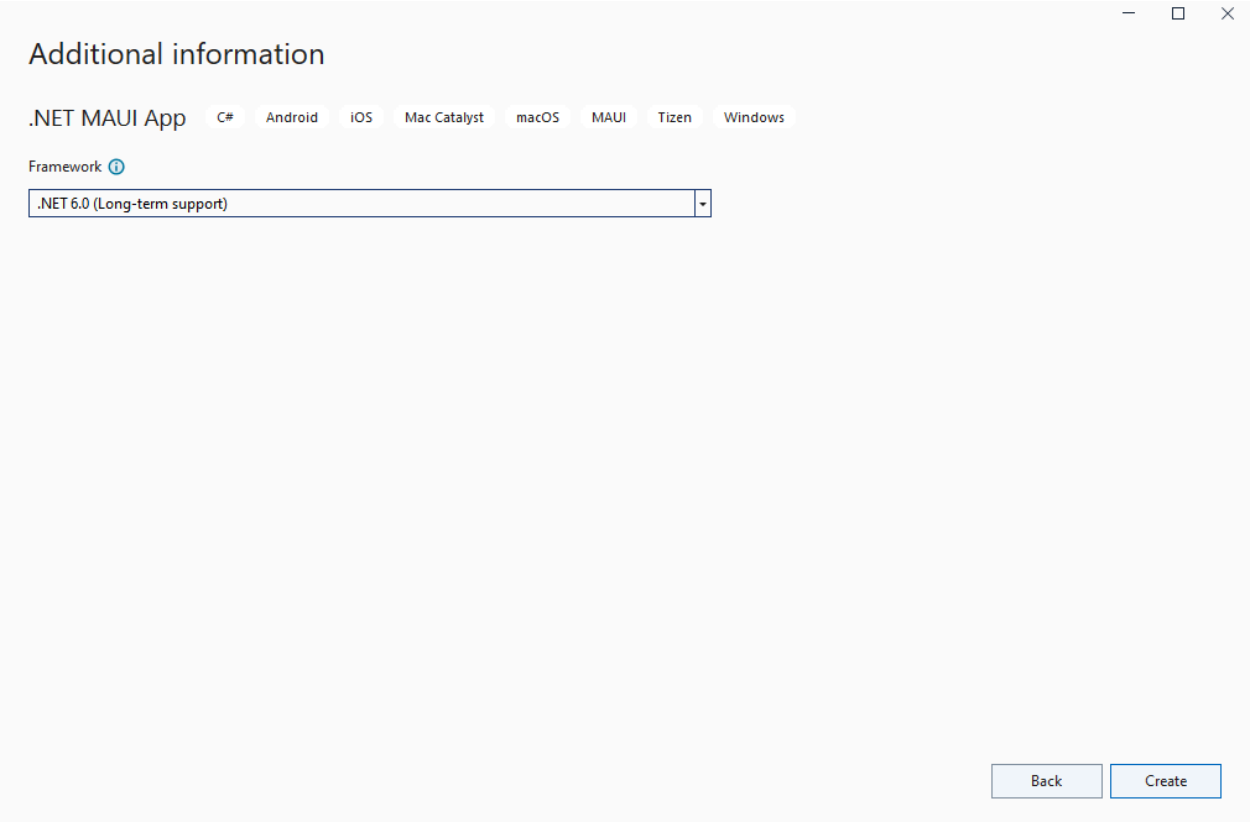
In the **Create a new project** window, select **MAUI** in the All project types drop-down, select the **.NET MAUI App** template, and click the **Next** button:



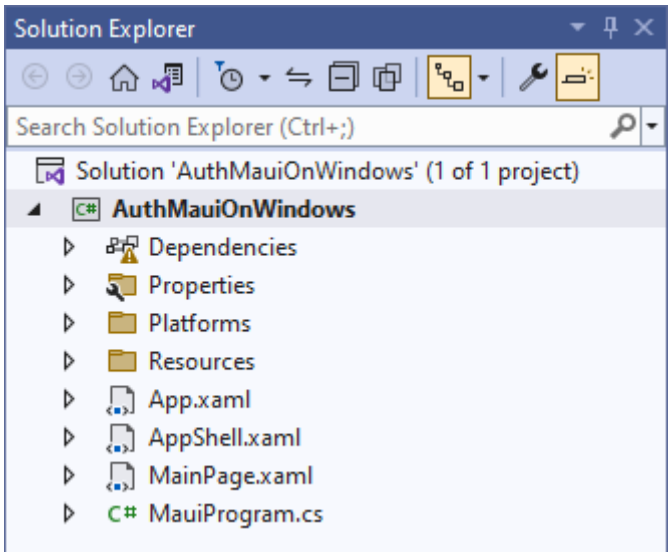
In the **Configure your new project** window, give your project a name, choose a location for it, and click the **Next** button:



In the **Additional information** window, click the **Create** button:



Wait for the project to be created, and for its dependencies to be restored:



In the Visual Studio toolbar, press the **Windows Machine** button to build and run the app. Click the **Click me** button and verify that the button content updates with the number of clicks.

Now that you have verified that the .NET MAUI app on Windows is working as expected, we can integrate the Graph SDK. In the next section, you'll add the packages necessary to authenticate and make calls to Microsoft Graph.

Register your client in Azure

An **app registration** in Azure with the **User.Read** scope granted to the app is required to read user data from Microsoft Graph. To register your application, follow these steps:

Sign in to the Azure portal.

If you have access to multiple tenants, use the **Directories + subscriptions** filter in the top menu to switch to the tenant in which you want to register the application.

Search for and select **Azure Active Directory**.

Under **Manage**, select **App registrations** > **New registration**.

Enter a **Name** for your application, for example `Win-App-calling-MsGraph`. Users of your app might see this name, and you can change it later.

In the Supported account types section, select Accounts in any organizational directory and personal Microsoft accounts (for example, Skype, Xbox, Outlook.com).

Select Register to create the application.

Copy and save the of the **Application (client) ID** and the **Directory (tenant) ID** values for later use. We will store those values in the GraphService class in the next section.

Under **Manage**, select **Authentication**.

Select **Add a platform** > **Mobile and desktop applications**.

In the **Redirect URIs** section, select `https://login.microsoftonline.com/common/oauth2/nativeclient` and in **Custom redirect URIs** add `http://localhost`.

Select **Configure**.

Under **Manage**, select **API permissions**.

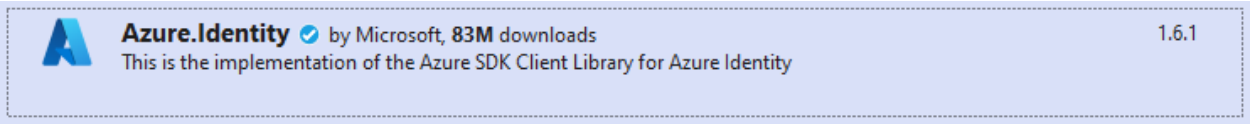
If the Microsoft Graph **User.Read** permission is not present under **Configured permissions**, select **Add permission**. In the **Request API permissions** screen, select **Microsoft Graph** > **Application permissions** and search for **User.Read**. Expand **User**, select **User.Read**, and click **Add permissions**.

Integrate the Graph SDK and Azure Identity

Now that you have your .NET MAUI app running on Windows and have configured the app registration in Azure, let's add a couple of NuGet packages to the project to integrate with Azure Identity and Microsoft Graph.

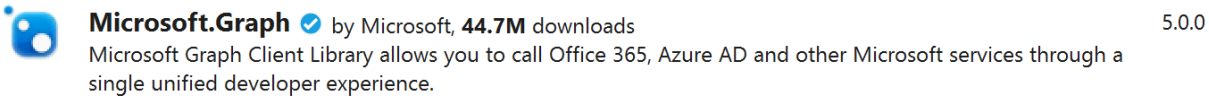
Right-click the project in **Solution Explorer** and select **Manage NuGet Packages...** from the context menu.

In the **NuGet Package Manager** window, select the **Browse** tab and search for **Azure.Identity**:



Add the latest stable version of the **Azure.Identity** package to the project by clicking **Install**.

Next, search for **Microsoft.Graph**:

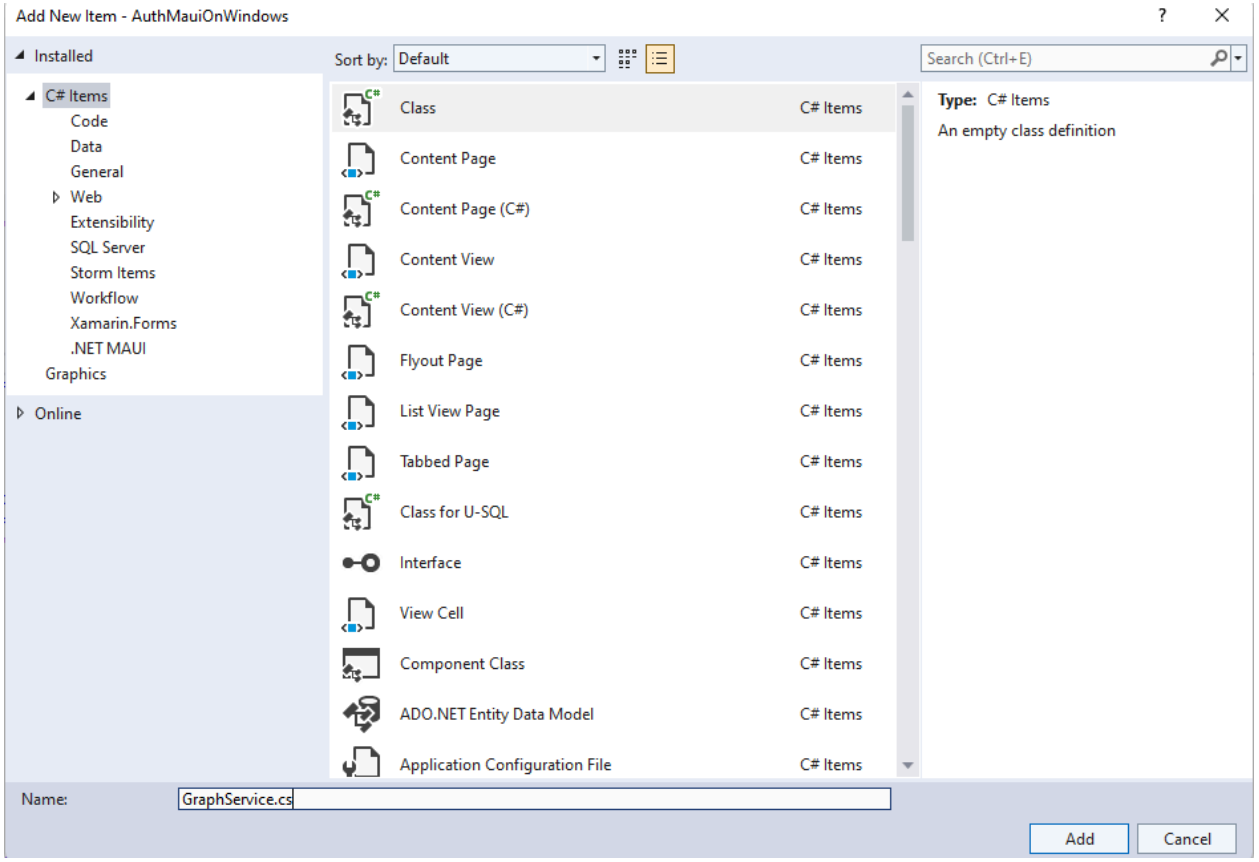


Add the latest stable version of the **Microsoft.Graph** package to the project by clicking **Install**.

Close the **NuGet Package Manager** window after the new package has finished installing.

Right-click the project again and select **Add | Class** from the context menu.

In the **Add New Item** window that appears, name the class `GraphService` and click **Add**:



Add four private members to the `GraphService` class, substituting your own *client ID* and *tenant ID* values for the placeholder text:

C#

```
private readonly string[] _scopes = new[] { "User.Read" };
private const string TenantId = "<add your tenant id here>";
private const string ClientId = "<add your client id here>";
private GraphServiceClient _client;
```

Add an `Initialize()` method to `GraphService`, which will be called from the constructor. The initialization code will authenticate using the `InteractiveBrowserCredential` class. Upon successful authentication, the auth token will be provided to the `GraphServiceClient` by the credential class along with the requested scopes (`User.Read`).

C#

```
public GraphService()
{
    Initialize();
}

private void Initialize()
{
    // assume Windows for this sample
    if (OperatingSystem.IsWindows())
    {
        var options = new InteractiveBrowserCredentialOptions
        {
            TenantId = TenantId,
            ClientId = ClientId,
            AuthorityHost = AzureAuthorityHosts.AzurePublicCloud,
            RedirectUri = new Uri("http://localhost"),
        };

        InteractiveBrowserCredential interactiveCredential = new(options);
        _client = new GraphServiceClient(interactiveCredential, _scopes);
    }
    else
    {
        // TODO: Add iOS/Android support
    }
}
```

> **Note**
>
> The 'Initialize()' method currently only supports Windows. Authenticating on iOS and Android requires a different NuGet package (Microsoft.Identity.Client) and some additional steps. To read more about mobile authentication, see Configure a native client application.

Add a public async method named `GetMyDetailsAsync()` to return the `User` object for the authenticated user:

C#

```
public async Task<User> GetMyDetailsAsync()
{
    try
    {
        return await _client.Me.GetAsync();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error loading user details: {ex}");
        return null;
```

```
    }
}
```

Two `using` statements will be needed to compile the new code added to `GraphService`:

C#

```
using Azure.Identity;
using Microsoft.Graph;
using Microsoft.Graph.Models;
```

In **MainPage.xaml**, add an `x:Name` to the **Hello, World!** label:

XAML

```
<Label
    x:Name="HelloLabel"
    Text="Hello, World!"
    SemanticProperties.HeadingLevel="Level1"
    FontSize="32"
    HorizontalOptions="Center" />
```

Add a new button to the page after the existing `CounterBtn` to get the user information from Graph:

XAML

```
<Button
    x:Name="GetUserInfoBtn"
    Text="Load User Info"
    SemanticProperties.Hint="Loads user information from Microsoft Graph"
    Clicked="GetUserInfoBtn_Clicked"
    HorizontalOptions="Center" />
```

In **MainPage.xaml.cs**, add private variables to for `GraphService` and `User`:

C#

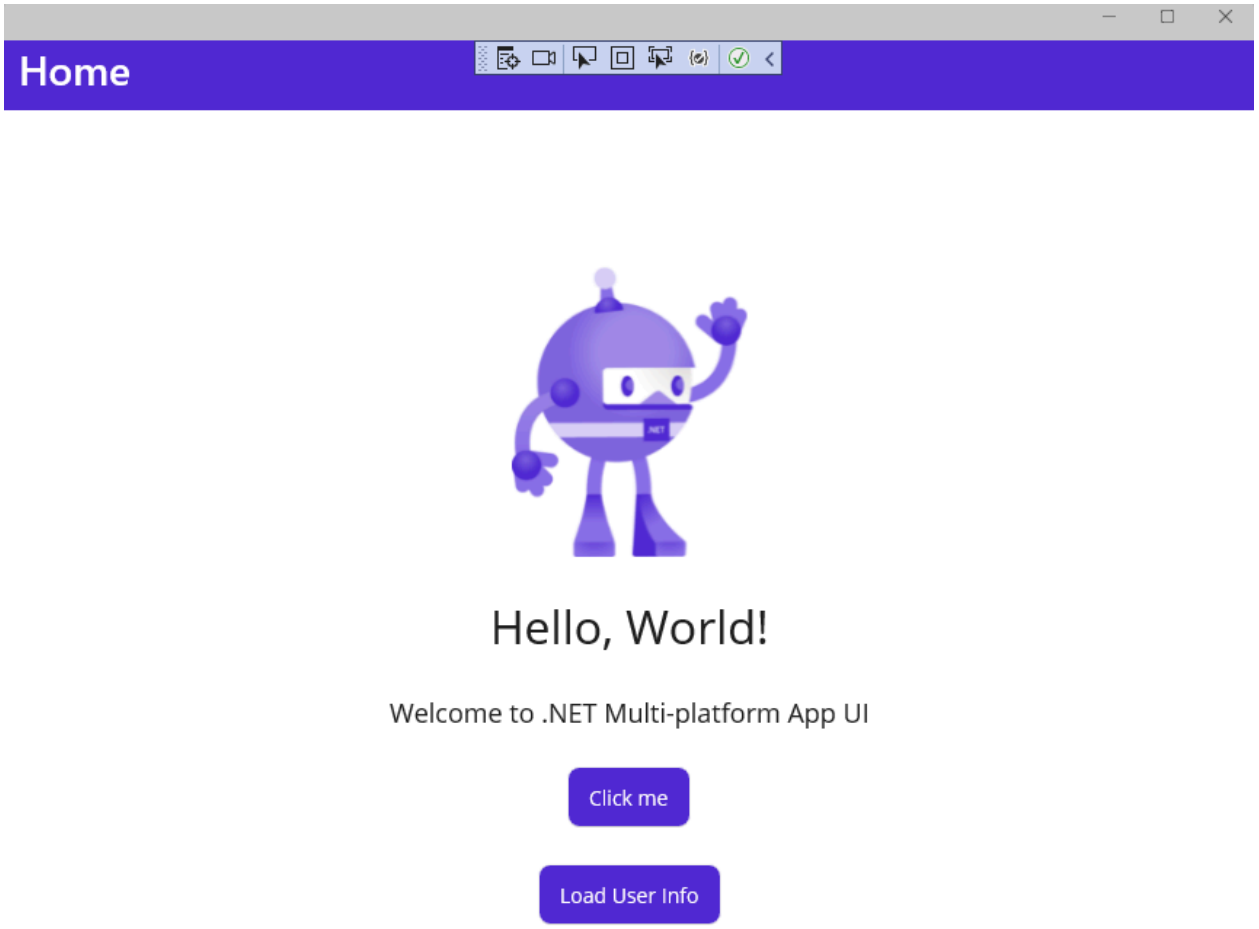
```
private GraphService graphService;
private User user;
```

Add an event handler for the `Clicked` event that you added to the `GetUserInfoButton` on **MainPage.xaml**. The event handler will create an instance of `GraphService` if it is null and make the call to fetch the user data. The `HelloLabel`'s text will be updated to say hello to the user, displaying the `DisplayName` property from Microsoft Graph:

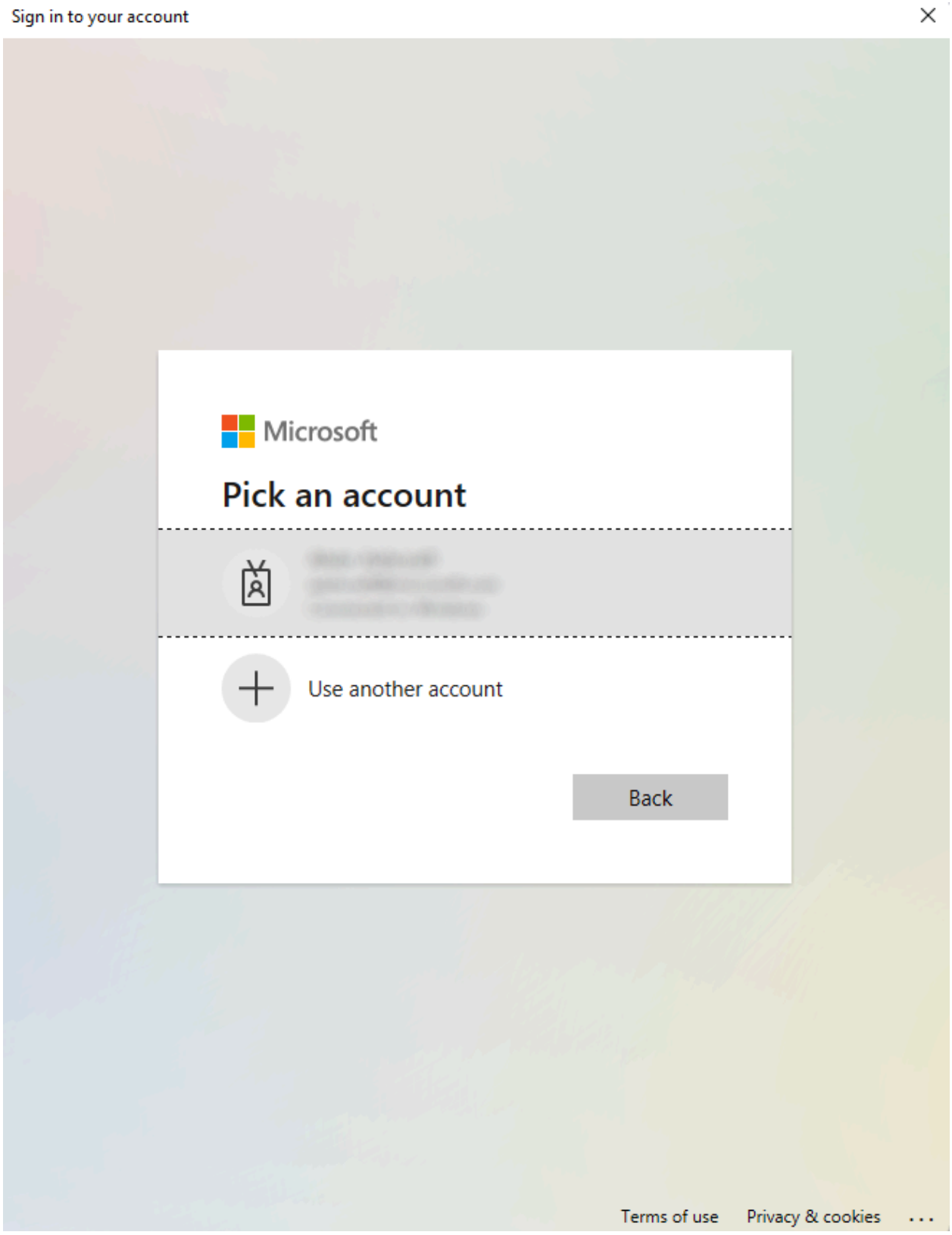
C#

```
private async void GetUserInfoBtn_Clicked(object sender, EventArgs e)
{
    if (graphService == null)
    {
        graphService = new GraphService();
    }
    user = await graphService.GetMyDetailsAsync();
    HelloLabel.Text = $"Hello, {user.DisplayName}!";
}
```

Run the app and click the **Load User Info** button:

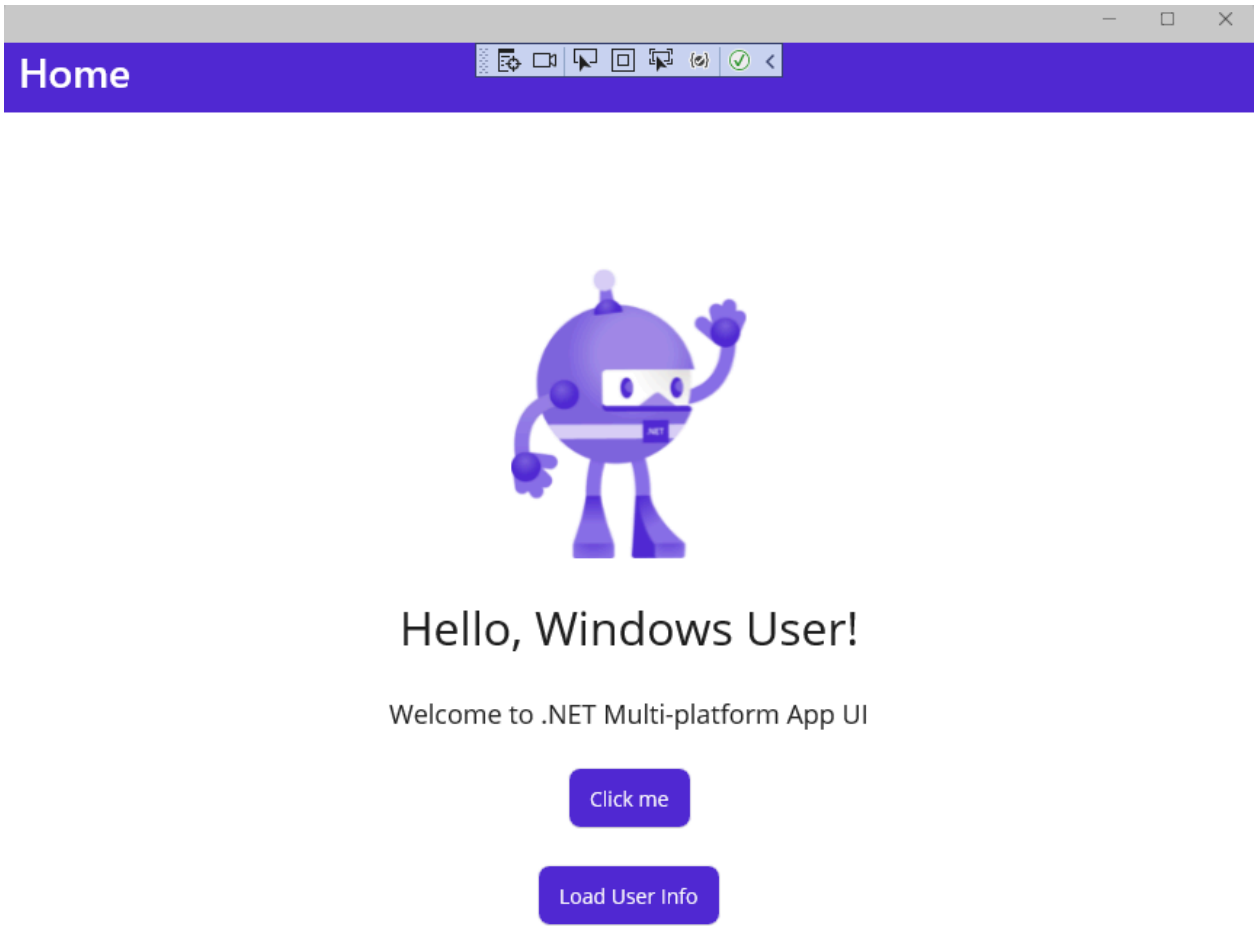


When the **Sign in to your account** window appears, select an existing account or click **Use another account**:



If you selected a different account, enter the email address and password for the account and sign in.

After the authentication completes, you will see your user's **DisplayName** in on the app:



Update the user interface to display user information

Now that the `GraphService` is returning user information, let's update the user interface to display some additional user profile information.

In MainPage.xaml, start by updating the contents of the `VerticalStackLayout`, removing the existing welcome label and `Image` control and adding four new labels to display the user information. Each label that will be updated is named, and we've provided some placeholder text until the data is loaded from the Graph query. The contents of the `VerticalStackLayout` should now look like this:

XAML

```
<VerticalStackLayout
    Spacing="25"
    Padding="30,0"
    VerticalOptions="Center">

    <Label
```

```
        x:Name="HelloLabel"
        Text="Hello, World!"
        SemanticProperties.Description="Displays a welcome message for the
user"
        SemanticProperties.HeadingLevel="Level1"
        FontSize="32"
        HorizontalOptions="Center" />

    <Button
        x:Name="CounterBtn"
        Text="Click me"
        SemanticProperties.Hint="Counts the number of times you click"
        Clicked="CounterBtn_Clicked"
        HorizontalOptions="Center" />

    <Button
        Text="Load User Info"
        SemanticProperties.Hint="Loads user information from Microsoft
Graph"
        Clicked="GetUserInfoBtn_Clicked"
        HorizontalOptions="Center" />

    <Label
        x:Name="DisplayNameLabel"
        Text="Display name"
        SemanticProperties.Description="Displays the user's display name
from Microsoft Graph."
        SemanticProperties.HeadingLevel="Level2"
        FontSize="18"
        HorizontalOptions="Center" />

    <Label
        x:Name="UserFirstNameLabel"
        Text="First name"
        SemanticProperties.Description="Displays the user's first name info
from Microsoft Graph"
        SemanticProperties.HeadingLevel="Level2"
        FontSize="18"
        HorizontalOptions="Center" />

    <Label
        x:Name="UserLastNameLabel"
        Text="Last name"
        SemanticProperties.Description="Displays the user's last name from
Microsoft Graph"
        SemanticProperties.HeadingLevel="Level2"
        FontSize="18"
        HorizontalOptions="Center" />

    <Label
        x:Name="UserPrincipalNameLabel"
        Text="User Principal Name"
        SemanticProperties.Description="Displays the user principal name
from Microsoft Graph"
        SemanticProperties.HeadingLevel="Level2"
```

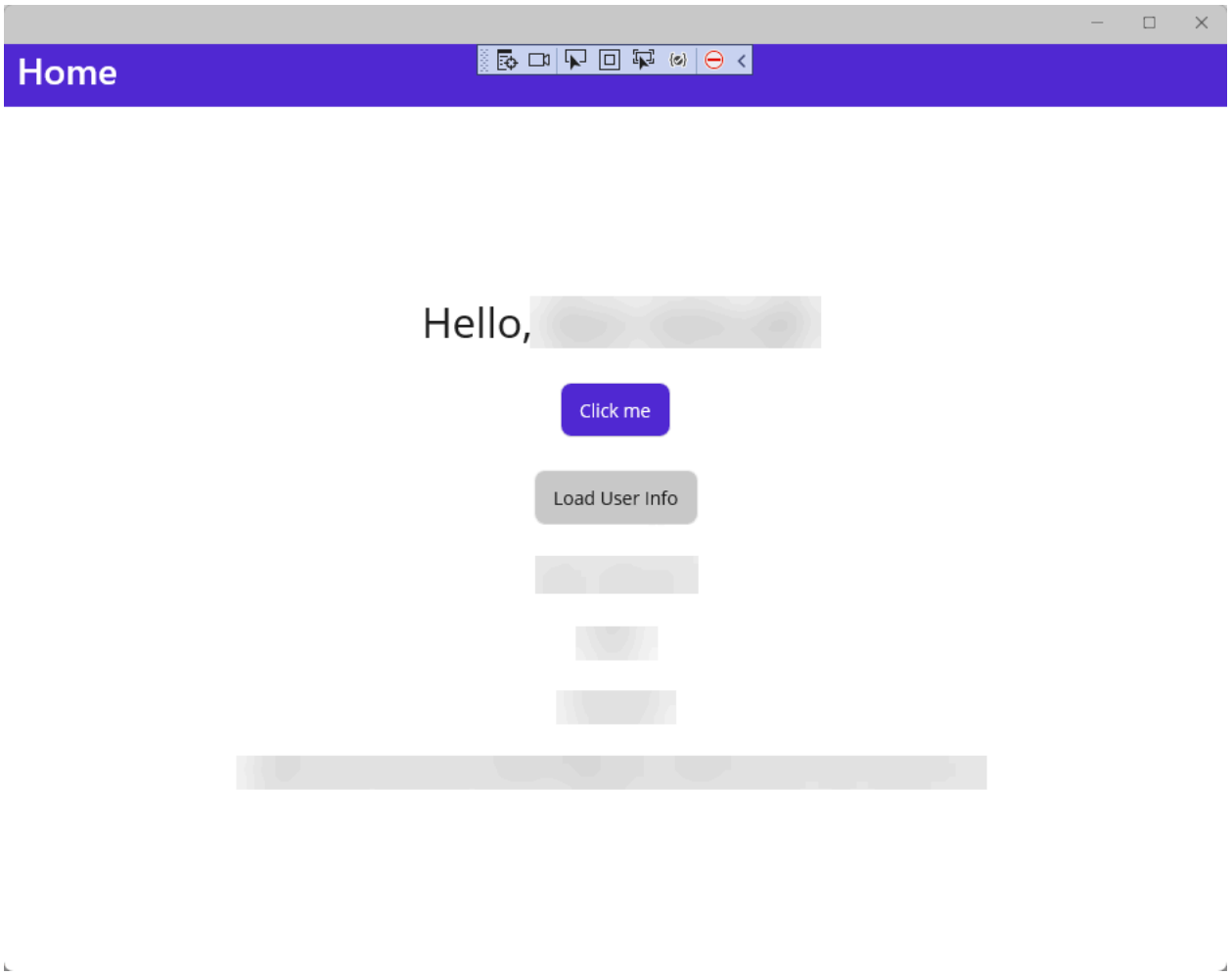
```
        FontSize="18"
        HorizontalOptions="Center" />

</VerticalStackLayout>
```

Finally, in MainPage.xaml.cs, update the UI elements with the values of the new properties in the `GetUserInfoBtn_Clicked` event handler using the properties of the Graph User object:

C#

```
private async void GetUserInfoBtn_Clicked(object sender, EventArgs e)
{
    if (graphService == null)
    {
        graphService = new GraphService();
    }
    user = await graphService.GetMyDetailsAsync();
    HelloLabel.Text = $"Hello, {user.DisplayName}!";

    DisplayNameLabel.Text = user.DisplayName;
    UserFirstNameLabel.Text = user.GivenName;
    UserLastNameLabel.Text = user.Surname;
    UserPrincipalNameLabel.Text = user.UserPrincipalName;
}
```

Run the app again and click the **Load User Info** button. You should see your user information displayed in the app after authenticating:



To learn how to install and get started with the Microsoft Graph SDK in your .NET MAUI apps, see Install the Microsoft Graph .NET SDK.

Next steps

Advance to the next article to learn how to...

Create a recommendation app with .NET MAUI and ChatGPT

See also

Resources for learning .NET MAUI

Microsoft Graph SDK overview

Configure a native client application

Azure AD identity and access management API overview

Tutorial: Create a recommendation app with .NET MAUI and ChatGPT

Article • 06/17/2024

In this tutorial, you'll learn how to create a .NET MAUI app for Windows in Visual Studio that calls OpenAI's ChatGPT APIs to provide recommendations based on a location entered by the user. The app will have a simple UI that allows the user to enter a location and get recommendations for restaurants, hotels, and attractions.

In this tutorial, you learn how to:

- ✓ Create a simple user interface for your .NET MAUI app
- ✓ Reference and bootstrap the **OpenAI .NET API** library
- ✓ Use an API key to link your app to an OpenAI API account
- ✓ Make calls to the OpenAI chat APIs to get recommendations

Prerequisites

- An OpenAI account
- An OpenAI API key
- The .NET MAUI installation requirements
- If you are new to .NET MAUI on Windows, you should start with the Build your first .NET MAUI app for Windows tutorial.

Set your environment variable

In order to use the OpenAI SDK, you'll need to set an environment variable with your API key. In this example, we'll use the `OPENAI_API_KEY` environment variable. Once you have your API key from the OpenAI developer dashboard, you can set the environment variable from the command line as follows:

PowerShell

```
setx OPENAI_API_KEY <your-api-key>
```

Note that this method works for development on Windows, but you'll want to use a more secure method for production apps and for mobile support. For example, you can store your API key in a secure key vault that a remote service can access on behalf of your app. See Best practices for OpenAI key safety for more information.

Create a new .NET MAUI project with the required UI elements

We're going to start by creating a new .NET MAUI project in Visual Studio. We'll use the **.NET MAUI App** template and add some UI elements to the **MainPage** to provide users with some recommendations based on a provided location. The UI will have buttons to get recommendations for restaurants, hotels, and attractions.

1. In Visual Studio, create a new **.NET MAUI App** project named **ChatGptRecommendationApp**.

2. Run the new project to make sure the app builds and runs successfully.

3. Open **MainPage.xaml** from the Solution Explorer.

4. Replace the contents of the `VerticalStackLayout` with the following XAML markup:

XAML

```
<Label
    Text="Local AI recommendations"
    SemanticProperties.HeadingLevel="Level1"
    FontSize="32"
    HorizontalOptions="Center" />

<Entry
    x:Name="LocationEntry"
    Placeholder="Enter your location"
    SemanticProperties.Hint="Enter the location for recommendations"
    HorizontalOptions="Center"/>

<Button
    x:Name="RestaurantBtn"
    Text="Get restaurant recommendations"
    SemanticProperties.Hint="Gets restaurant recommendations when you
click"
    Clicked="OnRestaurantClicked"
    HorizontalOptions="Center" />

<Button
    x:Name="HotelBtn"
    Text="Get hotel recommendations"
    SemanticProperties.Hint="Gets hotel recommendations when you click"
    Clicked="OnHotelClicked"
    HorizontalOptions="Center" />

<Button
    x:Name="AttractionBtn"
    Text="Get attraction recommendations"
    SemanticProperties.Hint="Gets attraction recommendations when you
```

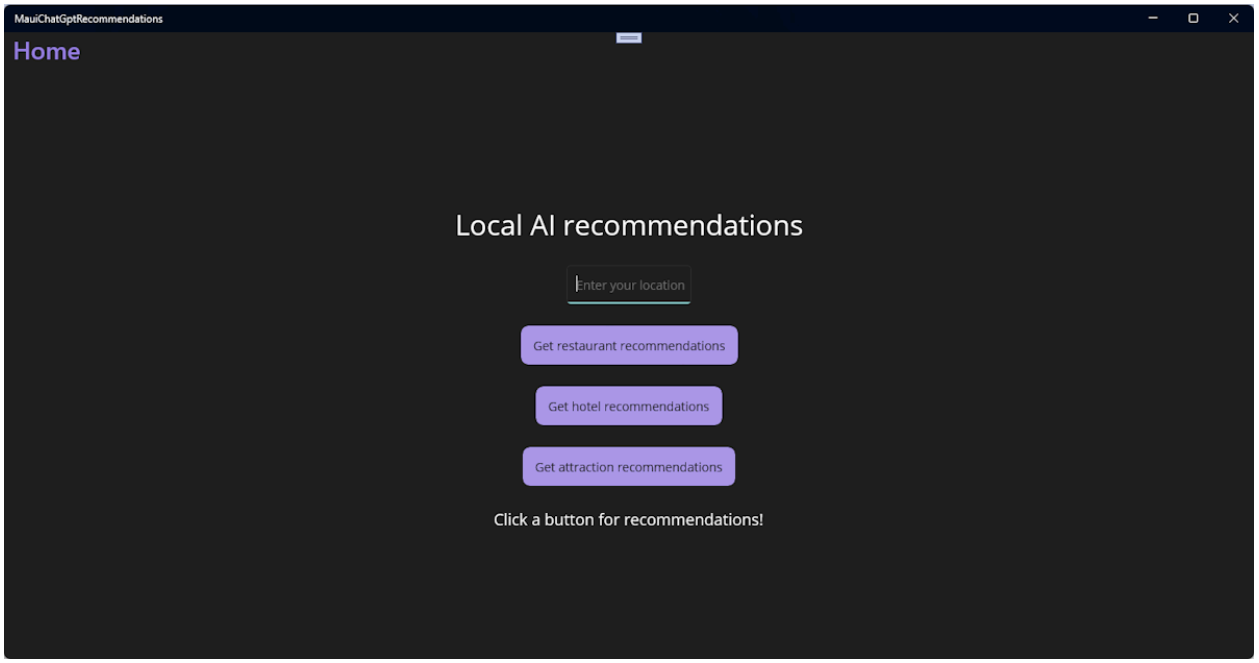
```
click"
    Clicked="OnAttractionClicked"
    HorizontalOptions="Center" />

<Label x:Name="SmallLabel"
    Text="Click a button for recommendations!"
    SemanticProperties.HeadingLevel="Level2"
    FontSize="18"
    HorizontalOptions="Center" />
```

5. In order to build the project, you will need to add `Clicked` event handlers for each of the buttons. Add the following code to the **MainPage.xaml.cs** file and remove the existing event handler:

C#

```
private async void OnRestaurantClicked(object sender, EventArgs e)
{
}

private async void OnHotelClicked(object sender, EventArgs e)
{
}

private async void OnAttractionClicked(object sender, EventArgs e)
{
}
```

The event handlers are all marked as `async` because we will be making asynchronous calls to the OpenAI .NET API library. Now when you run the app, you should see the following UI:



Users can enter their location in the `Entry` control and click one of the buttons to get recommendations for restaurants, hotels, or attractions. The `Label` control at the bottom of the UI will display the results.

Next, let's add the **OpenAI** library to the project and get it ready to make some API calls.

Reference and initialize the OpenAI library for .NET

To call OpenAI's ChatGPT APIs, we're going to use a pre-release version of the OpenAI library for .NET NuGet package. This library provides .NET applications with access to the OpenAI REST API. We'll add the library to our project and bootstrap it with our API key.

1. Open the **Package Manager Console** from the **Tools** menu in Visual Studio.
2. Install the OpenAI library by running the following command. The `IncludePrerelease` flag is required because the library is still in preview:

   PowerShell

   ```
   Install-Package OpenAI -IncludePrerelease
   ```

   This should install a pre-release of version 2.0.0 or later to your project. You can also install a specific version of the library by specifying the version number in the command. For example, to install version 2.0.0-beta.4, you would run the following command:

   PowerShell

   ```
   Install-Package OpenAI -Version 2.0.0-beta.4
   ```

3. Open **MainPage.xaml.cs**, the code-behind file for the **MainPage**. Add the following code to the top of the file to reference the OpenAI library and create a variable to hold your OpenAI client. Update the constructor to call the `MainPage_Loaded` method when the page is loaded, and add the `MainPage_Loaded` method to get your OpenAI API key from the Windows registry and initialize the OpenAI client:

   C#

   ```
   private OpenAIClient _chatGptClient;
   ```

```
public MainPage()
{
    InitializeComponent();
    this.Loaded += MainPage_Loaded;
}

private void MainPage_Loaded(object sender, EventArgs e)
{
    var openAiKey =
Environment.GetEnvironmentVariable("OPENAI_API_KEY");

    _chatGptClient = new(openAiKey);
}
```

This will bootstrap the OpenAI library with your API key. You can create your API key on the OpenAI API settings page.

4. In order to compile the project, you will need to add the following `using` statements to the top of the **MainPage.xaml.cs** file:

```
C#

using OpenAI;
using OpenAI.Chat;
using System.ClientModel;
```

Now we're ready to put it all together. In the next section, we'll add some code to the three event handlers to make calls to the OpenAI library and display the recommendation results.

Add ChatGPT API calls and test the app

It's time to add the code to our code-behind file that will use the OpenAI library for .NET to make calls to the OpenAI ChatGPT API. We'll add the code to the three event handlers we created earlier. The code will get the user's location from the `Entry` control and pass it to the API to get recommendations. Then we'll display the results in the `Label` control at the bottom of the UI.

1. Create an `async` method named `GetRecommendationAsync` and call it from each of the event handlers:

```
C#

private async void OnRestaurantClicked(object sender, EventArgs e)
{
    await GetRecommendationAsync("restaurant");
```

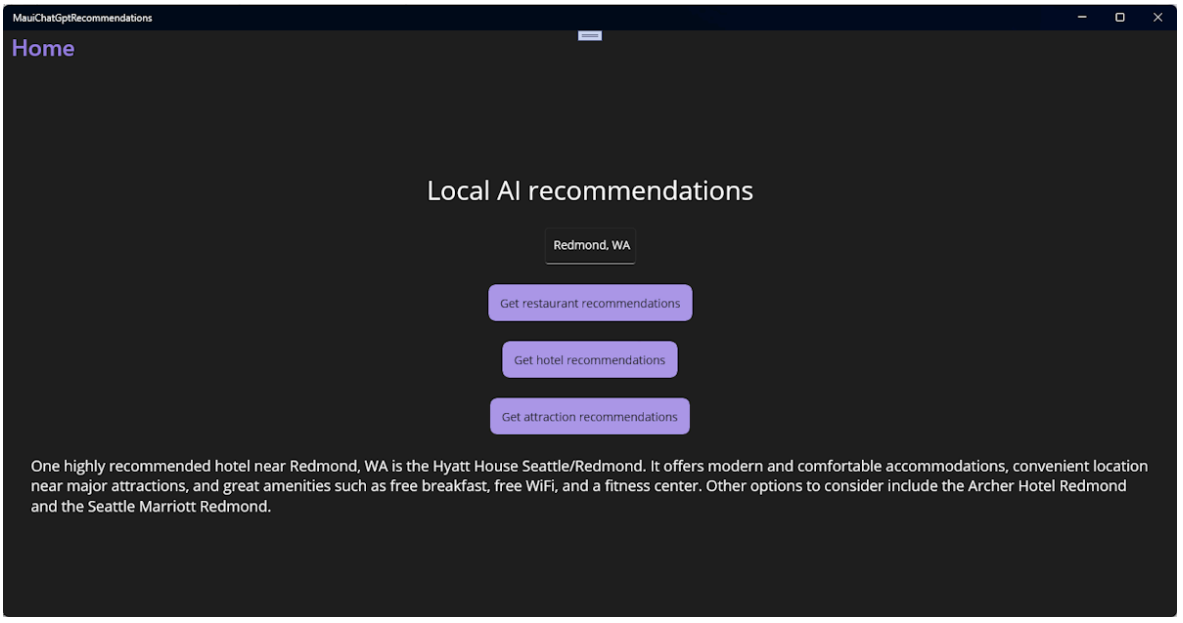
```
}

private async void OnHotelClicked(object sender, EventArgs e)
{
    await GetRecommendationAsync("hotel");
}

private async void OnAttractionClicked(object sender, EventArgs e)
{
    await GetRecommendationAsync("attraction");
}

private async Task GetRecommendationAsync(string recommendationType)
{
    if (string.IsNullOrWhiteSpace(LocationEntry.Text))
    {
        await DisplayAlert("Empty location", "Please enter a location
(city or postal code)", "OK");
        return;
    }

    // The model passed to GetChatClient must match an available OpenAI
    // model in your account.
    var client = _chatGptClient.GetChatClient("gpt-3.5-turbo-16k");
    string prompt = $"What is a recommended {recommendationType} near
{LocationEntry.Text}";

    AsyncResultCollection<StreamingChatCompletionUpdate> updates =
client.CompleteChatStreamingAsync(prompt);
    StringWriter responseWriter = new();

    await foreach (StreamingChatCompletionUpdate update in updates)
    {
        foreach (ChatMessageContentPart updatePart in
update.ContentUpdate)
        {
            responseWriter.Write(updatePart.Text);
        }
    }

    var returnMessage = responseWriter.ToString();
    SmallLabel.Text = returnMessage;
}
```

This code first checks to make sure the user has entered a location in the `Entry` control. If not, it displays an alert and returns. If the user has entered a location, it gets a `ChatClient` instance and calls the `CompleteChatStreamingAsync` method on the that object to make a call to OpenAI. The `CompleteChatStreamingAsync` method takes a `ChatMessage[]` parameter, which can be provided as a string, as we're doing in the example.

The method returns a `AsyncResultCollection<StreamingChatCompletionUpdate>`, which will stream responses from the API as they're generated. We iterate asynchronously over the `updates` and subsequently iterate over each `ChatMessageContentPart` in the response to build the response string. The response is then displayed in the `SmallLabel` control at the bottom of the UI.

> **Note**
>
> It's' a good idea to add error handling to the `GetRecommendationAsync` method to handle any exceptions that might be thrown by the API call. We've left that out of this example for simplicity.

2. Run the app, enter a location, and test the recommendation buttons. You should see a response from the API in the `Label` control at the bottom of the UI:



That's it! You've successfully created a Windows .NET MAUI app that uses the OpenAI ChatGPT API to provide recommendations for restaurants, hotels, and attractions. Try changing the prompts to see if you can improve the results. You can also try changing the model name passed to `GetChatClient` in `GetRecommendationAsync` to see if you get better results from a different model.

> **Important**
>
> Remember to keep an eye on your API usage after your trial period has expired. You can also set monthly spending limits on your OpenAI account to avoid unexpected charges.

Next steps

Advance to the next article to learn how to...

Add DALL-E to your .NET MAUI Windows desktop app

See also

- Create a .NET MAUI app with C# Markup and the Community Toolkit
- Build Windows apps with .NET MAUI
- Announcing the official OpenAI library for .NET
- Developing Responsible Generative AI Applications and Features on Windows
- Microsoft DevRadio Video: How to build an app with OpenAI and .NET MAUI

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Quickstart: Add DALL-E to your .NET MAUI Windows desktop app

Article • 06/17/2024

In this quickstart, we'll demonstrate how to integrate DALL-E's image generation capabilities into your .NET MAUI Windows desktop app.

Prerequisites

- Visual Studio 2022 17.8 or greater, with the .NET Multi-platform App UI workload installed. For more information, see Installation.
- A functional .NET MAUI project with OpenAI integration into which this capability will be integrated. See *Create a recommendation app with .NET MAUI and ChatGPT* - we'll demonstrate how to integrate DALL-E into the user interface from this how-to.
- An OpenAI API key from your OpenAI developer dashboard.
- A .NET OpenAI NuGet package version 2.0.0 or later installed in your project. This version is currently in pre-release. If you've followed along with the .NET MAUI ChatGPT tutorial, you will have this dependency installed and configured.

What problem will we solve?

You want to add DALL-E's image generation capabilities to your .NET MAUI Windows desktop app to provide users with a rich, interactive experience. They can already use the app to generate text-based recommendations, and you want to add the ability to generate images that visualize an activity in the location they have entered.

Set your environment variable

In order to use the OpenAI SDK, you'll need to set an environment variable with your API key. In this example, we'll use the `OPENAI_API_KEY` environment variable. Once you have your API key from the OpenAI developer dashboard, you can set the environment variable from the command line as follows:

PowerShell

```
setx OPENAI_API_KEY <your-api-key>
```

Note that this method works for development on Windows, but you'll want to use a more secure method for production apps and for mobile support. For example, you can store your API key in a secure key vault that a remote service can access on behalf of your app. See Best practices for OpenAI key safety for more information.

Install and initialize the OpenAI library for .NET

In this section, we'll install the SDK into the .NET MAUI project and initialize it with your OpenAI API key.

1. If you haven't already installed the `OpenAI` NuGet package, you can do so by running `dotnet add package OpenAI -IncludePrerelease` from Visual Studio's terminal window.

2. Once installed, you can initialize the `OpenAIClient` instance from the SDK with your OpenAI API key in `MainPage.xaml.cs` as follows:

```
C#

private OpenAIClient _chatGptClient;

public MainPage()
{
    InitializeComponent();
    this.Loaded += MainPage_Loaded;
}

private void MainPage_Loaded(object sender, EventArgs e)
{
    var openAiKey =
Environment.GetEnvironmentVariable("OPENAI_API_KEY");

    _chatGptClient = new(openAiKey);
}
```

Modify your app's UI

Next, we'll modify the user interface to include an `Image` control that displays a generated image below the recommendation text.

1. If you are are starting with a new project, copy the XAML for `MainPage.xaml` from the Create a recommendation app with .NET MAUI and ChatGPT tutorial.

2. Add a `StackLayout` containing a `Label` control and a `CheckBox` control to `MainPage.xaml` below the `LocationEntry` control to allow users to select whether to generate an image:

```
XML
```

```
...
<Entry
    x:Name="LocationEntry"
    Placeholder="Enter your location"
    SemanticProperties.Hint="Enter the location for recommendations"
    HorizontalOptions="Center"/>

<!-- Add this markup -->
<StackLayout Orientation="Horizontal"
                HorizontalOptions="Center">
    <Label Text="Generate image" VerticalOptions="Center"/>
    <CheckBox x:Name="IncludeImageChk" VerticalOptions="Center"/>
</StackLayout>
...
```

3. Add an `Image` control below the `SmallLabel` control to display the generated image:

```
XML
```

```
...
    <Image x:Name="GeneratedImage"
           WidthRequest="256"
           HeightRequest="256"
           HorizontalOptions="Center"/>
</VerticalStackLayout>
```

Implement DALL-E image generation

In this section, we'll add a method to handle image generation and call it from the existing `GetRecommendationAsync` method to display the generated image.

1. If you are are starting with a new project, make sure your code in `MainPage.xaml.cs` matches the code from the Create a recommendation app with .NET MAUI and ChatGPT tutorial.

2. Add a method named `GetImageAsync` to handle image generation. The new method will call the OpenAI API to generate an image based on the prompt we'll

build in the next step. It returns an `ImageSource` object that is used to display the image in the UI:

```
C#
```

```
public async Task<ImageSource> GetImageAsync(string prompt)
{
    // Use the DALL-E 3 model for image generation.
    ImageClient imageClient = _chatGptClient.GetImageClient("dall-e-
3");

    // Generate an image based on the prompt with a 1024x1024
resolution, the default for DALL-E 3.
    ClientResult<GeneratedImage> response = await
imageClient.GenerateImageAsync(prompt,
        new ImageGenerationOptions
        {
            Size = GeneratedImageSize.W1024xH1024,
            ResponseFormat = GeneratedImageFormat.Uri
        });

    // Image generation responses provide URLs you can use to retrieve
requested image(s).
    Uri imageUri = response.Value.ImageUri;

    return ImageSource.FromUri(imageUri);
}
```

3. Add a using directive for the image generation classes at the top of the file:

```
C#
```

```
using OpenAI.Images;
```

4. Add the following code to the end of the `GetRecommendationAsync` method to conditionally call the `GetImageAsync` method and display the generated image:

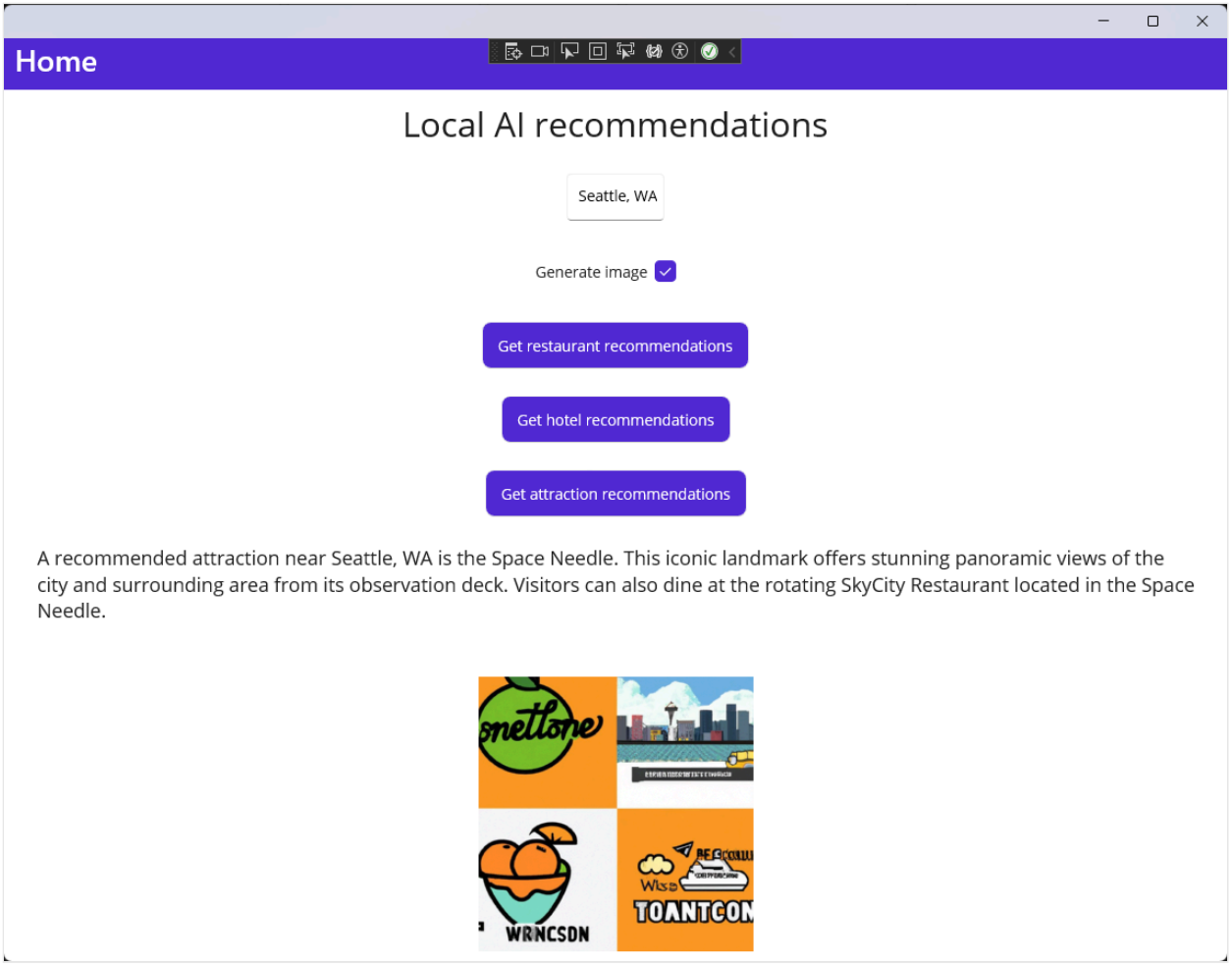
```
C#
```

```
if (IncludeImageChk.IsChecked)
{
    var imagePrompt = $"Show some fun things to do in
{LocationEntry.Text} when visiting a {recommendationType}.";
    GeneratedImage.Source = await GetImageAsync(imagePrompt);
}
```

The `imagePrompt` string is built based on the user's location input and the recommendation type selected.

Run and test

Run your app, enter a valid location, and click one of the recommendation buttons. You should see something like this:



How did we solve the problem?

We added DALL-E's image generation capabilities to our .NET MAUI Windows desktop app. Users can now generate images based on the location they enter and the type of recommendation they select. This provides a rich, interactive experience for users and enhances the app's functionality.

Clean up resources

It's important to make sure your OpenAI account is secure. If you're not planning to use the OpenAI API key for any other projects, you should delete it from your OpenAI developer dashboard. You should also set a reasonable spending limit on your OpenAI account to avoid any unexpected charges. You can check your current usage and spending in the OpenAI dashboard on the Usage page.

Related content

- .NET MAUI Installation
- Create a recommendation app with .NET MAUI and ChatGPT
- Add DALL-E to your WinUI 3 / Windows App SDK desktop app
- Create a .NET MAUI app with C# Markup and the Community Toolkit
- Developing Responsible Generative AI Applications and Features on Windows

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Tutorial: Create a .NET MAUI app with C# Markup and the Community Toolkit

Article • 04/27/2023

Build a **.NET MAUI** app with a user interface created without XAML by using **C# Markup** from the .NET MAUI Community Toolkit.

Introduction

The **.NET MAUI Community Toolkit** is a set of extensions, behaviors, animations, and other helpers. One of the features, C# Markup, provides the ability to create a user interface entirely in C# code. In this tutorial, you'll learn how to create a .NET MAUI app for Windows that uses C# Markup to create the user interface.

Setting up the environment

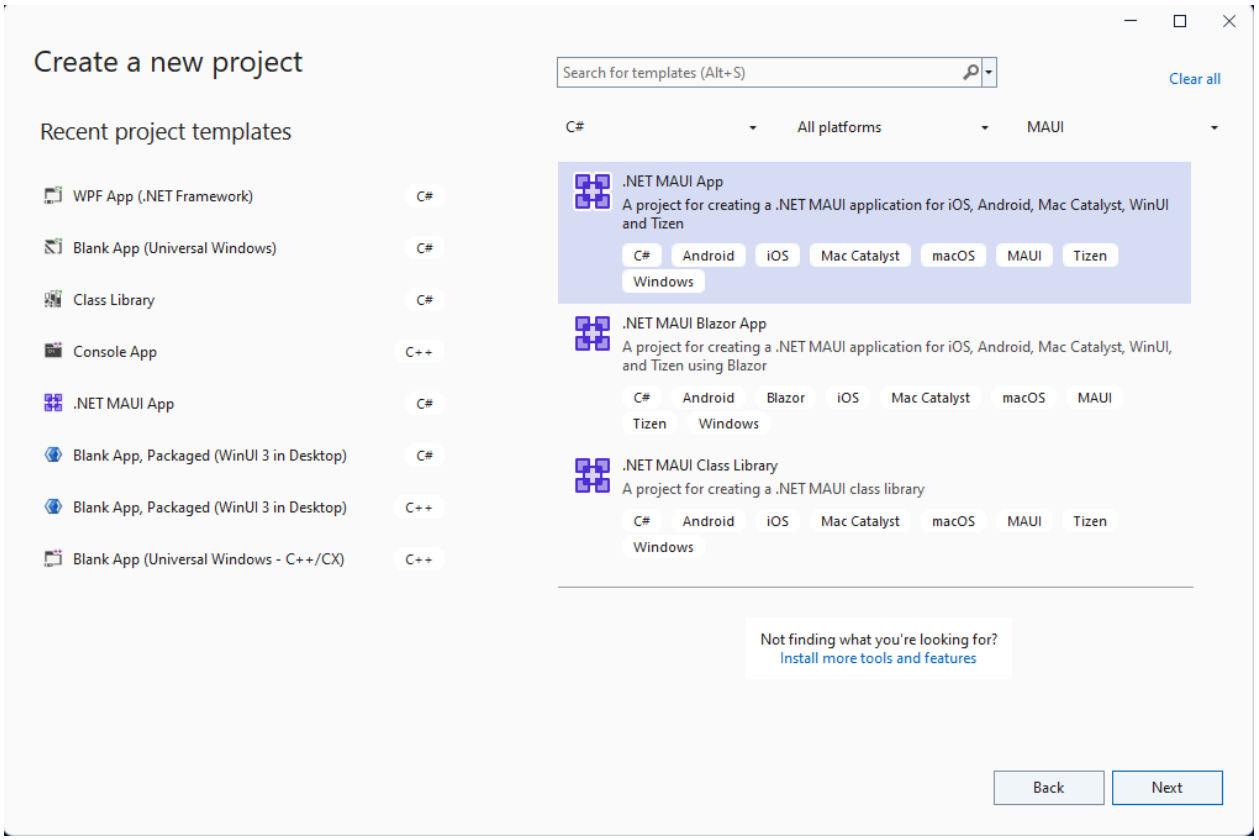
If you haven't already set up your environment for .NET MAUI development, please follow the steps to Get started with .NET MAUI on Windows.

Creating the .NET MAUI project

> ⓘ **Note**
>
> If you are already familiar with setting up a .NET MAUI project, you can skip to the next section.

Launch Visual Studio, and in the start window click **Create a new project** to create a new project.

In the **Create a new project** window, select **MAUI** in the All project types drop-down, select the **.NET MAUI App** template, and click the **Next** button:



Next, on the **Configure your new project** screen, give your project a name, choose a location for it, and click the **Next** button.

On the final screen, **Additional information**, click the **Create** button.

Wait for the project to be created, and for its dependencies to be restored.

In the Visual Studio toolbar, press the **Windows Machine** button to build and run the app. Click the **Click me** button and verify that the button content updates with the number of clicks.

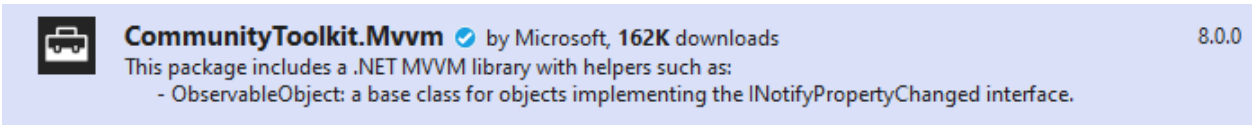
Now that you have verified that the .NET MAUI app on Windows is working as expected, we can integrate the MVVM Toolkit and C# Markup packages. In the next section, you'll add these packages to your new project.

Add C# Markup from the .NET MAUI Community Toolkit

Now that you have your .NET MAUI app running on Windows, let's add a couple of NuGet packages to the project to integrate with the MVVM Toolkit and **C# Markup** from the **.NET MAUI Community Toolkit**.

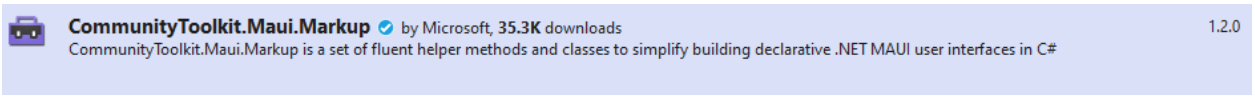
Right-click the project in **Solution Explorer** and select **Manage NuGet Packages...** from the context menu.

In the **NuGet Package Manager** window, select the **Browse** tab and search for **CommunityToolkit.MVVM**:



Add the latest stable version of the **CommunityToolkit.MVVM** package to the project by clicking **Install**.

Next, search for CommunityToolkit.Maui:



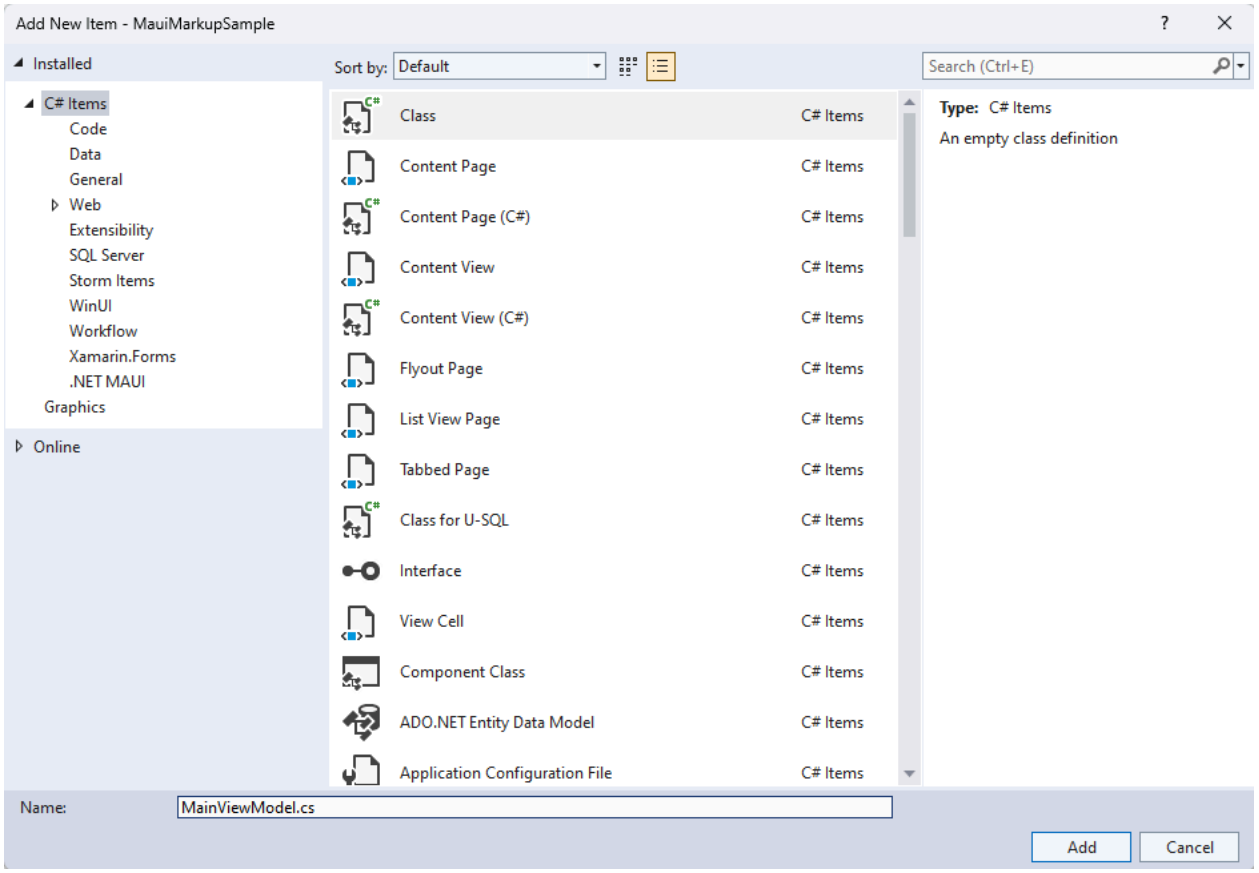
Add the latest stable version of the **CommunityToolkit.Maui.Markup** package to the project by clicking **Install**.

Close the **NuGet Package Manager** window after the new packages have finished installing.

Add a ViewModel to the project

We are going to add a simple **Model-View-ViewModel (MVVM)** implementation with the MVVM Toolkit. Let's start by creating a viewmodel to pair with our view (**MainPage**). Right-click the project again and select **Add | Class** from the context menu.

In the **Add New Item** window that appears, name the class **MainViewModel** and click **Add**:



We are going to leverage the power of the MVVM Toolkit in `MainViewModel`. Replace the contents of the class with the following code:

C#

```
using CommunityToolkit.Mvvm.ComponentModel;
using System.ComponentModel;
using System.Diagnostics;

namespace MauiMarkupSample
{
    [INotifyPropertyChanged]
    public partial class MainViewModel
    {
        [ObservableProperty]
        private string name;
        partial void OnNameChanging(string value)
        {
            Debug.WriteLine($"Name is about to change to {value}");
        }
        partial void OnNameChanged(string value)
        {
            Debug.WriteLine($"Name has changed to {value}");
        }
    }
}
```

If you have completed the Build your first .NET MAUI app for Windows tutorial, you will understand what the code above does. The `MainViewModel` class is decorated with the `INotifyPropertyChanged` attribute, which allows the MVVM Toolkit to generate the `INotifyPropertyChanged` implementation for the class. Marking `MainViewModel` as a `partial class` is required for the .NET source generator to work. The `ObservableProperty` attribute on the `name` private field will add a `Name` property for the class with the proper `INotifyPropertyChanged` implementation in the generated partial class. Adding the `OnNameChanging` and `OnNameChanged` partial methods is optional, but allows you to add custom logic when the `Name` property is changing or has changed.

Build a UI with C# Markup

When building a UI with C# Markup, the first step is to update the `CreateMauiApp()` method in **MauiProgram.cs**. Replace the contents of the method with the following code:

C#

```
public static MauiApp CreateMauiApp()
{
    var builder = MauiApp.CreateBuilder();

    builder
        .UseMauiApp<App>()
        .UseMauiCommunityToolkitMarkup();

    return builder.Build();
}
```

You also need to add a new `using` statement to the top of the file: `using CommunityToolkit.Maui.Markup;`. The call to `UseMauiCommunityToolkitMarkup()` will add the C# Markup support to the app, allowing you to construct your UI with C# code instead of XAML.

The **MainPage.xaml** file will no longer be used when rendering the UI, so you can remove the contents of the `ContentPage`.

XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```
             x:Class="MauiMarkupSample.MainPage">
</ContentPage>
```

In **MainPage.xaml.cs**, remove the click event handler and add three private members to the class:

C#

```
private readonly MainViewModel ViewModel = new();
private enum Row { TextEntry }
private enum Column { Description, Input }
```

The `ViewModel` property will create an instance of the `MainViewModel` class to be used when data binding the UI. The `Row` and `Column` enums will be used to define the layout of the UI with C# Markup. It's a simple UI with a single row and two columns which we'll be defining in the next step. You'll also need to add a namespace directive to the top of the file: `using static CommunityToolkit.Maui.Markup.GridRowsColumns;`.

Because the UI elements are going to be defined in the C# code, the `InitializeComponent()` method will not be needed. Remove the call and replace it with the following code to create the UI:

C#

```
public MainPage()
{
    BindingContext = ViewModel;
    Content = new Grid
    {
        RowDefinitions = Rows.Define(
            (Row.TextEntry, 36)),

        ColumnDefinitions = Columns.Define(
            (Column.Description, Star),
            (Column.Input, Stars(2))),

        Children =
        {
            new Label()
                .Text("Customer name:")
                .Row(Row.TextEntry).Column(Column.Description),

            new Entry
            {
                Keyboard = Keyboard.Numeric,
                BackgroundColor = Colors.AliceBlue,
            }.Row(Row.TextEntry).Column(Column.Input)
                .FontSize(15)
                .Placeholder("Enter name")
```

```
                .TextColor(Colors.Black)
                .Height(44)
                .Margin(6, 6)
                .Bind(Entry.TextProperty, nameof(ViewModel.Name),
BindingMode.TwoWay)
        }
    };
}
```

The new code in the `MainPage` constructor is leveraging C# Markup to define the UI. A `Grid` is set as the `Content` of the page. Our new grid defines a row with a height of 36 pixels and two columns with their widths defined using **Star** values, rather than absolute pixel values. The `Input` column will always be twice the width of the `Description` column. The equivalent XAML for these definitions would be:

XAML

```
<Grid.RowDefinitions>
    <RowDefinition Height="36" />
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="*" />
    <ColumnDefinition Width="2*" />
</Grid.ColumnDefinitions>
```

The rest of the code to create the grid adds two `Children`, a `Label` and an `Entry`. The `Text`, `Row`, and `Column` properties are set on the `Label` element, and the `Entry` is created with the following properties:

Expand table

Property	Value	Description
`Row`	`Row.TextEntry`	Defines the row number.
`Column`	`Column.Input`	Defines the column number.
`FontSize`	`15`	Sets the font size.
`Placeholder`	`"Enter name"`	Sets the placeholder text to display when the element is empty.
`TextColor`	`Colors.Black`	Sets the text color.
`Height`	`44`	Sets the height of the element.
`Margin`	`6, 6`	Defines the margin around the element.

Property	Value	Description
`Bind`	`Entry.TextProperty, nameof(ViewModel.Name), BindingMode.TwoWay`	Binds the `Text` property of the element to the `Name` property of the view model using two-way data binding.

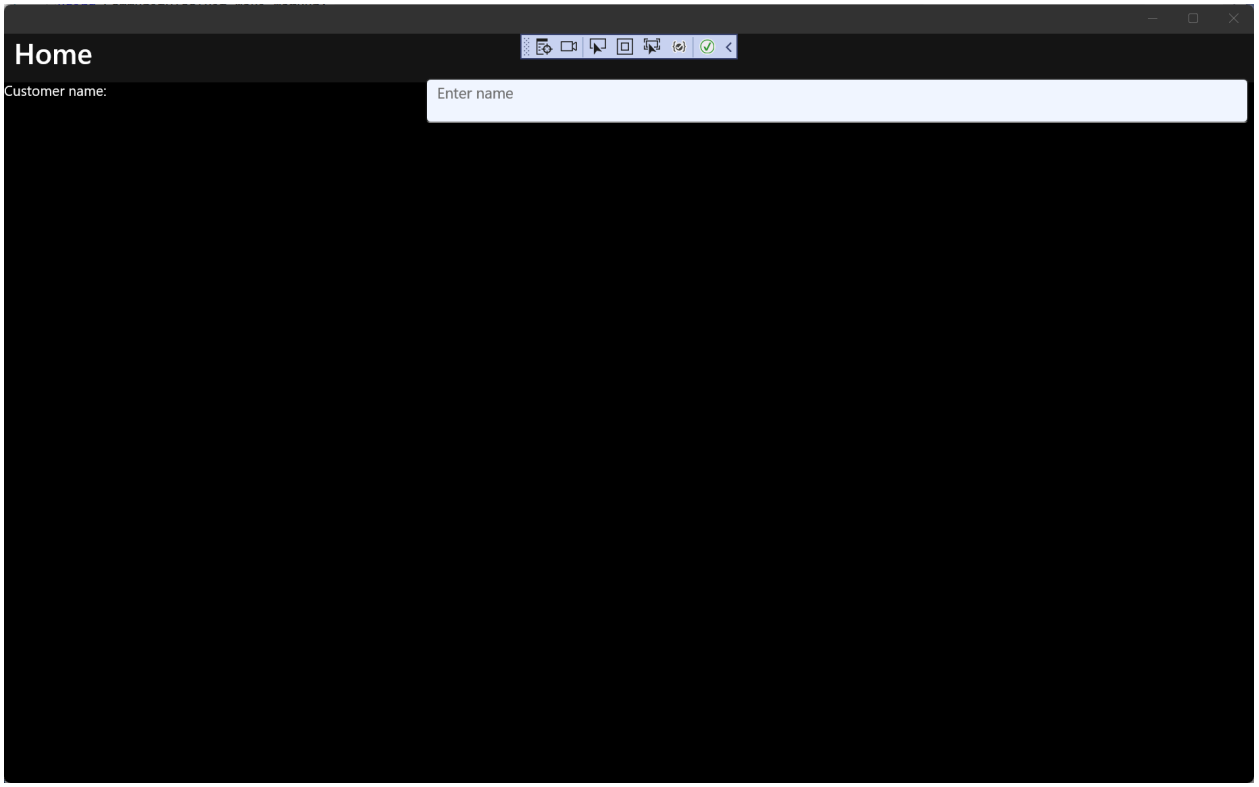
The equivalent XAML to define these child elements would be:

XAML

```
<Label Text="Customer name:"
       Grid.Row="0" Grid.Column="0" />
<Entry Grid.Row="1" Grid.Column="0"
       FontSize="15"
       Placeholder="Enter name"
       HeightRequest="44"
       Margin="6, 6"
       Text="{Binding Path=ViewModel.Name, Mode=TwoWay}" />
```

You may have noticed that the `TextColor` property is not set in the markup above. Setting the `TextColor` of a control requires setting a custom style. For more information about using styles in .NET MAUI, see Style apps using XAML. This is one example where setting properties in C# Markup can be more streamlined than the equivalent XAML. However, using styles in adds ease of reuse and inheritance.

You're now ready to run the app. Press **F5** to build and run the project. The app should look like the following screenshot:



You've now created your first C# Markup app on Windows with .NET MAUI. To learn more about what you can do with C# Markup, see C# Markup documentation.

Related topics

Resources for learning .NET MAUI

.NET MAUI Community Toolkit documentation

C# Markup documentation

Migrate from UWP to the Windows App SDK

Article • 09/10/2024

To migrate your app from the Universal Windows Platform (UWP) to the Windows App SDK, your UI code likely needs just a few namespace changes, while much of your platform code can stay the same. You'll need to adjust some code due to differences between UWP apps and desktop apps. But we expect that for most apps (depending on codebase size, of course), migration will take on the order of days rather than weeks.

> **Note**
>
> Your existing UWP app will continue to function as expected. However, to take advantage of modern features in **WinUI 3** and the **Windows App SDK** we recommend migrating your app.

Below are the steps, at a high level, for migrating manually. But see the following section for info about using the .NET Upgrade Assistant tool).

1. Create a new WinUI 3 packaged desktop project (see Create your first WinUI 3 project). That could go into your project's existing solution.
2. Copy your XAML/UI code. In many cases you can simply change namespaces (for example, **Windows.UI.*** to **Microsoft.UI.***).
3. Copy your app logic code. Some APIs need tweaks, such as **Popup**, **Picker**s, and **SecondaryTile**s.

For full details, see the topics in the table below. They describe and demonstrate how to manually migrate your Universal Windows Platform (UWP) application to the Windows App SDK.

Take particular note of What's supported when migrating from UWP to WinUI 3, which describes any functionality that's not yet supported in WinUI 3 and the Windows App SDK. If your app needs any of those features/libraries, then you might want to consider postponing migration.

> **Note**
>
> If your UWP app's source code is written in C++/CX, then also see **Move to C++/WinRT from C++/CX**.

Migrating by using the .NET Upgrade Assistant tool

As a further step in assisting you to migrate your UWP apps to the Windows App SDK and WinUI 3, we've leveraged the .NET Upgrade Assistant, adding support for migrating C# UWP apps. The UWP support automates much of the migration process. For more info, see the topic Migrate from UWP to the Windows App SDK with the .NET Upgrade Assistant.

Containerization benefits

When transitioning to the Windows App SDK, UWP applications might lose the inherent containerization benefits of their original platform. However, those benefits can be restored by using Win32 App Isolation—a new security feature that enhances protection by isolating applications within a sandbox environment. This approach offers additional security against potential threats with minimal changes to your existing code. For more info, and to begin using Win32 App Isolation, see Win32 app isolation overview.

Topics in this section

Expand table

Topic	Description
Overall migration strategy	Considerations and strategies for approaching the migration process, and how to set up your development environment for migrating.
Mapping UWP features to the Windows App SDK	This topic compares major feature areas in the different forms in which they appear in UWP and in the Windows App SDK.
What's supported	Learn what features are currently available in WinUI 3 Desktop to evaluate whether you should attempt migrating your UWP app today.
Mapping UWP APIs and libraries to Windows App SDK	This topic provides a mapping of UWP APIs and libraries to their Windows App SDK equivalents.
Feature area guides	A collection of migration guidance topics, each focusing on a specific feature area.
Case study 1—PhotoLab (C#)	This topic is a case study of taking the C# UWP PhotoLab sample app, and migrating it to the Windows App SDK.

Topic	Description
Case study 2—Photo Editor (C++/WinRT)	This topic is a case study of taking the C++/WinRT UWP Photo Editor sample app, and migrating it to the Windows App SDK.
Additional migration guidance	This topic contains additional migration guidance not categorized into a feature area in the feature area guides.
Migrate from UWP to the Windows App SDK with the .NET Upgrade Assistant	The .NET Upgrade Assistant is a command-line tool that can assist with migrating a C# UWP app to a WinUI 3 app that uses the Windows App SDK.

See Also

- Windows App SDK and supported Windows releases

Overall migration strategy

Article • 03/27/2023

Introduction

The Windows App SDK provides a broad set of Windows APIs—with implementations that are decoupled from the OS, and released to developers via NuGet packages. As a developer with a Universal Windows Platform (UWP) application, you can make great use of your existing skill set, and your source code, by moving your app to the Windows App SDK.

With the Windows App SDK you can incorporate the latest runtime, language, and platform features into your app. Since each application is different, and so are your requirements and preferences, there are many different ways to approach the process of migrating your app's source code. But the high-level approach, and code changes needed for various feature areas, are similar. So in this topic we'll review strategies on how you can approach migrating your app, as well as more info (and limitations) about certain feature areas. So also see What's supported when porting from UWP to WinUI 3.

Most Windows Runtime (WinRT) APIs can be used by Windows App SDK apps. But there are some that aren't supported in desktop apps, or have restrictions.

- APIs that have dependencies on UI features that were designed for use only in UWP apps.
- APIs that require package identity. These APIs are supported only in desktop apps that are packaged using MSIX.

For those APIs, we'll show you what alternatives to use. Most of those alternatives are available in WinUI, or via WinRT COM interfaces that are available in the Windows App SDK.

For example, we'll see certain UI scenarios where you'll need to track your main window object, and use various **HWND**-based APIs and interoperation patterns, such as **IInitializeWithWindow::Initialize**.

> **Note**
>
> Also see **Windows Runtime APIs not supported in desktop apps**. Windows App SDK apps are *one* kind of desktop app. Other kinds of desktop app include .NET desktop apps, and C/C++ Win32 desktop apps. The audience of that topic is

developers wishing to migrate to anything in the union of those different kinds of desktop app, including (but not limited to) Windows App SDK apps.

We'd love to hear your feedback about this migration guide, and about your own migration experience. Use the **Feedback** section right at the foot of this page like this:

- For questions and feedback about the Windows App SDK, or just to start a discussion, use the **This product** button. You can also start a discussion on the Discussions tab of the **WindowsAppSDK** GitHub repo. Using those channels, you could also tell us what problem you're trying to solve, how you've tried to solve it so far, and what would be the ideal solution for your app.
- For feedback about missing or incorrect information in this migration guide, use the **This page** button.

Why migrate to the Windows App SDK?

The Windows App SDK offers you an opportunity to enhance your app with new platform features, as well as with the modern Windows UI 3 Library (WinUI 3), which is developed and designed to delight your users.

In addition, the Windows App SDK is backward compatible; it supports apps down to Windows 10, version 1809 (10.0; Build 17763)—also known as the Windows 10 October 2018 Update.

The value proposition of moving the Windows App SDK is manifold. Here are some considerations:

- Latest user interface (UI) platform and controls such as WinUI 3 and WebView2.
- A single API surface across desktop app platforms.
- More frequent release cadence that releases separately from Windows.
- A consistent experience across Windows versions.
- .NET compatibility.
- Backward-compatible down to Windows 10, version 1809.
- Improved runtime environment. See MSIX container.

For more info, see Windows App SDK.

Overview of the migration process

> **Note**

You can think of the UWP version of the app that you want to migrate as the *source* solution/project (it's the *source* of the migration process). The Windows App SDK version is the *target* solution (it's the *target* of the migration process).

Install the Windows App SDK VSIX

Download the Windows App SDK Visual Studio extension (VSIX) installer from Stable release channel for the Windows App SDK, and run to install it.

Create a new project

In Visual Studio, Create your first WinUI 3 project. For example, use the **Blank App, Packaged (WinUI 3 in Desktop)** project template. You can find that project template in the **Create a new project** dialog by choosing language: *C#* or *C++*; platform: *Windows App SDK*; project type: *WinUI* or *Desktop*.

You'll see two projects in **Solution Explorer**—one is qualified as **(Desktop)**, and the other as **(Package)**.

Migrate code with the least dependencies first

To illustrate this recommendation, let's take the PhotoLab case study as an example. For the PhotoLab sample app, one perhaps obvious approach might be to begin by migrating **MainPage**—since that's such an important and prominent piece of the app. But if we were to do that, then we'd soon realize that **MainPage** has a dependency on the **DetailPage** view; and then that **DetailPage** has a dependency on the **Photo** model. If we were to follow that path, then we might be constantly interrupting ourselves (switching over to work on each newly discovered dependency). Certainly we wouldn't expect to get a clean *build* until we'd chased down every dependency, and essentially ported the whole project in one giant leap.

If on the other hand we were to begin with the piece of the project that doesn't depend on anything else, and work outward from there (from least- to most-dependent piece), then we'd be able to focus on each piece one at a time. And we'd also be able to achieve a clean build after migrating each piece, and in that way confirm that the migration process is staying on track.

So when you're migrating your own apps, we recommend that you migrate code with the least dependencies first.

Copy files, or copy file contents?

When you migrate, you will of course be copying over asset *files* (and not asset file *contents*). But what about source code files? As an example let's again take the **MainPage** class from the PhotoLab case study and the Photo Editor case study.

C#. In the C# version (PhotoLab), **MainPage** is made up of the source code files `MainPage.xaml` and `MainPage.xaml.cs`.

C++/WinRT. In the C++/WinRT version (Photo Editor), **MainPage** is made up of the source code files `MainPage.xaml`, `MainPage.idl`, `MainPage.h`, and `MainPage.cpp`.

So you have the choice between these two options:

- (Recommended) you can copy over *the files themselves* (using **File Explorer**), and then add the copies to the target project. Exceptions to this recommendation are files such as `App.xaml` and `App.xaml.cs`, since those files already exist in the target project, and they contain source code that's specific to the Windows App SDK. For those you'll need to *merge* the source code that's already there with source code from the source project.
- Or you can use Visual Studio to create new **Page** files (such as `MainPage.xaml` and `MainPage.xaml.cs`) in the target project, and then copy the *contents* of those source code files over from the source project to the target project. For a C++/WinRT project, this option involves a lot more steps.

Also see the section MainPage and MainWindow.

Folder and file name differences (C++/WinRT)

In a C++/WinRT UWP project, code-behind files for XAML views are named of the form `MainPage.h` and `MainPage.cpp`. But in a C++/WinRT Windows App SDK, you'll need to rename those to `MainPage.xaml.h` and `MainPage.xaml.cpp`.

In a C++/WinRT UWP project, when migrating a hypothetical XAML view (in the sense of models, views, and viewmodels) named **MyPage**, in `MyPage.xaml.cpp` you'll need to add `#include "MyPage.g.cpp"` immediately after `#include "DetailPage.xaml.h"`. And for a hypothetical model named **MyModel**, in `MyModel.cpp` add `#include "MyModel.g.cpp"` immediately after #include "MyModel.h". For an example, see Migrate DetailPage source code.

If you change the name of your migrated project

While migrating, you might choose to give your target project a different name from that of the source project. If you do, then that will affect the default namespace within the source project. As you copy source code over from the source project to the target project, you might need to change namespace names that are mentioned in the source code.

Changing project name (and consequently default namespace name) is something that we do, for example, in the case study A Windows App SDK migration of the UWP PhotoLab sample app (C#). In that case study, instead of copying over file *contents* from the source to the target project, we copy entire files using File Explorer. The source project/namespace name is *PhotoLab*, and the target project/namespace name is *PhotoLabWinUI3*. So we need to search for *PhotoLab* in the contents of any source code files we copied over, and change that to *PhotoLabWinUI3*

In that particular case study, we made those changes in `App.xaml`, `App.xaml.cs`, `MainPage.xaml`, `MainPage.xaml.cs`, `DetailPage.xaml`, `DetailPage.xaml.cs`, `ImageFileInfo.cs`, and `LoadedImageBrush.cs`.

Install the same NuGet packages that were installed in the source project

One task during the migration process is to identify any NuGet packages that the source project has dependencies on. And then install those same NuGet packages in the target project.

For example, in the case study A Windows App SDK migration of the UWP PhotoLab sample app (C#), the source project contains references to the **Microsoft.Graphics.Win2D** NuGet package. So in that case study we add a references to the same NuGet package to the target project.

Related topics

- Windows App SDK and supported Windows releases
- Windows Runtime (WinRT) APIs
- WinUI
- Stable release channel for the Windows App SDK
- PhotoLab case study

- Photo Editor case study
- Windows Runtime APIs not supported in desktop apps

Mapping UWP features to the Windows App SDK

Article • 03/19/2025

This topic compares major feature areas in the different forms in which they appear in UWP and in the Windows App SDK. The content in this migration guide supports moving from UWP XAML to Windows App SDK XAML—moving to a different UI framework, such as Windows Presentation Foundation (WPF), is outside of the scope of this guidance.

Expand table

Feature	UWP	Windows App SDK (packaged apps)	Migration notes
Packaging	MSIX App has identity	MSIX App has identity	UWP apps migrating to the Windows App SDK should stay on MSIX to ensure trusted clean install and uninstall experience, as well as access to all APIs, including those that require identity.
Container	App container: - security = LowIL - file system access is brokered - no registry access	MSIX Container: - security = MediumIL - file system access same as user, AppData writes virtualized - HKCU registry writes virtualized	Moving to a higher integrity level with the Windows App SDK allows your app to have greater functionality. However, be aware of virtualization if you want to expand the capabilities of your migrated application to write to HKCU or AppData.
Activation and instancing	Package identity + CoreApplication activation, single-instanced by default	Package identity, Main/WinMain + Windows App SDK activation, multi-instanced by default	Ensure your application can handle multi-instance behavior, or use AppInstance to manage your instances.
Lifecycle-managed	Suspend/resume	Power/State notifications	You can use Power/State change notifications to reduce system load.

Feature	UWP	Windows App SDK (packaged apps)	Migration notes
Background tasks	InProc and OOP background tasks	Full trust COM background task implementation	For more info, see Background task migration strategy.
Windowing	CoreWindow, AppWindow (preview)	HWND, AppWindow v2	Windowing behavior has significantly changed in Windows App SDK. See Windowing functionality migration.
Messaging	CoreDispatcher and DispatcherQueue	DispatcherQueue, WndProc	**DispatcherQueue** supports Win32 apps. For additional details on moving from CoreDispatcher to DispatcherQueue see Threading functionality migration.
UI Platform	System XAML, WebView, DirectX, and others	WinUI 3, Webview2, DirectX, and others	For more info, see WinUI migration.
Text-rendering	DirectWrite	DWriteCore	Enables applications to access the latest DWrite features downlevel and receive new DWrite updates separate from the OS release schedule. For more info, see DirectWrite to DWriteCore migration.
Resources	MRT	MRTCore	For more info, see MRT to MRTCore migration.
.NET Runtime	.NET Native / C# 7	.NET 6+/C# 9	The Windows App SDK provides access to the modern .NET runtime, and access to new language features. However, .NET ReadyToRun compilation is not the same as .NET Native, so you should evaluate performance tradeoffs.
2D Graphics	Win2D	Win2D for WinUI 3	We're currently working on a version of Win2D that works with the

Feature	UWP	Windows App SDK (packaged apps)	Migration notes
			Windows App SDK, in progress. See the documentation for more information.
Web authentication	WebAuthenticationBroker	OAuth2Manager	The Windows App SDK provides a new API for OAuth 2.0 functionality. See Implement OAuth functionality in Windows apps for more information.
Windows Runtime components	Windows Runtime component project templates for UWP	- C++: Use the **Windows Runtime Component (WinUI 3)** project template. - C#: Use C#/WinRT to author Windows Runtime Components in a .NET Class Library.	We're currently working on support to Author Windows Runtime Components using C#/WinRT for use in the Windows App SDK and WinUI 3.

See Also

- Windows App SDK and supported Windows releases

What's supported when migrating from UWP to WinUI 3

Article • 03/19/2025

WinUI 3 and the Windows App SDK are new technologies and, when compared to UWP, there are some features that aren't supported. This topic provides information on which features are supported before you attempt migration.

Expand table

UWP feature	WinUI 3 status
Background acrylic	✅ Available via DesktopAcrylicController.
Background tasks	✅ Supported; see BackgroundTaskBuilder (introduced in 1.7).
Common UI controls	✅ Supported
CameraCaptureUI	✅ Supported; see CameraCaptureUI (introduced in 1.7). For alternative APIs, see Using video capture.
Composition/DirectX interop	✅ Most Composition and Drawing features are supported (global composition effects aren't supported in 1.7); see Enhance UI with the Visual layer.
Distributing via Store	✅ Supported
Live Tiles (on Windows 10)	✅ Supported
MapControl	✅ Supported; see MapControl (introduced in 1.5).
MediaElement and MediaPlayerElement	✅ Use MediaPlayerElement, which was introduced in 1.2.
MSAL library	✅ Supported
MSIX	✅ Supported
Single-instancing	✅ Supported
TaskbarManager API	✅ Supported; for details, see TaskbarManager desktop samples.
Toast notifications	✅ Supported
Visual Studio App Center	✅ Supported

UWP feature	WinUI 3 status
WebAuthenticationBroker	✅ Supported; see Microsoft.Security.Authentication.OAuth (introduced in 1.7).
Best launch speed and performance	⚠️ Slight disadvantage, see performance considerations.
CoreTextServicesManager	⚠️ Supported only on Windows 11
PrintManager	⚠️ Supported on Windows 11 (not yet available on Windows 10)
CoreWindow and related APIs	❌ Not supported in 1.7. For alternative APIs with some of the same functionality, see AppWindow, and HWND-based APIs.
Virtual key support for gamepad input	❌ Not supported in 1.7; see Gamepad support in WinUI 3
InkCanvas	❌ Not supported in 1.7
Single-app kiosk	❌ Not supported in 1.7
Xbox and HoloLens	❌ Not supported in 1.7

Performance considerations

Today in version 1.7 of the Windows App SDK, launch speeds, RAM usage, and installation size of WinUI 3 apps are larger/slower than seen in UWP. We're actively working to improve this.

Visual Studio

The **Design** tab of the XAML Designer in Visual Studio (and Blend for Visual Studio) doesn't currently support WinUI 3 projects (as of version 1.7 of the Windows App SDK). For more info, see Create a UI by using XAML Designer.

See Also

- Windows App SDK and supported Windows releases

Mapping UWP APIs and libraries to the Windows App SDK

Article • 06/16/2022

This topic provides a mapping of UWP APIs to their Windows App SDK equivalents. In some cases the mapping isn't one-to-one; either platform might have more or less functionality than the other for a namespace or class.

There are differences in the names of namespaces and classes (including UI controls) between the two platforms. In many cases, it's as easy as changing a namespace name and then your code will compile. Sometimes, a class or API name has changed as well as the namespace name. Other times, the mapping takes a bit more work, and in rare cases requires a change in approach.

How to use the table. First, search for the name of the class member you're using. Members are listed whenever the mapping is more complicated than a simple class name or namespace name change. If your member isn't listed, then search for the name of the class you're using. Classes are listed whenever the mapping is more complicated than simply changing the namespace name (for specific guidance, see the Feature area guides). If your class isn't listed, then the mapping is just a namespace change. So, find your class's namespace name and then you'll find the equivalent Windows App SDK namespace name. Your class will be in that namespace. If your namespace isn't listed, then its name hasn't changed.

Expand table

UWP	Windows App SDK
(**Windows.ApplicationModel.Activation**) LaunchActivatedEventArgs class	(**Microsoft.UI.Xaml**) LaunchActivatedEventArgs class; for example, in **App.OnLaunched**.
(**Windows.ApplicationModel.Background**) BackgroundTaskBuilder class	(**Microsoft.Windows.ApplicationModel.Background**) BackgroundTaskBuilder class. See Using background tasks in Windows apps.
(**Windows.ApplicationModel.Core**) CoreApplication.CreateNewView method	(**Microsoft.UI.Windowing**) AppWindow.Create method
(**Windows.ApplicationModel.Core**) CoreApplicationViewTitleBar class	(**Microsoft.UI.Windowing**) AppWindowTitleBar class
(**Windows.ApplicationModel.Core**) CoreApplicationViewTitleBar.ExtendViewIntoTitleBar property	(**Microsoft.UI.Windowing**) AppWindowTitleBar.ExtendsContentIntoTitleBar property. The platform continues to draw the **Minimize**/**Maximize**/**Close** buttons for you, and reports the occlusion information.

UWP	Windows App SDK
(**Windows.ApplicationModel.Core**) **CoreApplicationViewTitleBar.Height** property	(**Microsoft.UI.Windowing**) **AppWindowTitleBar.Height** property
(**Windows.ApplicationModel.Core**) **CoreApplicationViewTitleBar.SystemOverlayLeftInset** property	(**Microsoft.UI.Windowing**) **AppWindowTitleBar.LeftInset** property
(**Windows.ApplicationModel.Core**) **CoreApplicationViewTitleBar.SystemOverlayRightInset** property	(**Microsoft.UI.Windowing**) **AppWindowTitleBar.RightInset** property
Windows.ApplicationModel.Resources.Core namespace	**Microsoft.Windows.ApplicationModel.Resources**. See MRT to MRT Core migration.
(**Windows.ApplicationModel.Resources.Core**) **ResourceContext.GetForCurrentView** method	(**Microsoft.Windows.ApplicationModel.Resources**) **ResourceManager.CreateResourceContext** method. See ResourceContext.GetForCurrentView, and ResourceContext.GetForViewIndependentUse.
(**Windows.ApplicationModel.Resources.Core**) **ResourceContext.GetForViewIndependentUse** method	(**Microsoft.Windows.ApplicationModel.Resources**) **ResourceManager.CreateResourceContext** method. See ResourceContext.GetForCurrentView, and ResourceContext.GetForViewIndependentUse.
(**Windows.ApplicationModel.Resources.Core**) **ResourceManager.Current** property	Create a new (**Microsoft.Windows.ApplicationModel.Resources**) **ResourceManager** object. See ResourceManager class.
(**Windows.ApplicationModel.Resources.Core**) **ResourceQualifierObservableMap.MapChanged** event	Detect environment changes for yourself. See Resource qualifier value change.
(**Windows.Graphics.Printing**) **PrintManager** class	Not supported in Windows App SDK 1.0.
(**Windows.Media.Capture**) **CameraCaptureUI** class	Not supported in Windows App SDK 1.0.
(**Windows.Security.Authentication.Web**) **WebAuthenticationBroker** class	(**Microsoft.Security.Authentication.OAuth**) OAuth2Manager class (supported in Windows App SDK 1.7 and later). See Implement OAuth functionality in Windows apps for more information on using **OAuth2Manager** and related APIs for performing OAuth 2.0 authentication. See GitHub for a full sample application.
(**Windows.Storage.Pickers**) **FileOpenPicker**, **FileSavePicker**, and **FolderPicker** classes	Supported, but you must use the **IInitializeWithWindow** interface. See MessageDialog, and Pickers.
(**Windows.System.Display**) **DisplayRequest** class	Not supported in Windows App SDK 1.0.
Windows.UI.Composition namespace	**Microsoft.UI.Composition** namespace

UWP	Windows App SDK
(**Windows.UI.Core**) CoreDispatcher class	(**Microsoft.UI.Dispatching**) DispatcherQueue class. See Change CoreDispatcher to DispatcherQueue.
(**Windows.UI.Core**) CoreDispatcher.RunAsync method	(**Microsoft.UI.Dispatching**) DispatcherQueue.TryEnqueue method. See Change RunAsync to TryEnqueue.
(**Windows.UI.Core**) CoreWindow class	(**Microsoft.UI.Windowing**) AppWindow class. See Windowing functionality migration.
(**Windows.UI.Core**) CoreWindow.Bounds property (commonly appears in C# as `CoreWindow.GetForCurrentThread.Bounds`)	(**Microsoft.UI.Windowing**) AppWindow.Size property
(**Windows.UI.Core**) CoreWindow.GetForCurrentThread method	No direct 1:1 mapping to a Windows App SDK API. When using XAML, you can get **Window.AppWindow** to get the **AppWindow** associated with a XAML **Window**, but an app needs to cache the **Window** or **AppWindow** if it wants to access it from somewhere that doesn't otherwise have access. We recommend caching and exposing the **Window** on the **App** object.
(**Windows.UI.Core**) CoreWindow.Activate method	(**Microsoft.UI.Windowing**) AppWindow.Show method
(**Windows.UI.Core**) CoreWindow.Dispatcher property	(**Microsoft.UI.Xaml**) Window.DispatcherQueue property. See Change CoreWindow.Dispatcher to Window.DispatcherQueue.
(**Windows.UI.Core**) CoreWindow.SizeChanged event	(**Microsoft.UI.Windowing**) AppWindowChangedEventArgs.DidSizeChange method
(**Windows.UI.Core**) SystemNavigationManager class (global back button)	Not supported. Implement a back button within your own UI. See A Windows App SDK migration of the UWP PhotoLab sample app (C#).
(**Windows.UI.Core**) WindowSizeChangedEventArgs class	(**Microsoft.UI.Xaml**) WindowSizeChangedEventArgs class; for example, in a handler for (**Microsoft.UI.Xaml**) Window.SizeChanged.
(**Windows.UI.Popups**) MessageDialog class	Supported, but you must use the IInitializeWithWindow interface. See MessageDialog, and Pickers.
(**Windows.UI.Text.Core**) CoreTextServicesManager class	Supported only on Windows 11.
(**Windows.UI.ViewManagement**) AccessibilitySettings.HighContrastChanged event	(**Microsoft.UI.System**) ThemeSettings.Changed event

UWP	Windows App SDK
(Windows.UI.ViewManagement) ApplicationView class	(Microsoft.UI.Windowing) AppWindow class
(Windows.UI.ViewManagement) ApplicationView.Title property	(Microsoft.UI.Windowing) AppWindow.Title property
(Windows.UI.ViewManagement) ApplicationView.TryConsolidateAsync method	(Microsoft.UI.Windowing) AppWindow.Destroy method
(Windows.UI.ViewManagement) ApplicationView.TryEnterFullScreenMode method	(Microsoft.UI.Windowing) AppWindow.SetPresenter method with AppWindowPresenterKind.FullScreen
(Windows.UI.ViewManagement) ApplicationView.TryEnterViewModeAsync method with ApplicationViewMode.CompactOverlay	(Microsoft.UI.Windowing) AppWindow.SetPresenter method with AppWindowPresenterKind.CompactOverlay
(Windows.UI.ViewManagement) ApplicationView.TryResizeView method	(Microsoft.UI.Windowing) AppWindow.Resize method
(Windows.UI.ViewManagement) ApplicationViewMode.CompactOverlay field	(Microsoft.UI.Windowing) AppWindowPresenterKind.CompactOverlay field
(Windows.UI.ViewManagement) ApplicationViewTitleBar class	(Microsoft.UI.Windowing) AppWindowTitleBar class
(Windows.UI.ViewManagement) ApplicationViewWindowingMode.FullScreen field	(Microsoft.UI.Windowing) AppWindowPresenterKind.FullScreen field
(Windows.UI.WindowManagement) AppWindow class	(Microsoft.UI.Windowing) AppWindow class
(Windows.UI.WindowManagement) AppWindow.Changed event	(Microsoft.UI.Windowing) Microsoft.UI.Windowing.AppWindow.Changed event
(Windows.UI.WindowManagement) AppWindow.CloseAsync method	(Microsoft.UI.Windowing) AppWindow.Destroy method
(Windows.UI.WindowManagement) AppWindow.GetPlacement method	(Microsoft.UI.Windowing) AppWindow.Position property
(Windows.UI.WindowManagement) Appwindow.RequestMoveXxx methods	(Microsoft.UI.Windowing) AppWindow.Move method
(Windows.UI.WindowManagement) AppWindow.RequestSize method	(Microsoft.UI.Windowing) AppWindow.Resize method
(Windows.UI.WindowManagement) AppWindow.Title property	(Microsoft.UI.Windowing) AppWindow.Title property
(Windows.UI.WindowManagement) AppWindow.TryCreateAsync method	(Microsoft.UI.Windowing) AppWindow.Create method

UWP	Windows App SDK
(**Windows.UI.WindowManagement**) AppWindow.TryShowAsync method	(**Microsoft.UI.Windowing**) AppWindow.Show method
(**Windows.UI.WindowManagement**) AppWindowChangedEventArgs.DidSizeChange method	(**Microsoft.UI.Windowing**) AppWindowChangedEventArgs.DidSizeChange method
(**Windows.UI.WindowManagement**) AppWindowPlacement.Size property	(**Microsoft.UI.Windowing**) AppWindow.Size property
(**Windows.UI.WindowManagement**) AppWindowPresentationKind.CompactOverlay field	(**Microsoft.UI.Windowing**) AppWindowPresenterKind.CompactOverlay field
(**Windows.UI.WindowManagement**) AppWindowPresentationKind.FullScreen field	(**Microsoft.UI.Windowing**) AppWindowPresenterKind.FullScreen field
(**Windows.UI.WindowManagement**) AppWindowPresenter.RequestPresentation method	(**Microsoft.UI.Windowing**) AppWindow.SetPresenter method
(**Windows.UI.WindowManagement**) AppWindowTitleBar class	(**Microsoft.UI.Windowing**) AppWindowTitleBar class
(**Windows.UI.WindowManagement**) AppWindowTitleBar.GetPreferredVisibility	No direct 1:1 mapping to a Windows App SDK API
(**Windows.UI.WindowManagement**) AppWindowTitleBar.GetTitleBarOcclusions	(**Microsoft.UI.Windowing**) AppWindowTitleBar.LeftInset and RightInset, coupled with Height, provide the same information.
(**Windows.UI.WindowManagement**) AppWindowTitleBar.SetPreferredVisibility	No direct 1:1 mapping to a Windows App SDK API
(**Windows.UI.WindowManagement**) AppWindowTitleBarOcclusion property	(**Microsoft.UI.Windowing**) AppWindowTitleBar.LeftInset and RightInset, coupled with Height, provide the same information.
(**Windows.UI.WindowManagement**) AppWindowTitleBarVisibility	No direct 1:1 mapping to a Windows App SDK API
Windows.UI.Xaml namespace	Microsoft.UI.Xaml namespace
(**Windows.UI.Xaml**) Application.OnActivated method (commonly appears in C# as `App.OnActivated`)	See OnActivated, OnBackgroundActivated, and other activation-handling methods.
(**Windows.UI.Xaml**) Application.OnBackgroundActivated method (commonly appears in C# as `App.OnBackgroundActivated`)	See OnActivated, OnBackgroundActivated, and other activation-handling methods.
(**Windows.UI.Xaml**) Application.OnCachedFileUpdaterActivated method	See OnActivated, OnBackgroundActivated, and other activation-handling methods.

UWP	Windows App SDK
(commonly appears in C# as `App.OnCachedFileUpdaterActivated`)	
(**Windows.UI.Xaml**) **Application.OnFileActivated** method (commonly appears in C# as `App.OnFileActivated`)	See File type association and OnActivated, OnBackgroundActivated, and other activation-handling methods.
(**Windows.UI.Xaml**) **Application.OnFileOpenPickerActivated** method (commonly appears in C# as `App.OnFileOpenPickerActivated`)	See OnActivated, OnBackgroundActivated, and other activation-handling methods.
(**Windows.UI.Xaml**) **Application.OnFileSavePickerActivated** method (commonly appears in C# as `App.OnFileSavePickerActivated`)	See OnActivated, OnBackgroundActivated, and other activation-handling methods.
(**Windows.UI.Xaml**) **Application.OnSearchActivated** method (commonly appears in C# as `App.OnSearchActivated`)	See OnActivated, OnBackgroundActivated, and other activation-handling methods.
(**Windows.UI.Xaml**) **Application.OnShareTargetActivated** method (commonly appears in C# as `App.OnShareTargetActivated`)	See OnActivated, OnBackgroundActivated, and other activation-handling methods.
(**Windows.UI.Xaml**) **Window.Current** property	Deprecated, and returns `null`. For an alternative, see Change Window.Current to App.Window.
(**Windows.UI.Xaml**) **Window.Dispatcher** property (commonly written in C# as `Window.Current.Dispatcher`)	(**Microsoft.UI.Xaml**) **Window.DispatcherQueue** property
(**Windows.UI.Xaml.Controls**) **ContentDialog** class	(**Microsoft.UI.Xaml.Controls**) **ContentDialog** is supported, but you must set its XamlRoot property. See ContentDialog, and Popup.
(**Windows.UI.Xaml.Controls**) **InkCanvas** class	Not supported in Windows App SDK 1.0.
(**Windows.UI.Xaml.Controls**) **MediaElement** class	Not supported in Windows App SDK 1.0.
(**Windows.UI.Xaml.Controls.Maps**) **MapControl** class	(**Microsoft.UI.Xaml.Controls**) **MapControl** class
(**Windows.UI.Xaml.Controls.Primitives**) **Popup** class	(**Microsoft.UI.Xaml.Controls.Primitives**) **Popup** is supported, but you must set its XamlRoot property. See ContentDialog, and Popup.
(**Windows.UI.Xaml.Media**) **AcrylicBrush.BackgroundSource** property	In the Windows App SDK, the (**Microsoft.UI.Xaml.Media**) **AcrylicBrush** always samples from the app content. See AcrylicBrush.BackgroundSource property.

UWP	Windows App SDK
C++/WinRT. `co_await winrt.resume_foreground(this->Dispatcher());`	See Migrate winrt.resume_foreground

Third-party libraries

This section documents various libraries that were supported in UWP, and need to be changed when using WinUI 3.

Expand table

UWP	Windows App SDK
OneDrive SDK	Microsoft Graph SDK

See Also

- Windows App SDK and supported Windows releases

Overview of feature area guides

Article • 03/19/2025

A collection of migration guidance topics, each focusing on a specific feature area.

Feature area guides in this section

Expand table

Topic	Description
User interface migration (including WinUI 3)	This topic shows how to migrate your user interface (UI) code, including migrating to WinUI 3.
Windowing functionality migration	This topic contains guidance related to window management, including migrating from UWP's **ApplicationView**/**CoreWindow** or **AppWindow** to the Window App SDK **Microsoft.UI.Windowing.AppWindow**.
Application lifecycle functionality migration	This topic contains migration guidance in the application lifecycle feature area.
Toast notifications functionality migration	This topic contains migration guidance in the toast notifications feature area.
Push notifications functionality migration	This topic contains migration guidance in the push notifications feature area.
DirectWrite to DWriteCore migration	DWriteCore is the Windows App SDK implementation of DirectWrite.
MRT to MRT Core migration	This topic contains guidance for migrating from UWP's Resource Management System (also known as MRT) to the Windows App SDK's MRT Core.
Threading functionality migration	This topic shows how to migrate your threading code.
Background task migration strategy	This topic contains considerations and strategies for approaching the migration process, and how to migrate your UWP background tasks to use the Windows App SDK background task APIs.

See Also

- Windows App SDK and supported Windows releases

User interface migration (including WinUI 3)

Article • 03/27/2023

This topic shows how to migrate your user interface (UI) code, including migrating to WinUI 3.

Summary of API and/or feature differences

The **Window.Current** property migrates to **App.Window**. And the **CoreDispatcher.RunAsync** method migrates to **DispatcherQueue.TryEnqueue**.

You need to set your window's *handle* (**HWND**) on a **MessageDialog**, and on **Picker**s.

To use **DataTransferManager** APIs, you need to associate them with your window.

For **ContentDialog** and **Popup**, you need to set their XamlRoot property.

You may need to refactor your Visual State Manager and **Page.Resources** XAML markup.

In the Windows App SDK, the **AcrylicBrush** always samples from the app content.

Change Windows.UI.Xaml.Window.Current to App.Window

This section applies if you're using the **Windows.UI.Xaml.Window.Current** property in your UWP app. That property isn't supported in the Windows App SDK, so this section describes how to port UWP code that uses **Window.Current**.

C#

```
// MainPage.xaml.cs in a UWP app
var width = Window.Current.Bounds.Width;
```

Your Windows App SDK app can add its own notion of a *current*, or *main* window by using a public static property on your **App** class.

C#

```
// App.xaml.cs in a Windows App SDK app
public partial class App : Application
```

```
{
    ...
    public static Window Window { get { return m_window; } }
    private static Window m_window;
}
```

Then, within the **App** class itself, you can change `Window.Current` to simply `window`. Outside of the **App** class, change `Window.Current` to `App.Window`, like this:

C#

```
// MainPage.xaml.cs in a UWP app
var width = App.Window.Bounds.Width;
```

MessageDialog, and Pickers

In your UWP app, if you use certain types from the **Windows.UI.Popups** or **Windows.Storage.Pickers** namespaces, then this section contains info to help you migrate that code. The code examples below use **MessageDialog**, but you can apply exactly the same techniques to displaying a picker (for example, a **FileOpenPicker**, a **FileSavePicker**, or a **FolderPicker**).

The steps that you have to follow in a desktop app are described in Display WinRT UI objects that depend on CoreWindow.

> **Note**
>
> For new apps, we recommend using the ContentDialog control instead of MessageDialog. For more info, see the ContentDialog, and Popup section below.

Here's some typical UWP code to display a **MessageDialog**.

C#

```
// In a UWP app
var showDialog = new Windows.UI.Popups.MessageDialog("Message here");
await showDialog.ShowAsync();
```

And here's the equivalent code in a Windows App SDK app.

C#

```
// MainWindow.xaml.cs in a WinUI 3 app
var showDialog = new Windows.UI.Popups.MessageDialog("Message here");
WinRT.Interop.InitializeWithWindow.Initialize(showDialog,
    WinRT.Interop.WindowNative.GetWindowHandle(this));
await showDialog.ShowAsync();
```

DataTransferManager

In your UWP app, if you call the **DataTransferManager.ShowShareUI** method, then this section contains info to help you migrate that code.

Here's some typical UWP code that calls **ShowShareUI**.

C#

```
// In a UWP app
var dataTransferManager =
Windows.ApplicationModel.DataTransfer.DataTransferManager.GetForCurrentView(
);

dataTransferManager.DataRequested += (sender, args) =>
{
    args.Request.Data.Properties.Title = "In a UWP app...";
    args.Request.Data.SetText("...display the user interface for sharing
content with another app.");
    args.Request.Data.RequestedOperation =
        Windows.ApplicationModel.DataTransfer.DataPackageOperation.Copy;
};

Windows.ApplicationModel.DataTransfer.DataTransferManager.ShowShareUI();
```

To use **DataTransferManager.ShowShareUI** in your Windows App SDK app, you need to associate the Share UI with your window. And that needs to be done manually. For more info, and code examples, see Display WinRT UI objects that depend on CoreWindow.

ContentDialog, and Popup

If in your UWP app you're using the **Windows.UI.Xaml.Controls.ContentDialog** or **Windows.UI.Xaml.Controls.Primitives.Popup** classes, then this section contains info to help you migrate that code. The code examples below use **ContentDialog**, but you can apply exactly the same techniques to displaying a **Popup** object.

Here's some typical UWP code to display a **ContentDialog**.

C#

```
// MainPage.xaml.cs in a UWP app
var unsupportedFilesDialog = new ContentDialog();
// Set Title, Content, etc.
await unsupportedFilesDialog.ShowAsync();
```

In your Windows App SDK app, you just need to also set the dialog's XamlRoot property. Here's how.

C#

```
// MainPage.xaml.cs in a Windows App SDK app
var unsupportedFilesDialog = new ContentDialog();
// Set Title, Content, etc.
unsupportedFilesDialog.XamlRoot = this.Content.XamlRoot;
await unsupportedFilesDialog.ShowAsync();
```

Do I need to implement page navigation?

In a UWP project, by default there will be navigation code in the methods of the **App** class, even if your app is simple enough that it has only one **Page**.

When you create a new Windows App SDK project in Visual Studio, the project template provides you with a **MainWindow** class (of type **Microsoft.UI.Xaml.Window**), but no **Page**. And the project template doesn't provide any navigation code.

For a Windows App SDK app that's simple enough (a single-page app), you might be able to simplify it. It might be that you needn't create pages or user controls in your Windows App SDK project—but instead copy the XAML markup and code-behind of that single page into **MainWindow**. However, there are some things that **MainWindow** doesn't support. **Window** isn't a DependencyObject, so capabilities such as **Resources** and **DataContext** don't exist on it. Neither do events such as **Load** and **Unload**. For more info, and workarounds, see Visual State Manager, and Page.Resources.

If on the other hand you want or need navigation between pages in your Windows App SDK app, then you can do so by migrating the **App.OnLaunched** and **App::OnNavigationFailed** methods from your UWP app. In **App.OnLaunched**, locate the navigation code (the code that creates *rootFrame*, and navigates to the first page of your app) and merge it right in between the two existing lines of code (the lines that create a window and then activate it). You'll also need to migrate the code you've copy-pasted. For a simple code example, see **Page class**.

Visual State Manager, and Page.Resources

Also see Do I need to implement page navigation?. If you *do* have a UWP app that's simple enough where you can copy your XAML markup and code-behind into **MainWindow**, then bear in mind these exceptions.

Your **MainWindow** class (of type **Microsoft.UI.Xaml.Window**) isn't a control, so it doesn't support Visual State Manager XAML markup and code-behind (see Tutorial: Create adaptive layouts). You have these two options, though:

- Add a **UserControl** item to the project, and migrate your markup and code-behind to that. Then place an instance of that user control in **MainWindow**.
- Add a **Page** item to the project, and migrate your markup and code-behind to that. Then add code to your **App** class to navigate to that **Page** on startup, as described in Do I need to implement page navigation?.

In addition, you won't be able to copy a `<Page.Resources>` element over to **MainWindow** and just rename it to `<Window.Resources>`. Instead, parent the **Resources** element under the root layout container (for example, a **Grid**) in the XAML markup for **MainWindow**. That will look like this:

XAML

```
<Window ...>
    <Grid>
        <Grid.Resources>...</Grid.Resources>
        ...
    </Grid>
</Window>
```

AcrylicBrush.BackgroundSource property

The **AcrylicBrush.BackgroundSource** property exists in UWP, but not in the Windows App SDK. In the Windows App SDK, the **AcrylicBrush** always samples from the app content.

So if you're accessing the **AcrylicBrush.BackgroundSource** property in the source code of your UWP app (whether that's in XAML markup or in imperative code), then remove that code when migrating your app to the Windows App SDK. Instead, use the DesktopAcrylicController class.

Related topics

- Windows App SDK and supported Windows releases

- Display WinRT UI objects that depend on CoreWindow

Keyboard API and events parity between UWP and WinUI 3

Article • 09/25/2024

The migration of apps from the Universal Windows Platform (UWP) to the Windows App SDK (specifically, WinUI 3) might require a few changes in the way input keyboard events are handled, primarily due to the app model differences. This topic describes commonly identified differences.

By using **UIElement**-specific WinUI 3 event handlers, and developing custom solutions for event routing, you can easily transition your apps to modern Windows applications.

Mapping to WinUI 3 APIs from UWP APIs

UWP APIs

In UWP, **CoreWindow** serves as the fundamental component attached directly to each application window, handling global input events across the application. In this topic, some of the commonly used events in conjunction with **CoreWindow** are discussed:

- **KeyDown**. Event raised by any key press when the app window has focus.
- **KeyUp**. Event raised when any key is released.
- **CharacterReceived**. Event raised when a new character is received by the input queue.
- **CoreDispatcher.AcceleratorKeyActivated**. Event raised when an accelerator key is activated (either pressed or held down).

C#

```
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        this.InitializeComponent();
        var coreWindow = CoreWindow.GetForCurrentThread();
        coreWindow.KeyDown += CoreWindow_KeyDown;
        coreWindow.KeyUp += CoreWindow_KeyUp;
        coreWindow.CharacterReceived += CoreWindow_CharacterReceived;
        coreWindow.Dispatcher.AcceleratorKeyActivated +=
Dispatcher_AcceleratorKeyActivated;
    }

    private void CoreWindow_KeyDown(CoreWindow sender, KeyEventArgs args)
```

```
    {
        System.Diagnostics.Debug.WriteLine($"KeyDown: {args.VirtualKey}");
    }
    private void CoreWindow_KeyUp(CoreWindow sender, KeyEventArgs args)
    {
        System.Diagnostics.Debug.WriteLine($"KeyUp: {args.VirtualKey}");
    }

    private void CoreWindow_CharacterReceived(CoreWindow sender,
CharacterReceivedEventArgs args)
    {
        System.Diagnostics.Debug.WriteLine($"CharacterReceived:
{(char)args.KeyCode}");
    }

    private void Dispatcher_AcceleratorKeyActivated(CoreDispatcher sender,
AcceleratorKeyEventArgs args)
    {
        if (args.EventType == CoreAcceleratorKeyEventType.KeyDown)
        {
            bool isCtrlPressed =
(Window.Current.CoreWindow.GetKeyState(VirtualKey.Control) &
CoreVirtualKeyStates.Down) ==
            CoreVirtualKeyStates.Down;
            if (isCtrlPressed && args.VirtualKey == VirtualKey.C)
            {
                System.Diagnostics.Debug.WriteLine("Ctrl+C Pressed");
            }
            else if (args.VirtualKey == VirtualKey.F1)
            {
                System.Diagnostics.Debug.WriteLine("F1 Key Pressed");
            }
        }
    }
}
```

WinUI 3 APIs (Windows App SDK)

WinUI 3 (Windows App SDK) has a different concept from **CoreWindow**. Instead, WinUI 3 provides an input event handling mechanism using **Microsoft.Ui.Xaml.UIElement** (UIElement). Each class that inherits from **UIElement** can implement various keyboard or pointer events (which are also present in UWP's **CoreWindow**) such as **KeyUp**, **KeyDown**, **CharacterReceived**, and others. This approach allows event handling at the level of a specific UI control.

For detailed info about event handling, see Keyboard events. You can associate keyboard events either XAML or in imperative source code. These examples demonstrate key event handling through code and accelerator key events in XAML.

XAML

```
<Window
    x:Class="SampleApp.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:SampleApp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <StackPanel Orientation="Horizontal" x:Name="stackPanelControl"
HorizontalAlignment="Center" VerticalAlignment="Center">
        <Button x:Name="myButton" Click="myButton_Click">Click Me</Button>
        <StackPanel.KeyboardAccelerators>
            <KeyboardAccelerator Key="C" Modifiers="Control"
Invoked="OnCtrlCInvoked"/>
        </StackPanel.KeyboardAccelerators>
    </StackPanel>
</Window>
```

C#

```
public sealed partial class MainWindow : Window
{
    public MainWindow()
    {
        this.InitializeComponent();
        stackPanelControl.KeyDown += OnKeyDownHandler;
        stackPanelControl.KeyUp += OnKeyUpHandler;
        stackPanelControl.CharacterReceived += OnCharacterReceivedHandler;
    }

    private void myButton_Click(object sender, RoutedEventArgs e)
    {
        myButton.Content = "Clicked";
    }

    private void OnKeyDownHandler(object sender, KeyRoutedEventArgs e)
    {
        System.Diagnostics.Debug.WriteLine($"KeyDown: {e.Key}");
    }

    private void OnKeyUpHandler(object sender, KeyRoutedEventArgs e)
    {
        System.Diagnostics.Debug.WriteLine($"KeyUp: {e.Key}");
    }

    private void OnCharacterReceivedHandler(UIElement sender,
CharacterReceivedRoutedEventArgs args)
    {
        System.Diagnostics.Debug.WriteLine($"CharacterReceived:
{(char)args.Character}");
```

```
    }

    private void OnCtrlCInvoked(KeyboardAccelerator sender,
KeyboardAcceleratorInvokedEventArgs args)
    {
        if (args.KeyboardAccelerator.Key == VirtualKey.C &&
args.KeyboardAccelerator.Modifiers == VirtualKeyModifiers.Control)
        {
            System.Diagnostics.Debug.WriteLine("Ctrl+C Pressed");
            args.Handled = true;
        }
    }
}
```

UWP sample apps for keyboard events

- UWP custom edit control sample app.
- UWP basic suspension sample app.

WinUI 3 sample apps for keyboard events and KeyboardAccelerator

- Input sample apps: KeyboardShortcutManager.xaml
- Input sample apps: KeyboardShortcutManager.xaml.cs

See also

UWP APIs

- CoreWindow class (Windows.UI.Core)
- KeyboardAccelerator class (Windows.UI.Xaml.Input)
- UIElement.CharacterReceived event (Windows.UI.Xaml)
- UIElement.KeyDown event (Windows.UI.Xaml)
- UIElement.KeyUp event (Windows.UI.Xaml)

WinUI 3 APIs (Windows App SDK)

- KeyboardAccelerator class (Microsoft.UI.Xaml.Input)
- UIElement.CharacterReceived event (Microsoft.UI.Xaml)
- UIElement.KeyDown event (Microsoft.UI.Xaml)
- UIElement.KeyUp event (Microsoft.UI.Xaml)

Windowing functionality migration

Article • 04/05/2023

This topic contains guidance related to window management, including migrating from UWP's ApplicationView/CoreWindow or AppWindow to the Window App SDK Microsoft.UI.Windowing.AppWindow.

Important APIs

- Microsoft.UI.Windowing.AppWindow
- Windows.UI.Core.CoreWindow.Dispatcher property
- Microsoft.UI.Window.DispatcherQueue property

Summary of API and/or feature differences

The Windows App SDK provides a Microsoft.UI.Windowing.AppWindow class that's based on the Win32 HWND model. That **AppWindow** class is the Windows App SDK's version of UWP's ApplicationView/CoreWindow and AppWindow.

To take advantage of the Windows App SDK windowing APIs means that you'll migrate your UWP code to use the Win32 model. For more info about the Windows App SDK **AppWindow**, see Manage app windows.

> **Tip**
>
> The **Manage app windows** topic contains a code example demonstrating how to retrieve an **AppWindow** from a WinUI 3 window. In your WinUI 3 app, use that code pattern so that you can call the **AppWindow** APIs mentioned in the rest of this topic.

Window types in UWP versus the Windows App SDK

In a UWP app, you can host window content using ApplicationView/CoreWindow, or AppWindow. The work involved in migrating that code to the Windows App SDK depends on which of those two windowing models your UWP app uses. If you're familiar with UWP's Windows.UI.WindowManagement.AppWindow, then you might see similarities between that and Microsoft.UI.Windowing.AppWindow.

UWP window types

- Windows.UI.ViewManagement.ApplicationView/Windows.UI.Core.CoreWindow.
- Windows.UI.WindowManagement.AppWindow. **AppWindow** consolidates the UI thread and the window that the app uses to display content. UWP apps that use **AppWindow** will have less work to do than **ApplicationView**/**CoreWindow** apps to migrate to the Windows App SDK **AppWindow**.

Windows App SDK window type

- Microsoft.UI.Windowing.AppWindow is the high-level abstraction of a system-managed container of the content of an app.

Keep in mind that the differences in windowing models between UWP and Win32 mean that there's not a direct 1:1 mapping between the UWP API surface and the Windows App SDK API surface. Even for class and member names that *do* carry over from UWP (reflected in this topic's mapping tables), behavior might also differ.

Splash screens

Unlike UWP apps, Win32 apps don't by default show a splash screen on launch. UWP apps relying on this feature for their launch experience might choose to implement a custom transition to their first app window.

Create, show, close, and destroy a window

The lifetime of a Microsoft.UI.Windowing.AppWindow is the same as for an HWND; meaning that the **AppWindow** object is available immediately after the window has been created, and is destroyed when the window is closed.

Create and show

AppWindow.Create creates an app window with the default configuration. Creating and showing a window is only necessary for scenarios where you're not working with a UI framework. If you're migrating your UWP app to a Win32-compatible UI framework, then you can still reach your **AppWindow** object from an already-created window by using the windowing interop methods.

Expand table

UWP ApplicationView/CoreWindow	UWP AppWindow	Windows App SDK
CoreApplication.CreateNewView or CoreWindow.GetForCurrentThread	AppWindow.TryCreateAsync	AppWindow.Create
CoreWindow.Activate	AppWindow.TryShowAsync	AppWindow.Show

Close

In UWP, ApplicationView.TryConsolidateAsync is the programmatic equivalent of the user initiating a close gesture. This concept of *consolidation* (in UWP's **ApplicationView/CoreWindow** windowing model) doesn't exist in Win32. Win32 doesn't require that windows exist in separate threads. Replicating the UWP's **ApplicationView/CoreWindow** windowing model would require the developer to create a new thread, and create a new window there. Under the Win32 model, the default system behavior is **Close** > **Hide** > **Destroy**.

Expand table

UWP ApplicationView/CoreWindow	UWP AppWindow	Windows App SDK
ApplicationView.TryConsolidateAsync	AppWindow.CloseAsync	AppWindow.Destroy

Basic window customization

As you migrate from UWP to the Windows App SDK, you can expect the same experience from your default **AppWindow**. But if needed, you can change the default **Microsoft.UI.Windowing.AppWindow** for customized windowing experiences. See Microsoft.UI.Windowing.AppWindow for more info on how to customize your windows.

Resizing a window

Expand table

UWP ApplicationView/CoreWindow	UWP AppWindow	Windows App SDK
ApplicationView.TryResizeView	AppWindow.RequestSize	AppWindow.Resize
CoreWindow.Bounds (commonly appears in C# as `CoreWindow.GetForCurrentThread.Bounds`)	AppWindowPlacement.Size	AppWindow.Size

Positioning a window

Expand table

UWP ApplicationView/CoreWindow	UWP AppWindow	Windows App SDK
Not possible	AppWindow.GetPlacement	AppWindow.Position
Not possible	Appwindow.RequestMoveXxx	AppWindow.Move

Window title

Expand table

UWP ApplicationView/CoreWindow	UWP AppWindow	Windows App SDK
ApplicationView.Title	AppWindow.Title	AppWindow.Title

Compact overlay, and full-screen

Apps that enter into compact overlay, or full-screen, should take advantage of the Windows App SDK AppWindowPresenter. If you're familiar with the UWP **AppWindow**, then you might already be familiar with the concept of presenters.

There isn't a 1:1 mapping of functionality and behavior from UWP app window presenters to Windows App SDK app window presenters. If you have a UWP **ApplicationView/CoreWindow** app, then you can still have compact overlay (picture-in-picture) or full-screen windowing experiences in your app, but the concept of presenters might be new to you. For more information on app window presenters, see Presenters. By default, an overlapped presenter is applied to an **AppWindow** at creation time. **CompactOverlay** and **FullScreen** are the only available presenters, aside from default.

Compact overlay

If you used UWP's **ApplicationViewMode** or **AppWindowPresentionKind** to present a compact overlay window, then you should use the compact overlay **AppWindowPresenterKind**. The Microsoft.UI.Windowing.CompactOverlayPresenter supports only three fixed window sizes at a 16:9 aspect ratio, and can't be resized by the user. Instead of **ApplicationViewMode.TryEnterViewModeAsync** or **AppWindowPresenterKind.RequestPresentation**, you should use AppWindow.SetPresenter to change the presentation of the **AppWindow**.

Expand table

UWP ApplicationView/CoreWindow	UWP AppWindow	Windows App SDK
ApplicationViewMode.CompactOverlay	AppWindowPresentationKind.CompactOverlay	AppWindowPresenterKind.CompactOverlay
ApplicationView.TryEnterViewModeAsync with	AppWindowPresenter.RequestPresentation with	AppWindow.SetPresenter with **AppWindowPresenterKind.CompactOverlay**

UWP ApplicationView/CoreWindow	UWP AppWindow	Windows App SDK
ApplicationViewMode.CompactOverlay	AppWindowPresenterKind.CompactOverlay	

Full-screen

If you used UWP's **ApplicationViewWindowingMode** or **AppWindowPresentionKind** classes to present a full-screen window, then you should use the full-screen **AppWindowPresenterKind**. The Windows App SDK supports only the most restrictive full-screen experience (that is, when **FullScreen** is **IsExclusive**). For **ApplicationView/CoreWindow**, you can use the **ApplicationView.ExitFullScreenMode** to take the app out of full-screen. When using presenters, you can take an app out of full-screen by setting the presenter back to overlapped/default by using **AppWindow.SetPresenter**.

Expand table

UWP ApplicationView/CoreWindow	UWP AppWindow	Windows App SDK
ApplicationViewWindowingMode.FullScreen	AppWindowPresentationKind.FullScreen	AppWindowPresenterKind.FullScreen
ApplicationView.TryEnterFullScreenMode	AppWindowPresenter.RequestPresentation with **AppWindowPresenterKind.FullScreen**	AppWindow.SetPresenter with **AppWindowPresenterKind.FullScreen**

For more details on how to work with app window presenters, see the Windowing gallery sample. It demonstrates how to toggle different app window presenter states.

Custom title bar

> **Note**
>
> Title bar customization APIs currently work on Windows 11 only. We recommend that you check AppWindowTitleBar.IsCustomizationSupported in your code before you call these APIs.

If your app uses a default title bar, then there's no additional title bar work needed when you migrate to Win32. If on the other hand your UWP app has a custom title bar, then it's possible to recreate the following scenarios in your Windows App SDK app.

1. Customize the system-drawn title bar
2. App-drawn custom title bar

Code that uses the UWP **ApplicationViewTitleBar**, **CoreApplicationViewTitleBar**, and **AppWindowTitleBar** classes migrates to using the Windows App SDK **Microsoft.UI.Windowing.AppWindowTitleBar** class.

Customize the system-drawn title bar

Here's a table of the color customization APIs.

> **Note**
>
> When AppWindowTitleBar.ExtendsContentIntoTitleBar is `true`, transparency is supported only for the following properties: **AppWindowTitleBar.ButtonBackgroundColor**, **AppWindowTitleBar.ButtonInactiveBackgroundColor**, **AppWindowTitleBar.ButtonPressedBackgroundColor**, **AppWindowTitleBar.ButtonHoverBackgroundColor** and **AppWindowTitleBar.BackgroundColor** (implicitly set).

Expand table

UWP ApplicationView/CoreWindow	UWP AppWindow	Windows App SDK
Properties of ApplicationViewTitleBar	Properties of AppWindowTitleBar	Properties of AppWindowTitleBar
BackgroundColor	BackgroundColor	BackgroundColor
ButtonBackgroundColor	ButtonBackgroundColor	ButtonBackgroundColor
ButtonForegroundColor	ButtonForegroundColor	ButtonForegroundColor
ButtonHoverBackgroundColor	ButtonHoverBackgroundColor	ButtonHoverBackgroundColor
ButtonHoverForegroundColor	ButtonHoverForegroundColor	ButtonHoverForegroundColor
ButtonInactiveBackgroundColor	ButtonInactiveBackgroundColor	ButtonInactiveBackgroundColor
ButtonInactiveForegroundColor	ButtonInactiveForegroundColor	ButtonInactiveForegroundColor
ButtonPressedBackgroundColor	ButtonPressedBackgroundColor	ButtonPressedBackgroundColor
ButtonPressedForegroundColor	ButtonPressedForegroundColor	ButtonPressedForegroundColor
ForegroundColor	ForegroundColor	ForegroundColor
InactiveBackgroundColor	InactiveBackgroundColor	InactiveBackgroundColor
InactiveForegroundColor	InactiveForegroundColor	InactiveForegroundColor

These Windows App SDK APIs are for further customization of the system-drawn title bar in addition to the AppWindow.Title API.

- AppWindow.SetIcon. Sets the title bar and taskbar icon picture using either an hIcon handle or a string path to a resource or to a file.
- AppWindowTitleBar.IconShowOptions. Gets or sets a value that specifies how the window icon is displayed in the title bar. Supports two values currently—**HideIconAndSystemMenu** and **ShowIconAndSystemMenu**.
- AppWindowTitleBar.ResetToDefault. Resets the current title bar back to the default settings for the window.

App-drawn custom title bar (full customization)

If you're migrating to using **AppWindowTitleBar**, then we recommend that you check AppWindowTitleBar.IsCustomizationSupported in your code before calling the following custom title bar APIs.

Expand table

UWP ApplicationView/CoreWindow	Windows App SDK AppWindow
CoreApplicationViewTitleBar.ExtendViewIntoTitleBar	AppWindowTitleBar.ExtendsContentIntoTitleBar The platform continues to draw the **Minimize**/**Maximize**/**Close** buttons for you, and reports the occlusion information.
CoreApplicationViewTitleBar.SystemOverlayLeftInset	AppWindowTitleBar.LeftInset
CoreApplicationViewTitleBar.SystemOverlayRightInset	AppWindowTitleBar.RightInset
CoreApplicationViewTitleBar.Height	AppWindowTitleBar.Height
AppWindowTitleBarOcclusion AppWindowTitleBar.GetTitleBarOcclusions	Represents the system-reserved regions of the app window that will occlude app content if **ExtendsContentIntoTitleBar** is true. The Windows App SDK **AppWindow** left and right inset information, coupled with title bar height, provide the same information. AppWindowTitleBar.LeftInset, AppWindowTitleBar.RightInset, AppWindowTitleBar.Height

These Windows App SDK APIs are for full title bar customization.

- AppWindowTitleBar.SetDragRectangles. Sets the drag regions for the window.
- AppWindowTitleBar.ResetToDefault. Resets the current title bar back to the default settings for the window.

These UWP **AppWindow** APIs have no direct 1:1 mapping to a Windows App SDK API.

- AppWindowTitleBarVisibility. Defines constants that specify the preferred visibility of an **AppWindowTitleBar**.
- AppWindowTitleBar.GetPreferredVisibility. Retrieves the preferred visibility mode for the title bar.
- AppWindowTitleBar.SetPreferredVisibility. Sets the preferred visibility mode for the title bar.

For more details on how to work with **AppWindowTitleBar**, see the Windowing gallery sample. It demonstrates how to create a custom color title bar and how to draw a custom title bar.

Event handling

If your UWP app uses the AppWindow.Changed event, then you can migrate that code to the Microsoft.UI.Windowing.AppWindow.Changed event.

Size changed event

When migrating size changed event-handling code, you should switch to using the Windows App SDK AppWindowChangedEventArgs.DidSizeChange property. The value is `true` if the size of the app window changed, otherwise it's `false`.

Expand table

UWP ApplicationView/CoreWindow	UWP AppWindow	Windows App SDK
CoreWindow.SizeChanged	AppWindowChangedEventArgs.DidSizeChange	AppWindowChangedEventArgs.DidSizeChange

MainPage and MainWindow

When you create a new UWP project in Visual Studio, the project template provides you with a **MainPage** class. For your app, you might have renamed that class (and/or added more pages and user controls). The project template also provides you with navigation code in the methods of the **App** class.

When you create a new Windows App SDK project in Visual Studio, the project template provides you with a **MainWindow** class (of type Microsoft.UI.Xaml.Window), but no **Page**. And the project template doesn't provide any navigation code.

However, you have the option to add pages and user controls to your Windows App SDK project. For example, you could add a new page item to the project (**WinUI** > **Blank Page (WinUI 3)**), and name it `MainPage.xaml`, or some other name. That would add to your project a new class of type Microsoft.UI.Xaml.Controls.Page. Then, for info about adding navigation code to the project, see Do I need to implement page navigation?.

For Windows App SDK apps that are simple enough, you needn't create pages or user controls, and you can copy your XAML markup and code-behind into **MainWindow**. But for info about exceptions to that workflow, see Visual State Manager, and Page.Resources.

Change CoreWindow.Dispatcher to Window.DispatcherQueue

Some use cases for UWP's **Windows.UI.Core.CoreWindow** class migrate to the Windows App SDK's **Microsoft.UI.Xaml.Window**.

For example, if you're using the **Windows.UI.Core.CoreWindow.Dispatcher** property in your UWP app, then the solution is *not* to migrate to the **Microsoft.UI.Xaml.Window.Dispatcher** property (which always returns null). Instead, migrate to the **Microsoft.UI.Xaml.Window.DispatcherQueue** property, which returns a **Microsoft.UI.Dispatching.DispatcherQueue**.

For more info, and code examples, see Change Windows.UI.Core.CoreDispatcher to Microsoft.UI.Dispatching.DispatcherQueue.

Related topics

- Windows App SDK and supported Windows releases
- Migrate your threading functionality

Application lifecycle functionality migration

Article • 09/06/2023

This topic contains migration guidance in the application lifecycle area.

Important APIs

- **AppInstance** class
- **Application.OnLaunched** method
- **AppInstance.GetActivatedEventArgs** method
- **ExtendedActivationKind** enum

Summary of API and/or feature differences

Universal Windows Platform (UWP) apps are single-instanced by default; Windows App SDK (WinUI 3) apps are multi-instanced by default.

A UWP app has **App** methods such as **OnFileActivated**, **OnSearchActivated**, **OnActivated**, and **OnBackgroundActivated** that implicitly tell you how the app was activated; In a Windows App SDK app, in **App.OnLaunched** (or in any method), call (**AppInstance.GetActivatedEventArgs**) to retrieve the activated event args, and check them to determine how the app was activated.

Also see the *Background tasks* row in the table in the What's supported when migrating from UWP to WinUI 3 topic.

Single-instanced apps

Universal Windows Platform (UWP) apps are single-instanced by default (you can opt in to support multiple instances—see Create a multi-instance UWP app).

So the way a single-instanced UWP app behaves is that the second (and subsequent) time you launch, your current instance is activated. Let's say for example that in your UWP app you've implemented the file type association feature. If from File Explorer you open a file (of the type for which the app has registered a file type association)—and your app is already running—then that already-running instance is activated.

Windows App SDK (WinUI 3) apps, on the other hand, are multi-instanced by default. So, by default, the second (and subsequent) time you launch a Windows App SDK (WinUI 3) app, a new instance of the app is launched. If for example a Windows App SDK (WinUI 3) app implements file type association, and from File Explorer you open a file (of the right type) while that app is already running, then by default a new instance of the app is launched.

If you want your Windows App SDK (WinUI 3) app to be single-instanced like your UWP app is, then you can override the default behavior described above. You'll use **AppInstance.FindOrRegisterForKey** and **AppInstance.IsCurrent** to determine whether the current instance is the main instance. If it isn't, then you'll call **AppInstance.RedirectActivationToAsync** to redirect activation to the already-running main instance, and then exit from the current instance (without creating nor activating its main window).

For more info, see App instancing with the app lifecycle API.

> **Important**
>
> The code shown below works as expected provided that you target the *x64* architecture. That applies to both C# and C++/WinRT.

Single-instancing in Main or wWinMain

It's best to check for the need to redirect activation as early as possible in your app's execution. For that reason, we recommend that you perform your single-instancing logic in your app's **Main** (or **wWinMain** for C++/WinRT) function. This section shows you how.

Ordinarily, your app's **Main** function is auto-generated by the build system, and placed into a hidden file. So the first step is to configure your project not to auto-generate that function. To do that, you define the symbol *DISABLE_XAML_GENERATED_MAIN* in project **Properties**.

Instructions for C#

Go to **Properties** > (select **All Configurations** and **All Platforms**) > **Build** > **Conditional compilation symbols**, and paste in the symbol *DISABLE_XAML_GENERATED_MAIN*.

Because we've just prevented the project from auto-generating a **Main** function, the project won't build at the moment. So the second and last step is to implement our own

version of that function in a source code file.

Add a new project item of type **Class** to the project, and name it *Program.cs*. Inside `Program.cs`, replace the code `class Program {}` with your own implementation. For an example of the code to use, see `Program.cs` in the AppLifecycle sample.

Instructions for C++/WinRT

Go to **Properties** > (select **All Configurations** and **All Platforms**) > **Configuration Properties** > **C/C++** > **Preprocessor** > **Preprocessor Definitions**, **Edit** the value, and add the symbol *DISABLE_XAML_GENERATED_MAIN*.

Because we've just prevented the project from auto-generating a **wWinMain** function, the project won't build at the moment. So the second and last step is to implement our own version of that function in a source code file.

Add a reference to the Microsoft.Windows.ImplementationLibrary NuGet package, and update your project's `pch.h` and `App.xaml.cpp` source code files. For an example of the code to use, see the AppLifecycle sample. Be sure to change the namespace in `winrt::CppWinUiDesktopInstancing::implementation::App` to suit your particular project).

To resolve "error C2872: 'Microsoft': ambiguous symbol", change `using namespace Microsoft::UI::Xaml;` to `using namespace winrt::Microsoft::UI::Xaml;`. And make any additional similar changes to `using` directives.

Single-instancing in Application.OnLaunched

An alternative to using **Main** or **wWinMain** is to perform your single-instancing logic in the **Application.OnLaunched** method of your **App** class.

> **Important**
>
> Doing this work in **Application.OnLaunched** can simplify your app. However, a lot depends on what else your app is doing. If you're going to end up redirecting, and then terminating the current instance, then you'll want to avoid doing any throwaway work (or even work that needs explicitly undoing). In cases like that, **Application.OnLaunched** might be too late, and you might prefer to do the work in your app's **Main** or **wWinMain** function.

C#

```
// App.xaml.cs in a Windows App SDK (WinUI 3) app
...
protected override async void
OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
{
    // If this is the first instance launched, then register it as the
"main" instance.
    // If this isn't the first instance launched, then "main" will already
be registered,
    // so retrieve it.
    var mainInstance =
Microsoft.Windows.AppLifecycle.AppInstance.FindOrRegisterForKey("main");

    // If the instance that's executing the OnLaunched handler right now
    // isn't the "main" instance.
    if (!mainInstance.IsCurrent)
    {
        // Redirect the activation (and args) to the "main" instance, and
exit.
        var activatedEventArgs =

Microsoft.Windows.AppLifecycle.AppInstance.GetCurrent().GetActivatedEventArg
s();
        await mainInstance.RedirectActivationToAsync(activatedEventArgs);
        System.Diagnostics.Process.GetCurrentProcess().Kill();
        return;
    }

    m_window = new MainWindow();
    m_window.Activate();
}
```

Alternatively, you can call **AppInstance.GetInstances** to retrieve a collection of running **AppInstance** objects. If the number of elements in that collection is greater than 1, then your main instance is already running, and you should redirect to that.

File type association

In a Windows App SDK project, to specify the extension point for a file type association, you make the same settings in your `Package.appxmanifest` file as you would for a UWP project. Here are those settings.

Open `Package.appxmanifest`. In **Declarations**, choose **File Type Associations**, and click **Add**. Set the following properties.

Display name: MyFile **Name**: myfile **File type**: .myf

To register the file type association, build the app, launch it, and close it.

The difference comes in the imperative code. In a UWP app, you implement **App::OnFileActivated** in order to handle file activation. But in a Windows App SDK app, you write code in **App::OnLaunched** to check the extended activation kind (**ExtendedActivationKind**) of the activated event args (**AppInstance.GetActivatedEventArgs**), and see whether the activation is a file activation.

> ⓘ **Note**
>
> Don't use the Microsoft.UI.Xaml.LaunchActivatedEventArgs object passed to **App::OnLaunched** to determine the activation kind, because it reports "Launch" unconditionally.

If your app has navigation, then you'll already have navigation code in **App::OnLaunched**, and you might want to re-use that logic. For more info, see Do I need to implement page navigation?.

C#

```
// App.xaml.cs in a Windows App SDK app
...
using Microsoft.Windows.AppLifecycle;
...
protected override void
OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
{
    var activatedEventArgs =
Microsoft.Windows.AppLifecycle.AppInstance.GetCurrent().GetActivatedEventArg
s();
    if (activatedEventArgs.Kind ==
Microsoft.Windows.AppLifecycle.ExtendedActivationKind.File)
    {
        ...
    }
    ...
}
```

OnActivated, OnBackgroundActivated, and other activation-handling methods

In a UWP app, to override the various means by which your app can be activated, you can override corresponding methods on your **App** class, such as **OnFileActivated**, **OnSearchActivated**, or the more general **OnActivated**.

In a Windows App SDK app, in **App.OnLaunched** (or in fact at any time) you can call (**AppInstance.GetActivatedEventArgs**) to retrieve the activated event args, and check them to determine how the app was activated.

See the File type association section above for more details and a code example. You can apply the same technique for any activation kind specified by the **ExtendedActivationKind** enum.

Related topics

- Windows App SDK and supported Windows releases
- App instancing with the app lifecycle API
- Do I need to implement page navigation?

App notifications from UWP to WinUI 3 migration

Article • 11/17/2022

The only difference when migrating app notification code from UWP to WinUI 3 is in handling the activation of notifications. Sending and managing app notifications remains exactly the same.

> **Note**
>
> The term "toast notification" is being replaced with "app notification". These terms both refer to the same feature of Windows, but over time we will phase out the use of "toast notification" in the documentation.

> **Note**
>
> **Some information relates to pre-released product, which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.**

Activation differences

Windows App SDK

Expand table

Category	UWP	WinUI 3
Foreground activation entry point	`OnActivated` method inside `App.xaml.cs` is called	`OnLaunched` method inside `App.xaml.cs` is called.
Background activation entry point	Handled separately as a background task	Same as foreground activation. `OnLaunched` method inside `App.xaml.cs` is called. Use GetActivatedEventArgs to determine if the app should fully launch or just handle task and quit.
Window activation	Your window is automatically brought to	You must bring your window to the foreground if desired

Category	UWP	WinUI 3
	foreground when foreground activation occurs	

Migration for C# apps

Step 1: Install NuGet library

Windows App SDK

For a WinUI 3 app, you handle activation for notifications by using the AppNotificationManager class. This class is provided by the Microsoft.WindowsAppSDK Nuget package, which is included by default in the WinUI 3 Visual Studio project templates.

Step 2: Update your manifest

In your **Package.appxmanifest**, add:

1. Declaration for **xmlns:com**
2. Declaration for **xmlns:desktop**
3. In the **IgnorableNamespaces** attribute, **com** and **desktop**
4. **desktop:Extension** for **windows.toastNotificationActivation** to declare your toast activator CLSID (using a new GUID of your choice).
5. MSIX only: **com:Extension** for the COM activator using the GUID from step #4. Be sure to include the `Arguments="----AppNotificationActivated:"` so that you know your launch was from a notification

Windows App SDK

XML

```
<!--Add these namespaces-->
<Package
  ...
  xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"

xmlns:desktop="http://schemas.microsoft.com/appx/manifest/desktop/windows10"
```

```
  IgnorableNamespaces="... com desktop">
  ...
  <Applications>
    <Application>
      ...
      <Extensions>

        <!--Specify which CLSID to activate when app notification
clicked-->
        <desktop:Extension
Category="windows.toastNotificationActivation">
          <desktop:ToastNotificationActivation
ToastActivatorCLSID="replaced-with-your-guid-C173E6ADF0C3" />
        </desktop:Extension>

        <!--Register COM CLSID LocalServer32 registry key-->
        <com:Extension Category="windows.comServer">
          <com:ComServer>
            <com:ExeServer Executable="YourProject.exe" Arguments="----
AppNotificationActivated:" DisplayName="App notification activator">
              <com:Class Id="replaced-with-your-guid-C173E6ADF0C3"
DisplayName="App notification activator"/>
            </com:ExeServer>
          </com:ComServer>
        </com:Extension>

      </Extensions>
    </Application>
  </Applications>
 </Package>
```

Step 3: Handle activation

Windows App SDK

In your app's startup code (typically App.xaml.cs), update your code using the following steps:

1. In **OnLaunched**, get the default instance of the AppNotificationManager class.
2. Register for the AppNotificationManager.NotificationInvoked event.
3. Call Microsoft.Windows.AppNotifications.AppNotificationManager.Register to register your app to receive notification events. It is important that your call this method after registering the **NotificationInvoked** handler.
4. Refactor your window launch/activation code into a dedicated `LaunchAndBringToForegroundIfNeeded` helper method, so you can call it from multiple places.

5. Create a `HandleNotification` helper method, so that it can be called from multiple places.
6. Call AppInstance.GetActivatedEventArgs and check the AppActivationArguments.Kind property of the returned object for the value ExtendedActivationKind.AppNotification.
7. If the activation kind is not **AppNotification** call the **LaunchAndBringToForegroundIfNeeded** helper method.
8. If the activation kind is **AppNotification** cast the AppActivationArguments.Data property to an AppNotificationActivatedEventArgs and pass it to the `HandleNotification` helper method.
9. In your **ApplicationManager.NotificationInvoked** handler, call the `HandleNotification` helper method.
10. Within your `HandleNotification` helper method, be sure to dispatch to the App or Window dispatcher before executing any UI-related code like showing a window or updating UI
11. **Migrate** your old UWP `OnActivated` code that handled app notification activation to your new `HandleNotification` helper method.

Migrated App.xaml.cs

cs

```
protected override void
OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
{
    m_window = new MainWindow();

    // To ensure all Notification handling happens in this process
instance, register for
    // NotificationInvoked before calling Register(). Without this a new
process will
    // be launched to handle the notification.
    AppNotificationManager notificationManager =
AppNotificationManager.Default;
    notificationManager.NotificationInvoked +=
NotificationManager_NotificationInvoked;
    notificationManager.Register();

    var activatedArgs =
Microsoft.Windows.AppLifecycle.AppInstance.GetCurrent().GetActivatedEven
tArgs();
    var activationKind = activatedArgs.Kind;
    if (activationKind != ExtendedActivationKind.AppNotification)
    {
```

```
        LaunchAndBringToForegroundIfNeeded();
    } else
    {

HandleNotification((AppNotificationActivatedEventArgs)activatedArgs.Data
);
    }

}

private void LaunchAndBringToForegroundIfNeeded()
{
    if (m_window == null)
    {
        m_window = new MainWindow();
        m_window.Activate();

        // Additionally we show using our helper, since if activated via
a app notification, it doesn't
        // activate the window correctly
        WindowHelper.ShowWindow(m_window);
    }
    else
    {
        WindowHelper.ShowWindow(m_window);
    }
}

private void
NotificationManager_NotificationInvoked(AppNotificationManager sender,
AppNotificationActivatedEventArgs args)
{
    HandleNotification(args);
}

private void HandleNotification(AppNotificationActivatedEventArgs args)
{
  // Use the dispatcher from the window if present, otherwise the app
dispatcher
  var dispatcherQueue = m_window?.DispatcherQueue ??
DispatcherQueue.GetForCurrentThread();


  dispatcherQueue.TryEnqueue(async delegate
  {

      switch (args.Arguments["action"])
      {
          // Send a background message
          case "sendMessage":
              string message = args.UserInput["textBox"].ToString();
              // TODO: Send it

              // If the UI app isn't open
              if (m_window == null)
```

```
                {
                    // Close since we're done
                    Process.GetCurrentProcess().Kill();
                }

                break;

            // View a message
            case "viewMessage":

                // Launch/bring window to foreground
                LaunchAndBringToForegroundIfNeeded();

                // TODO: Open the message
                break;
        }
    });
}

private static class WindowHelper
{
    [DllImport("user32.dll")]
    private static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);

    [DllImport("user32.dll")]
    [return: MarshalAs(UnmanagedType.Bool)]
    private static extern bool SetForegroundWindow(IntPtr hWnd);

    public static void ShowWindow(Window window)
    {
        // Bring the window to the foreground... first get the window
handle...
        var hwnd = WinRT.Interop.WindowNative.GetWindowHandle(window);

        // Restore window if minimized... requires DLL import above
        ShowWindow(hwnd, 0x00000009);

        // And call SetForegroundWindow... requires DLL import above
        SetForegroundWindow(hwnd);
    }
}
```

Building app notification content

Windows App SDK

With Windows App SDK, you can still create app notification content using raw xml, but you can also create app notification content using the new **AppNotificationsBuilder** API which replaces the **ToastContentBuilder** class

provided by the Windows Community Toolkit. Send the app notification by calling AppNotificationManager.Show. Mixing Windows Community Toolkit and App SDK APIs is not recommended.

C#

```
using Microsoft.Windows.AppNotifications;
using Microsoft.Windows.AppNotifications.Builder;

...

var builder = new AppNotificationBuilder()
    .AddText("Send a message.")
    .AddTextBox("textBox")
    .AddButton(new AppNotificationButton("Send")
        .AddArgument("action", "sendMessage"));

var notificationManager = AppNotificationManager.Default;
notificationManager.Show(builder.BuildNotification());
```

Related topics

- Windows App SDK and supported Windows releases
- Send a local toast notification from C# apps
- Send a local toast notification from Win32 C++ WRL apps

Push notifications functionality migration

Article • 06/27/2024

This topic contains migration guidance in the push notifications feature area.

> **Important**
>
> Only raw push notifications and app push notifications are currently supported. Badge push notifications and tile push notifications are not supported.

Summary of API and/or feature differences

Push notifications can be broken down into these four separate stages.

Expand table

Stage	UWP	Windows App SDK
Identity	Partner Center (MSA)	Azure App Registration (AAD)
Channel request	Asynchronous	Asynchronous Azure App Registration Id Retry logic built in (up to 5 retries)
Activation	In-process, PushTrigger*, COM activation*	In-process, COM activation, ShellExecute
Sending push notifications	Uses login.live.com endpoint to receive an access token	Uses the `https://login.microsoftonline.com/{tenantID}/oauth2/token` endpoint for token request

* Supported for Windows 10, version 2004 (10.0; Build 19041), and later.

Identity setup

In the Windows App SDK, the push notifications feature uses identity from Azure App Registration (AAD), which removes the requirement of having a Package Family Name

(PFN) from Partner Center in order to use push notifications.

- For a **UWP app**, you sign up and register the application in Windows Store Partner Center.
- For a **Windows App SDK app**, you create an Azure account, and create an Azure App Registration (AAD).

Channel requests

Channel request are handled asynchronously, and require the Azure AppID GUID and Azure tenantID (you receive the Azure AppID and tenant ID from an AAD app registration). You use the Azure AppID for your identity in place of the Package Family Name (PFN) that a UWP app uses. In case the request runs into a retryable error, the notification platform will attempt multiple retries.

A Windows App SDK app can check the status of a channel request.

Activation

See the Windows App SDK registration and activation steps at Configure your app to receive push notifications.

Sending push notifications

A Windows App SDK app must request the access token from the AAD endpoint, rather than from the MSA endpoint.

Access Token Request

For **a UWP app**:

```HTTP
POST /accesstoken.srf HTTP/1.1
Host: login.live.com
Content-Type: application/x-www-form-urlencoded
Cookie: MSCC=73.140.231.96-US
Content-Length: 112

grant_type=client_credentials&client_id=<AppID_Here>&client_secret=
<Client_Secret_Here>&scope=notify.windows.com
```

For a **Windows App SDK app** (AAD access token request):

HTTP

```
POST /{tenantID}/oauth2/v2.0/token Http/1.1
Host: login.microsoftonline.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 160

grant_type=client_credentials&client_id=
<Azure_App_Registration_AppId_Here>&client_secret=
<Azure_App_Registration_Secret_Here>&resource=https://wns.windows.com/
```

HTTP Post to WNS

When it comes to sending a HTTP POST request to WNS, there are no changes from UWP. The access token is still passed in the authorization header.

HTTP

```
POST /?token=[ChannelURI] HTTP/1.1
Host: dm3p.notify.windows.com
Content-Type: application/octet-stream
X-WNS-Type: wns/raw
Authorization: Bearer [your access token]
Content-Length: 46

{ Sync: "Hello from the Contoso App Service" }
```

See Also

- Windows App SDK and supported Windows releases

DirectWrite to DWriteCore migration

Article • 06/24/2024

DWriteCore is the Windows App SDK implementation of DirectWrite. For more info, see DWriteCore overview.

Summary of API and/or feature differences

Nearly all DirectWrite APIs remain unchanged in DWriteCore. There are a few differences, as described in APIs that are new, or different, for DWriteCore.

As you'll see in that topic, DWriteCore has a more locked-down factory type, and has the ability to retrieve pixel data without using GDI.

Migration guidance

The only change necessary when moving from DirectWrite to DWriteCore is to include the `dwrite_core.h` header file. For more info, and code examples, see Programming with DWriteCore.

> ⚠ **Warning**
>
> DWriteCore does not currently support hardware-accelerated text rendering with Direct2D (D2D). It supports software text rendering only. This prevents apps that require D2D support from adopting DWriteCore at this time.

The DWriteCoreGallery sample app

Also see the DWriteCoreGallery sample app project, which demonstrates the DWriteCore API surface.

Related topics

- Windows App SDK and supported Windows releases
- DWriteCore overview
- DWriteCoreGallery sample app
- DirectWrite
- Windows App SDK

MRT to MRT Core migration

Article • 03/22/2022

This topic contains guidance for migrating from UWP's Resource Management System (also known as MRT) to the Windows App SDK's MRT Core.

MRT Core is a streamlined version of MRT. For more info, see Manage resources with MRT Core.

Summary of API and/or feature differences

For ease of migration, MRT Core APIs are very similar to MRT APIs. For API reference documentation, see the Microsoft.Windows.ApplicationModel.Resources namespace.

> **Note**
>
> Not all of the MRT APIs exist in MRT Core. But all of the APIs necessary for the basic functionality of MRT are included.

Change namespace

In UWP, the MRT APIs are in the **Windows.ApplicationModel.Resources.Core** namespace. In the Windows App SDK, the MRT Core APIs are in the **Microsoft.Windows.ApplicationModel.Resources** namespace. So you'll need to change that namespace name in your source code (add `Microsoft.` at the start, and remove the `.Core` at the end).

> **Note**
>
> In Windows App SDK 1.0 Preview 1 and later releases, MRT Core APIs are in the Microsoft.Windows.ApplicationModel.Resources namespace. In releases earlier than that, they are in the **Microsoft.ApplicationModel.Resources** namespace.

ResourceManager class

This section applies if you're using the **Windows.ApplicationModel.Resources.Core.ResourceManager.Current** property in your UWP app.

C#

```
// In a UWP app
using Windows.ApplicationModel.Resources.Core;
...
var currentResourceManager = ResourceManager.Current;
```

C++/WinRT

```
// In a UWP app
#include <winrt/Windows.ApplicationModel.Resources.Core.h>
using namespace winrt::Windows::ApplicationModel::Resources::Core;
...
auto currentResourceManager{ ResourceManager::Current() };
```

Instead, in your Windows App SDK app, create a new Microsoft.Windows.ApplicationModel.Resources.ResourceManager.

C#

```
// In a Windows App SDK app
using Microsoft.Windows.ApplicationModel.Resources;
...
var currentResourceManager = new ResourceManager();
```

C++/WinRT

```
// In a Windows App SDK app
#include <winrt/Microsoft.Windows.ApplicationModel.Resources.h>
using namespace winrt::Microsoft::Windows::ApplicationModel::Resources;
...
ResourceManager currentResourceManager;
```

ResourceContext.GetForCurrentView, and ResourceContext.GetForViewIndependentUse

UWP's MRT ResourceContext class distinguishes between a **ResourceContext** for the current view, and one for view-independent use.

For the Windows App SDK's MRT Core ResourceContext class, your app needs to determine the correct context (resource qualifier values), and the concepts of current view and view-independent use no longer apply.

- So if you're using the **ResourceContext.GetForCurrentView** API or the **ResourceContext.GetForViewIndependentUse** API, then use ResourceManager.CreateResourceContext instead.

Resource qualifier values

In UWP's MRT, the resource context qualifier values are determined for the app. While in MRT Core, only the language value is populated. Your app needs to determine other values for itself. Here's an example, where it's assumed your XAML view contains an element named *layoutRoot*.

C#

```
// In a Windows App SDK app
using Microsoft.Windows.ApplicationModel.Resources;
...
var currentResourceManager = new ResourceManager();
var resourceContext = currentResourceManager.CreateResourceContext();
int scaleFactor = Convert.ToInt32(layoutRoot.XamlRoot.RasterizationScale *
100);
resourceContext.QualifierValues[KnownResourceQualifierName.Scale] =
scaleFactor.ToString();
string s =
resourceContext.QualifierValues[KnownResourceQualifierName.Scale];
```

C++/WinRT

```
// In a Windows App SDK app
#include <winrt/Microsoft.Windows.ApplicationModel.Resources.h>
using namespace winrt::Microsoft::Windows::ApplicationModel::Resources;
...
ResourceManager currentResourceManager;
auto resourceContext{ currentResourceManager.CreateResourceContext() };
auto scaleFactor{ layoutRoot().XamlRoot().RasterizationScale() * 100 };
resourceContext.QualifierValues().Insert(L"Scale",
std::to_wstring((int)scaleFactor));
auto s{ resourceContext.QualifierValues().Lookup(L"Scale") };
```

Resource qualifier value change

UWP's MRT provides the ResourceQualifierObservableMap.MapChanged event. And this section applies if your UWP app is handling that event in order to listen to qualifier value changes.

MRT Core doesn't provide any notification mechanics around environment changes. So your Windows App SDK app needs to detect such changes on its own if you wish to update resources based on environment changes.

An MRT Core sample app

Also see the Load resources using MRT Core sample app project, which demonstrates how to use the MRT Core API surface.

Related Topics

- Windows App SDK and supported Windows releases
- Manage resources with MRT Core
- MRT Core images
- Resource Management System
- Windows App SDK

Background task migration strategy

Article • 03/19/2025

Background tasks are used heavily in UWP apps for performing jobs when a particular trigger is invoked on the machine. As these don't require any service to be running listening for the triggers, it's very power efficient. So, while migrating UWP apps to WinUI 3 and other desktop apps that use Windows App SDK, developers may require support for implementing background tasks on the platform.

Background tasks offered in the UWP app model can either be out-of-proc or in-proc. This article describes the migration strategy for each of these types when moving to BackgroundTaskBuilder APIs in Windows App SDK.

Out-of-proc background tasks

In the case of out-of-proc background tasks in UWP, background tasks will be written as a Windows Runtime (WinRT) component, and the component is set as the TaskEntryPoint on BackgroundTaskBuilder during registration. While migrating to Windows App SDK, developers can keep this as-is and package the WinRT component together with the desktop project. In this case, the broker infrastructure will be launching `backgroundtaskhost` process when the trigger is invoked, and the WinRT component background task would be executed in the process. In this approach, the background task would be running in a Low Integrity Level (IL) process (`backgroundtaskhost.exe`) while the main desktop project would be running in a Medium IL process.

If the application needs to to run a background task in a medium IL process itself, the full trust COM component needs to be used for background tasks. In that scenario, developers must use the Windows App SDK BackgroundTaskBuilder to register the full trust COM component. Note that the BackgroundTaskBuilder API from the **Windows.ApplicationModel.Background** namespace will throw exceptions while registering of some of the triggers.

A full WinUI 3 background task registration sample can be found on GitHub.

More details on the implementation are available here. The only required change would be to replace the WinRT BackgroundTaskBuilder API with the Windows App SDK API Microsoft.Windows.ApplicationModel.Background.BackgroundTaskBuilder.

In-proc background tasks

For in-proc background tasks in UWP, background task routines are implemented in the OnBackgroundActivated callback which runs as part of the foreground process. This won't be possible in a WinUI 3 application as the **OnBackgroundActivated** callbacks aren't available. The application needs to move the background task implementations to full trust COM tasks as described above and define the COM server in the package manifest to handle the COM activation of the task. When the trigger occurs, COM activation will happen on the corresponding COM Coclass registered for the trigger.

A full WinUI 3 background task registration sample can be found on GitHub.

More details on the implementation are available here. The only change would be to replace the WinRT BackgroundTaskBuilder API with the Windows App SDK APIs in Microsoft.Windows.ApplicationModel.Background.BackgroundTaskBuilder.

Windows App SDK BackgroundTaskBuilder API

This Windows App SDK **BackgroundTaskBuilder** API is created to support the full trust COM background task implementations in WinUI 3 and other desktop applications that use Windows App SDK, as the WinRT API throws exceptions during registration except for a few triggers.

The following code shows how to register background task using Windows App SDK BackgroundTaskBuilder APIs:

C++

```
//Using Windows App SDK API for BackgroundTaskBuilder
winrt::Microsoft::Windows::ApplicationModel::Background::BackgroundTaskBuild
er builder;
SystemTrigger trigger = SystemTrigger(SystemTriggerType::TimeZoneChange,
false);
auto backgroundTrigger = trigger.as<IBackgroundTrigger>();
builder.SetTrigger(backgroundTrigger);
builder.AddCondition(SystemCondition(SystemConditionType::InternetAvailable)
);
builder.SetTaskEntryPointClsid(classGuid);
builder.Register();
```

The following is the equivalent C# code:

C#

```
// Using Windows App SDK API for BackgroundTaskBuilder
var builder = new
Microsoft.Windows.ApplicationModel.Background.BackgroundTaskBuilder();
var trigger = new SystemTrigger(SystemTriggerType.TimeZoneChange, false);
```

```
builder.SetTrigger(trigger);
builder.AddCondition(new
SystemCondition(SystemConditionType.InternetAvailable));
builder.SetTaskEntryPointClsid(classGuid);
builder.Register();
```

To use this API, add the tag below to the project file to enable Windows App SDK background tasks:

XML

```
<WindowsAppSDKBackgroundTask>true</WindowsAppSDKBackgroundTask>
```

Also, in the manifest file, the **EntryPoint** for **BackgroundTask** is set to `Microsoft.Windows.ApplicationModel.Background.UniversalBGTask.Task`:

XML

```
<Extension Category="windows.backgroundTasks"
EntryPoint="Microsoft.Windows.ApplicationModel.Background.UniversalBGTask.Ta
sk">
    <BackgroundTasks>
        <Task Type="general"/>
    </BackgroundTasks>
</Extension>
```

For C# applications, an **ActivatableClass** registration should also be added to the manifest file:

XML

```
<Extension Category="windows.activatableClass.inProcessServer">
    <InProcessServer>

<Path>Microsoft.Windows.ApplicationModel.Background.UniversalBGTask.dll</Pat
h>
        <ActivatableClass
ActivatableClassId="Microsoft.Windows.ApplicationModel.Background.UniversalB
GTask.Task" ThreadingModel="both"/>
    </InProcessServer>
</Extension>
```

Leveraging TaskScheduler for background task migration

Task Scheduler helps desktop apps achieve the same functionality that is provided by BackgroundTaskBuilder in UWP apps. More details on implementations using **TaskScheduler** are available here.

ApplicationTrigger use in Windows App SDK applications

ApplicationTrigger is supported in UWP applications due to the lifetime management scenario where the application process can be suspended. This scenario does not occur for WinUI and other Windows App SDK desktop applications, so this trigger is not supported in WinUI applications. All logic related to **ApplicationTrigger** will need to be rewritten to execute by launching another process or by running in a thread pool thread. For more information, see CreateThread and CreateProcess.

Related content

- Using background tasks in Windows apps
- BackgroundTaskBuilder

Threading functionality migration

Article • 03/29/2023

This topic describes how to migrate the threading code in a Universal Windows Platform (UWP) application to the Windows App SDK.

Summary of API and/or feature differences

UWP's threading model is a variation of the single-threaded apartment (STA) model called Application STA (ASTA), which blocks reentrancy and helps avoid various reentrancy bugs and deadlocks. An ASTA thread is also known as a UI thread.

The Windows App SDK uses a standard STA threading model, which doesn't provide the same reentrancy safeguards.

The **CoreDispatcher** type migrates to **DispatcherQueue**. And the **CoreDispatcher.RunAsync** method migrates to **DispatcherQueue.TryEnqueue**.

C++/WinRT. If you're using **winrt::resume_foreground** with **CoreDispatcher**, then migrate that to use **DispatcherQueue** instead.

ASTA to STA threading model

For more detail on the ASTA threading model, see the blog post What is so special about the Application STA?.

As the Windows App SDK's STA threading model doesn't have the same assurances around preventing reentrancy issues, if your UWP app assumes the non re-entrant behavior of the ASTA threading model, then your code might not behave as expected.

One thing to watch out for is reentrancy into XAML controls (see the example in A Windows App SDK migration of the UWP Photo Editor sample app (C++/WinRT)). And for some crashes, such as access violations, the direct crash callstack is usually the right stack to use. But if it's a *stowed exception* crash—which has exception code: 0xc000027b—then more work is required to get the right callstack.

Stowed exceptions

Stowed exception crashes save away a possible error, and that gets used later if no part of the code handles the exception. XAML sometimes decides the error is fatal immediately, in which case the direct crash stack might be good. But more frequently

the stack has unwound before it was determined to be fatal. For more details about stowed exceptions, see the Inside Show episode Stowed Exception C000027B.

For stowed exception crashes (to see a nested message pump, or to see the XAML control's specific exception being thrown), you can get more info on the crash by loading a crash dump in the Windows Debugger (WinDbg) (see Download debugging tools for Windows), and then using `!pde.dse` to dump the stowed exceptions.

The PDE debugger extension (for the `!pde.dse` command) is available by downloading the PDE*.zip file from OneDrive. Put the appropriate x64 or x86 `.dll` from that zip file into the `winext` directory of your WinDbg install, and then `!pde.dse` will work on stowed exception crash dumps.

Frequently there'll be multiple stowed exceptions, with some at the end that were handled/ignored. Most commonly, the first stowed exception is the interesting one. In some cases, the first stowed exception might be a re-throw of the second, so if the second stowed exception shows deeper into the same stack as the first, then the second exception might be the origination of the error. The error code shown with each stowed exception is also valuable, since that provides the **HRESULT** associated with that exception.

Change Windows.UI.Core.CoreDispatcher to Microsoft.UI.Dispatching.DispatcherQueue

This section applies if you're using the **Windows.UI.Core.CoreDispatcher** class in your UWP app. That includes the use of any methods or properties that take or return a **CoreDispatcher**, such as the **DependencyObject.Dispatcher** and **CoreWindow.Dispatcher** properties. For example, you'll be calling **DependencyObject.Dispatcher** when you retrieve the **CoreDispatcher** belonging to a **Windows.UI.Xaml.Controls.Page**.

C#

```
// MainPage.xaml.cs in a UWP app
if (this.Dispatcher.HasThreadAccess)
{
    ...
}
```

Instead, in your Windows App SDK app, you'll need to use the **Microsoft.UI.Dispatching.DispatcherQueue** class. And the corresponding methods or properties that take or return a **DispatcherQueue**, such as the

DependencyObject.DispatcherQueue and Microsoft.UI.Xaml.Window.DispatcherQueue properties. For example, you'll be calling **DependencyObject.DispatcherQueue** when you retrieve the **DispatcherQueue** belonging to a Microsoft.UI.Xaml.Controls.Page (most XAML objects are **DependencyObject**s).

C#

```
// MainPage.xaml.cs in a Windows App SDK app
if (this.DispatcherQueue.HasThreadAccess)
{
    ...
}
```

Change CoreDispatcher.RunAsync to DispatcherQueue.TryEnqueue

This section applies if you're using the Windows.UI.Core.CoreDispatcher.RunAsync method to schedule a task to run on the main UI thread (or on the thread associated with a particular Windows.UI.Core.CoreDispatcher).

C#

```
// MainPage.xaml.cs in a UWP app
public void NotifyUser(string strMessage)
{
    if (this.Dispatcher.HasThreadAccess)
    {
        StatusBlock.Text = strMessage;
    }
    else
    {
        var task = this.Dispatcher.RunAsync(
            Windows.UI.Core.CoreDispatcherPriority.Normal,
            () => StatusBlock.Text = strMessage);
    }
}
```

In your Windows App SDK app, use the Microsoft.UI.Dispatching.DispatcherQueue.TryEnqueue) method instead. It adds to the Microsoft.UI.Dispatching.DispatcherQueue a task that will be executed on the thread associated with the **DispatcherQueue**.

C#

```
// MainPage.xaml.cs in a Windows App SDK app
public void NotifyUser(string strMessage)
{
    if (this.DispatcherQueue.HasThreadAccess)
    {
        StatusBlock.Text = strMessage;
    }
    else
    {
        bool isQueued = this.DispatcherQueue.TryEnqueue(
        Microsoft.UI.Dispatching.DispatcherQueuePriority.Normal,
        () => StatusBlock.Text = strMessage);
    }
}
```

Migrate winrt::resume_foreground (C++/WinRT)

This section applies if you use the **winrt::resume_foreground** function in a coroutine in your C++/WinRT UWP app.

In UWP, the use case for **winrt::resume_foreground** is to switch execution to a foreground thread (that foreground thread is often the one that's associated with a **Windows.UI.Core.CoreDispatcher**). Here's an example of that.

```
// MainPage.cpp in a UWP app
winrt::fire_and_forget MainPage::ClickHandler(IInspectable const&,
RoutedEventArgs const&)
{
    ...
    co_await winrt::resume_foreground(this->Dispatcher());
    ...
}
```

In your Windows App SDK app:

- Instead of **winrt::resume_foreground**, you'll need to use **wil::resume_foreground** (from the Windows Implementation Libraries (WIL)).
- And instead of **CoreDispatcher**, you'll need to use the **Microsoft.UI.Dispatching.DispatcherQueue** class, as described in Change Windows.UI.Core.CoreDispatcher to Microsoft.UI.Dispatching.DispatcherQueue.

So first add a reference to the Microsoft.Windows.ImplementationLibrary NuGet package.

Then add the following include to `pch.h` in the target project.

C++/WinRT

```
#include <wil/cppwinrt_helpers.h>
```

And then follow the pattern shown below.

C++/WinRT

```
// MainPage.xaml.cpp in a Windows App SDK app
...
winrt::fire_and_forget MainPage::ClickHandler(IInspectable const&,
RoutedEventArgs const&)
{
    ...
    co_await wil::resume_foreground(this->DispatcherQueue());
    ...
}
```

See Also

- Windows App SDK and supported Windows releases

A Windows App SDK migration of the UWP PhotoLab sample app (C#)

Article • 03/27/2023

This topic is a case study of taking the C# UWP PhotoLab sample app, and migrating it to the Windows App SDK.

Begin by cloning the UWP sample app's repo, and opening the solution in Visual Studio.

> **Important**
>
> For considerations and strategies for approaching the migration process, and how to set up your development environment for migrating, see **Overall migration strategy**. It's particularly important to see **What's supported when porting from UWP to WinUI 3** so that you can ensure that all the features you need for your app are supported before you attempt migration.

Install tools for the Windows App SDK

To set up your development computer, see Install tools for the Windows App SDK.

> **Important**
>
> You'll find release notes topics along with the **Windows App SDK release channels** topic. There are release notes for each channel. Be sure to check any *limitations and known issues* in those release notes, since those might affect the results of following along with this case study and/or running the migrated app.

Create a new project

In Visual Studio, create a new C# project from the **Blank App, Packaged (WinUI 3 in Desktop)** project template. Name the project *PhotoLabWinUI*, uncheck **Place solution and project in the same directory**. You can target the most recent release (not preview) of the client operating system.

> **Note**

We'll be referring to the UWP version of the sample project (the one that you cloned from its **repo**) as the *source* solution/project. We'll be referring to the Windows App SDK version as the *target* solution/project.

The order in which we'll migrate the code

MainPage is an important and prominent piece of the app. But if we were to begin by migrating that, then we'd soon realize that **MainPage** has a dependency on the **DetailPage** view; and then that **DetailPage** has a dependency on the **ImageFileInfo** model. So for this walkthrough we'll take this approach.

- We'll begin by copying over the asset files.
- Then we'll migrate the **ImageFileInfo** model.
- Next we'll migrate the **App** class (since that needs changes making to it that **DetailPage**, **MainPage**, and **LoadedImageBrush** will depend on).
- Then we'll migrate the **LoadedImageBrush** class.
- Then we'll begin migrating the views, starting with **DetailPage** first.
- And we'll finish up by migrating the **MainPage** view.

Copy asset files

1. In your target project in Visual Studio, in **Solution Explorer**, right-click the **Assets** folder, and add a new folder named `Samples`.

2. In your clone of the source project, in **File Explorer**, locate the folder **Windows-appsample-photo-lab** > **PhotoLab** > **Assets**. You'll find seven asset files in that folder, together with a subfolder named **Samples** containing sample images. Select those seven asset files, and the **Samples** subfolder, and copy them to the clipboard.

3. Also in **File Explorer**, now locate the corresponding folder in the target project that you created. The path to that folder is **PhotoLabWinUI** > **PhotoLabWinUI** > **Assets**. Paste into that folder the asset files and the subfolder that you just copied, and accept the prompt to replace any files that already exist in the destination.

4. In your target project in Visual Studio, in **Solution Explorer**, with the **Assets** folder expanded, you'll see in the **Samples** folder the contents of the **Samples** subfolder (which you just pasted). You can hover the mouse pointer over the asset files. A thumbnail preview will appear for each, confirming that you've replaced/added the asset files correctly.

Migrate the ImageFileInfo model

ImageFileInfo is a *model* (in the sense of models, views, and viewmodels) that represents an image file, such as a photo.

Copy ImageFileInfo source code files

1. In your clone of the source project, in **File Explorer**, locate the folder **Windows-appsample-photo-lab** > **PhotoLab**. In that folder you'll find the source code file `ImageFileInfo.cs`; that file contains the implementation of **ImageFileInfo**. Select that file, and copy it to the clipboard.

2. In Visual Studio, right-click the target project node, and click **Open Folder in File Explorer**. This opens the target project folder in **File Explorer**. Paste into that folder the file that you just copied.

Migrate ImageFileInfo source code

1. Make the following find/replacements (match case and whole word) in the `ImageFileInfo.cs` file that you just pasted.

- `namespace PhotoLab` => `namespace PhotoLabWinUI`
- `Windows.UI.Xaml` => `Microsoft.UI.Xaml`

Windows.UI.Xaml is the namespace for UWP XAML; **Microsoft.UI.Xaml** is the namespace for WinUI XAML.

> ⓘ **Note**
>
> The Mapping UWP APIs to the Windows App SDK topic provides a mapping of UWP APIs to their Windows App SDK equivalents. The change we made above is an example of a namespace name change necessary during the migration process.

2. Now confirm that you can build the target solution (but don't run yet).

Migrate the App class

1. From the source project, in the `<Application.Resources>` element in `App.xaml`, find the following four lines. Copy them, and paste them into the target project.

```
XAML
```

```
<SolidColorBrush x:Key="RatingControlSelectedForeground" Color="White"/>
<!--  Window width adaptive breakpoints.  -->
<x:Double x:Key="MinWindowBreakpoint">0</x:Double>
<x:Double x:Key="MediumWindowBreakpoint">641</x:Double>
<x:Double x:Key="LargeWindowBreakpoint">1008</x:Double>
```

> **Note**
>
> Because the target project will use different (and simpler) navigation from the source project, there's no need to copy over any further code from the source project's `App.xaml.cs`.

2. In the target project, **App** stores the main window object in its private field *m_window*. Later in the migration process (when we migrate the source project's use of **Window.Current**), it'll be convenient if that private field is instead a public static property. So replace the *m_window* field with a *Window* property, and change references to *m_window*, as shown below.

C#

```
// App.xaml.cs
public partial class App : Application
{
    ...
    protected override void
OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
    {
        Window = new MainWindow();
        Window.Activate();
    }

    public static MainWindow Window { get; private set; }
}
```

3. Later in the migration process (when we migrate the code that displays a FileSavePicker), it'll be convenient if **App** exposes the main window's *handle* (**HWND**). So add a *WindowHandle* property, and initialize it in the **OnLaunched** method, as shown below.

C#

```
// App.xaml.cs
public partial class App : Application
{
    ...
```

```
    protected override void
OnLaunched(Microsoft.UI.Xaml.LaunchActivatedEventArgs args)
    {
        Window = new MainWindow();
        Window.Activate();
        WindowHandle = WinRT.Interop.WindowNative.GetWindowHandle(Window);
    }

    public static IntPtr WindowHandle { get; private set; }
}
```

Migrate the LoadedImageBrush model

LoadedImageBrush is a specialization of **XamlCompositionBrushBase**. The *PhotoLab* sample app uses the **LoadedImageBrush** class to apply effects to photos.

Reference the Win2D NuGet package

To support code in **LoadedImageBrush**, the source project has a dependency on Win2D. So we'll also need a dependency on Win2D in our target project.

In the target solution in Visual Studio, click **Tools** > **NuGet Package Manager** > **Manage NuGet Packages for Solution...** > **Browse**, and type or paste *Microsoft.Graphics.Win2D*. Select the correct item in search results, check the *PhotoLabWinUI* project, and click **Install** to install the package into that project.

Copy LoadedImageBrush source code files

Copy `LoadedImageBrush.cs` from the source project to the target project in the same way that you copied `ImageFileInfo.cs`.

Migrate LoadedImageBrush source code

1. Make the following find/replacements (match case and whole word) in the `LoadedImageBrush.cs` file that you just pasted.

- `namespace PhotoLab` => `namespace PhotoLabWinUI`
- `Windows.UI.Composition` => `Microsoft.UI.Composition`
- `Windows.UI.Xaml` => `Microsoft.UI.Xaml`
- `Window.Current.Compositor` => `App.Window.Compositor` (see Change Windows.UI.Xaml.Window.Current to App.Window)

2. Confirm that you can build the target solution (but don't run yet).

Migrate the DetailPage view

DetailPage is the class that represents the photo editor page, where Win2D effects are toggled, set, and chained together. You get to the photo editor page by selecting a photo thumbnail on **MainPage**. **DetailPage** is a *model* (in the sense of models, views, and viewmodels).

Copy DetailPage source code files

Copy `DetailPage.xaml` and `DetailPage.xaml.cs` from the source project to the target project in the same way that you copied files in previous steps.

Migrate DetailPage source code

1. Make the following find/replacements (match case and whole word) in the `DetailPage.xaml` file that you just pasted.

- `PhotoLab` => `PhotoLabWinUI`

2. Make the following find/replacements (match case and whole word) in the `DetailPage.xaml.cs` file that you just pasted.

- `namespace PhotoLab` => `namespace PhotoLabWinUI`
- `Windows.UI.Colors` => `Microsoft.UI.Colors`
- `Windows.UI.Xaml` => `Microsoft.UI.Xaml`

3. For the next step, we'll make the change that's explained in ContentDialog, and Popup. So, still in `DetailPage.xaml.cs`, in the **ShowSaveDialog** method, immediately *before* the line `ContentDialogResult result = await saveDialog.ShowAsync();`, add this code.

C++/WinRT

```
if
(Windows.Foundation.Metadata.ApiInformation.IsApiContractPresent("Windows.Fo
undation.UniversalApiContract", 8))
{
    saveDialog.XamlRoot = this.Content.XamlRoot;
}
```

4. Still in `DetailPage.xaml.cs`, in the **OnNavigatedTo** method, delete the following two lines of code. Just those two lines; later in this case study, we'll reintroduce the back button functionality that we just removed.

```
C#

...
SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility =
    AppViewBackButtonVisibility.Visible;
...
SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility =
    AppViewBackButtonVisibility.Collapsed;
...
```

5. For this step, we'll make the change that's explained in MessageDialog, and Pickers. Still in `DetailPage.xaml.cs`, in the **ExportImage** method, immediately *before* the line `var outputFile = await fileSavePicker.PickSaveFileAsync();`, add this line of code.

```
C#

WinRT.Interop.InitializeWithWindow.Initialize(fileSavePicker,
App.WindowHandle);
```

MainPage has dependencies on **DetailPage**, which is why we migrated **DetailPage** first. But **DetailPage** has dependencies on **MainPage**, too, so we won't be able to build yet.

Migrate the MainPage view

The main page of the app represents the view that you see first when you run the app. It's the page that loads the photos from the **Samples** folder that's built into the sample app, and displays a tiled thumbnail view.

Copy MainPage source code files

Copy `MainPage.xaml` and `MainPage.xaml.cs` from the source project to the target project in the same way that you copied files in previous steps.

Migrate MainPage source code

1. Make the following find/replacements (match case and whole word) in the `MainPage.xaml` file that you just pasted.

- `PhotoLab` => `PhotoLabWinUI`

2. Still in `MainPage.xaml`, find the markup `animations:ReorderGridAnimation.Duration="400"`, and delete that.

3. Make the following find/replacements (match case and whole word) in the `MainPage.xaml.cs` file that you just pasted.

- `namespace PhotoLab` => `namespace PhotoLabWinUI`
- `Windows.UI.Xaml` => `Microsoft.UI.Xaml`

4. For this step, we'll make the change that's explained in ContentDialog, and Popup. So, still in `MainPage.xaml.cs`, in the **GetItemsAsync** method, immediately *before* the line `ContentDialogResult resultNotUsed = await unsupportedFilesDialog.ShowAsync();`, add this code.

C++/WinRT

```
if
(Windows.Foundation.Metadata.ApiInformation.IsApiContractPresent("Windows.Fo
undation.UniversalApiContract", 8))
{
    unsupportedFilesDialog.XamlRoot = this.Content.XamlRoot;
}
```

5. Still in `MainPage.xaml.cs`, in the **OnNavigatedTo** method, delete the following line of code.

C#

```
SystemNavigationManager.GetForCurrentView().AppViewBackButtonVisibility =
    AppViewBackButtonVisibility.Collapsed;
```

Later in this case study, we'll reintroduce the back button functionality that we just removed.

6. Confirm that you can build the target solution (but don't run yet).

Navigate to MainPage

The *PhotoLab* sample app uses navigation logic to navigate initially to **MainPage** (and then between **MainPage** and **DetailPage**). For more info about Windows App SDK apps that need navigation (and those that don't), see Do I need to implement page navigation?.

So the changes we'll make next support that navigation.

1. In `MainWindow.xaml`, delete the `<StackPanel>` element, and replace it with just a named `<Frame>` element. The result looks like this:

XAML

```
<Window ...>
    <Frame x:Name="rootFrame"/>
</Window>
```

2. In `MainWindow.xaml.cs`, delete the **myButton_Click** method.

3. Still in `MainWindow.xaml.cs`, add the following line of code to the constructor.

C#

```
public sealed partial class MainWindow : Window
{
    public MainWindow()
    {
        this.InitializeComponent();
        rootFrame.Navigate(typeof(MainPage));
    }
}
```

4. Confirm that you can build the target solution (but don't run yet).

Restoring back button functionality

1. In `DetailPage.xaml`, the root element is a **RelativePanel**. Add the following markup inside that **RelativePanel**, immediately after the **StackPanel** element.

XAML

```
<AppBarButton x:Name="BackButton" Click="BackButton_Click"
Margin="0,0,12,0">
    <SymbolIcon Symbol="Back"/>
</AppBarButton>
```

2. In `DetailPage.xaml.cs`, add the following two lines of code to the **OnNavigatedTo** method, in the place indicated.

C#

```
if (this.Frame.CanGoBack)
{
    BackButton.Visibility = Microsoft.UI.Xaml.Visibility.Visible;
}
else
{
    BackButton.Visibility = Microsoft.UI.Xaml.Visibility.Collapsed;
}
```

3. Still in `DetailPage.xaml.cs`, add the following event handler.

C#

```
private void BackButton_Click(object sender,
Microsoft.UI.Xaml.RoutedEventArgs e)
{
    Frame.GoBack();
}
```

Test the migrated app

Now build the project, and run the app to test it. Select an image, set a zoom level, choose effects, and configure them.

See Also

- Windows App SDK and supported Windows releases

A Windows App SDK migration of the UWP Photo Editor sample app (C++/WinRT)

Article • 04/01/2023

This topic is a case study of taking the C++/WinRT UWP Photo Editor sample app, and migrating it to the Windows App SDK.

- Begin by cloning the UWP sample app's repo, and opening the solution in Visual Studio.

> **Important**
>
> For considerations and strategies for approaching the migration process, and how to set up your development environment for migrating, see **Overall migration strategy**.

Install tools for the Windows App SDK

To set up your development computer, see Install tools for the Windows App SDK.

> **Important**
>
> You'll find release notes topics along with the **Windows App SDK release channels** topic. There are release notes for each channel. Be sure to check any *limitations and known issues* in those release notes, since those might affect the results of following along with this case study and/or running the migrated app.

Create a new project

- In Visual Studio, create a new C++/WinRT project from the **Blank App, Packaged (WinUI 3 in Desktop)** project template. Name the project *PhotoEditor*, uncheck **Place solution and project in the same directory**. You can target the most recent release (not preview) of the client operating system.

> **Note**

We'll be referring to the UWP version of the sample project (the one that you cloned from its **repo**) as the *source* solution/project. We'll be referring to the Windows App SDK version as the *target* solution/project.

The order in which we'll migrate the code

MainPage is an important and prominent piece of the app. But if we were to begin by migrating that, then we'd soon realize that **MainPage** has a dependency on the **DetailPage** view; and then that **DetailPage** has a dependency on the **Photo** model. So for this walkthrough we'll take this approach.

- We'll begin by copying over the asset files.
- Then we'll migrate the **Photo** model.
- Next we'll migrate the **App** class (because that needs some members adding to it that **DetailPage** and **MainPage** will depend on).
- Then we'll begin migrating the views, starting with **DetailPage** first.
- And we'll finish up by migrating the **MainPage** view.

We'll be copying entire source code files

In this walkthrough we'll be copying over source code files using **File Explorer**. If you prefer to copy file *contents* over, then see the Appendix: copying the contents of the **Photo** model's files section at the end of this topic for an example of how you could do that for **Photo** (and you could then apply a similar procedure to other types in the project). That option does involve a lot more steps, though.

Copy asset files

1. In your clone of the source project, in **File Explorer**, locate the folder **Windows-appsample-photo-editor** > **PhotoEditor** > **Assets**. You'll find eight asset files in that folder. Select those eight files, and copy them to the clipboard.

2. Also in **File Explorer**, now locate the corresponding folder in the target project that you created. The path to that folder is **PhotoEditor** > **PhotoEditor** > **Assets**. Paste into that folder the asset files that you just copied, and accept the prompt to replace the seven files that already exist in the destination.

3. In your target project in Visual Studio, in **Solution Explorer**, expand the **Assets** folder. Add to that folder the existing `bg1.png` asset file that you just pasted. You

can hover the mouse pointer over the asset files. A thumbnail preview will appear for each, confirming that you've replaced/added the asset files correctly.

Migrate the Photo model

Photo is a runtime class that represents a photo. It's a *model* (in the sense of models, views, and viewmodels).

Copy Photo source code files

1. In your clone of the source project, in **File Explorer**, locate the folder **Windows-appsample-photo-editor** > **PhotoEditor**. In that folder you'll find the three source code files `Photo.idl`, `Photo.h`, and `Photo.cpp`; those files together implement the **Photo** runtime class. Select those three files, and copy them to the clipboard.

2. In Visual Studio, right-click the target project node, and click **Open Folder in File Explorer**. This opens the target project folder in **File Explorer**. Paste into that folder the three files that you just copied.

3. Back in **Solution Explorer**, with the target project node selected, make sure that **Show All Files** is toggled on. Right-click the three files that you just pasted, and click **Include In Project**. Toggle **Show All Files** off.

4. In the source project, in **Solution Explorer**, `Photo.h` and `.cpp` are nested under `Photo.idl` to indicate that they're generated from (dependent upon) it. If you like that arrangement, then you can do the same thing in the target project by manually editing `\PhotoEditor\PhotoEditor\PhotoEditor\PhotoEditor.vcxproj` (you'll first need to **Save All** in Visual Studio). Find the following:

   XML

   ```
   <ClInclude Include="Photo.h" />
   ```

   And replace it with this:

   XML

   ```
   <ClInclude Include="Photo.h">
     <DependentUpon>Photo.idl</DependentUpon>
   </ClInclude>
   ```

Repeat that for `Photo.cpp`, and save and close the project file. When you set focus back to Visual Studio, click **Reload**.

Migrate Photo source code

1. In `Photo.idl`, search for the namespace name `Windows.UI.Xaml` (which is the namespace for UWP XAML), and change that to `Microsoft.UI.Xaml` (which is the namespace for WinUI XAML).

> **Note**
>
> The Mapping UWP APIs to the Windows App SDK topic provides a mapping of UWP APIs to their Windows App SDK equivalents. The change we made above is an example of a namespace name change necessary during the migration process.

2. In `Photo.cpp`, add `#include "Photo.g.cpp"` to the existing include directives, immediately after `#include "Photo.h"`. This is one of the Folder and file name differences (C++/WinRT) to be aware of between UWP and Windows App SDK projects.

3. Make the following find/replacement (match case and whole word) in the contents of all of the source code in the files that you just copied and pasted.

   - `Windows::UI::Xaml` => `Microsoft::UI::Xaml`

4. From `pch.h` in the source project, copy the following includes, and paste them into `pch.h` in the target project. This is a subset of the header files included in the source project; these are just the headers we need to support the code we've migrated so far.

   C++/WinRT

   ```
   #include <winrt/Microsoft.UI.Xaml.Media.Imaging.h>
   #include <winrt/Windows.Storage.h>
   #include <winrt/Windows.Storage.FileProperties.h>
   #include <winrt/Windows.Storage.Streams.h>
   ```

5. Now confirm that you can build the target solution (but don't run yet).

Migrate the App class

No changes are necessary to the target project's `App.idl` and `App.xaml`. But we do need to edit `App.xaml.h` and App`.xaml.cpp` to add some new members to the **App** class. We'll do so in a way that lets us build after each section (with the exception of the last section, which is about **App::OnLaunched**).

Making the main window object available

In this step we'll make the change that's explained in Change Windows.UI.Xaml.Window.Current to App.Window.

In the target project, **App** stores the main window object in its private data member *window*. Later in the migration process (when we migrate the source project's use of **Window.Current**), it'll be convenient if that *window* data member is static; and is also made available via an accessor function. So we'll make those changes next.

- Since we're making *window* static, we'll need to initialize it in `App.xaml.cpp` instead of via the default member initializer that the code is currently using. Here are what those changes look like in `App.xaml.h` and `App.xaml.cpp`.

C++/WinRT

```
// App.xaml.h
...
struct App : AppT<App>
{
     ...
     static winrt::Microsoft::UI::Xaml::Window Window(){ return window; }
};

private:
     static winrt::Microsoft::UI::Xaml::Window window;
};
...

// App.xaml.cpp
...
winrt::Microsoft::UI::Xaml::Window App::window{ nullptr };
...
```

App::OnNavigationFailed

The *Photo Editor* sample app uses navigation logic to navigate between **MainPage** and **DetailPage**. For more info about Windows App SDK apps that need navigation (and those that don't), see Do I need to implement page navigation?.

So the members we'll migrate in the next few sections all exist to support navigation within the app.

1. Let's begin by migrating the **OnNavigationFailed** event handler. Copy the declaration and the definition of that member function from the source project, and paste it into the target project (in `App.xaml.h` and `App.xaml.cpp`).

2. In the code you pasted into `App.xaml.h`, change `Windows::UI::Xaml` to `Microsoft::UI::Xaml`.

App::CreateRootFrame

1. The source project contains a helper function named **App::CreateRootFrame**. Copy the declaration and the definition of that helper function from the source project, and paste it into the target project (in `App.xaml.h` and `App.xaml.cpp`).

2. In the code you pasted into `App.xaml.h`, change `Windows::UI::Xaml` to `Microsoft::UI::Xaml`.

3. In the code you pasted into `App.xaml.cpp`, change the two occurrences of `Window::Current()` to `window` (which is the name of the data member of the **App** class that we saw earlier).

App::OnLaunched

The target project already contains an implementation of the **OnLaunched** event handler. Its parameter is a constant reference to a **Microsoft::UI::Xaml::LaunchActivatedEventArgs**, which is correct for the Windows App SDK (contrast that to the source project, which uses **Windows::ApplicationModel::Activation::LaunchActivatedEventArgs**, which is correct for UWP).

- We just need to merge the two definitions (source and target) of **OnLaunched** so that **App::OnLaunched** in `App.xaml.cpp` in the target project looks like the listing below. Note that it uses `window` (instead of `Window::Current()`, like the UWP version does).

C++/WinRT

```
void App::OnLaunched(LaunchActivatedEventArgs const&)
{
     window = make<MainWindow>();
```

```
        Frame rootFrame = CreateRootFrame();
        if (!rootFrame.Content())
        {
            rootFrame.Navigate(xaml_typename<PhotoEditor::MainPage>());
        }

        window.Activate();
    }
```

The code above gives **App** a dependency on **MainPage**, so we won't be able to build from this point until we've migrated **DetailPage** and then **MainPage**. When we're able to build again, we'll say so.

Migrate the DetailPage view

DetailPage is the class that represents the photo editor page, where Win2D effects are toggled, set, and chained together. You get to the photo editor page by selecting a photo thumbnail on **MainPage**. **DetailPage** is a *view* (in the sense of models, views, and viewmodels).

Reference the Win2D NuGet package

To support code in **DetailPage**, the source project has a dependency on Microsoft.Graphics.Win2D. So we'll also need a dependency on Win2D in our target project.

- In the target solution in Visual Studio, click **Tools** > **NuGet Package Manager** > **Manage NuGet Packages for Solution...** > **Browse**. Make sure that **Include prerelease** is unchecked, and type or paste *Microsoft.Graphics.Win2D* into the search box. Select the correct item in search results, check the *PhotoEditor* project, and click **Install** to install the package.

Copy DetailPage source code files

1. In your clone of the source project, in **File Explorer**, locate the folder **Windows-appsample-photo-editor** > **PhotoEditor**. In that folder you'll find the four source code files `DetailPage.idl`, `DetailPage.xaml`, `DetailPage.h`, and `DetailPage.cpp`; those files together implement the **DetailPage** view. Select those four files, and copy them to the clipboard.

2. In Visual Studio, right-click the target project node, and click **Open Folder in File Explorer**. This opens the target project folder in **File Explorer**. Paste into that folder

the four files that you just copied.

3. Still in **File Explorer**, change the names of `DetailPage.h` and `DetailPage.cpp` to `DetailPage.xaml.h` and `DetailPage.xaml.cpp`, respectively. This is one of the Folder and file name differences (C++/WinRT) to be aware of between UWP and Windows App SDK projects.

4. Back in **Solution Explorer**, with the target project node selected, make sure that **Show All Files** is toggled on. Right-click the four files that you just pasted (and renamed), and click **Include In Project**. Toggle **Show All Files** off.

5. In the source project, in **Solution Explorer**, `DetailPage.idl` is nested under `DetailPage.xaml`. If you like that arrangement, then you can do the same thing in the target project by manually editing `\PhotoEditor\PhotoEditor\PhotoEditor.vcxproj` (you'll first need to **Save All** in Visual Studio). Find the following:

   ```
   XML
   <Midl Include="DetailPage.idl" />
   ```

   And replace it with this:

   ```
   XML
   <Midl Include="DetailPage.idl">
     <DependentUpon>DetailPage.xaml</DependentUpon>
   </Midl>
   ```

Save and close the project file. When you set focus back to Visual Studio, click **Reload**.

Migrate DetailPage source code

1. In `DetailPage.idl`, search for `Windows.UI.Xaml`, and change that to `Microsoft.UI.Xaml`.

2. In `DetailPage.xaml.cpp`, change `#include "DetailPage.h"` to `#include "DetailPage.xaml.h"`.

3. Immediately below that, add `#include "DetailPage.g.cpp"`.

4. For the call to the static **App::Window** method (which we're about to add) to compile, still in `DetailPage.xaml.cpp`, add `#include "App.xaml.h"` immediately

before `#include "Photo.h"`.

5. Make the following find/replacements (match case and whole word) in the contents of the source code in the files that you just copied and pasted.

   - In `DetailPage.xaml.h` and `.xaml.cpp`, `Windows::UI::Composition` => `Microsoft::UI::Composition`
   - In `DetailPage.xaml.h` and `.xaml.cpp`, `Windows::UI::Xaml` => `Microsoft::UI::Xaml`
   - In `DetailPage.xaml.cpp`, `Window::Current()` => `App::Window()`

6. From `pch.h` in the source project, copy the following includes, and paste them into `pch.h` in the target project.

   C++/WinRT

   ```
   #include <winrt/Windows.Graphics.Effects.h>
   #include <winrt/Microsoft.Graphics.Canvas.Effects.h>
   #include <winrt/Microsoft.Graphics.Canvas.UI.Xaml.h>
   #include <winrt/Microsoft.UI.Composition.h>
   #include <winrt/Microsoft.UI.Xaml.Input.h>
   #include <winrt/Windows.Graphics.Imaging.h>
   #include <winrt/Windows.Storage.Pickers.h>
   ```

7. Also, at the top of `pch.h`, immediately after `#pragma once`, add this:

   C++/WinRT

   ```
   // This is required because we are using std::min and std::max,
   otherwise
   // we have a collision with min and max macros being defined elsewhere.
   #define NOMINMAX
   ```

We can't build yet, but we will be able to after migrating **MainPage** (which is next).

Migrate the MainPage view

The main page of the app represents the view that you see first when you run the app. It's the page that loads the photos from the **Pictures Library**, and displays a tiled thumbnail view.

Copy MainPage source code files

1. Similar to what you did with **DetailPage**, now copy over `MainPage.idl`, `MainPage.xaml`, `MainPage.h`, and `MainPage.cpp`.

2. Rename the `.h` and `.cpp` files to `.xaml.h` and `.xaml.cpp`, respectively.

3. Include all four files in the target project like before.

4. In the source project, in **Solution Explorer**, `MainPage.idl` is nested under `MainPage.xaml`. If you like that arrangement, then you can do the same thing in the target project by manually editing `\PhotoEditor\PhotoEditor\PhotoEditor.vcxproj`. Find the following:

   XML

   ```
   <Midl Include="MainPage.idl" />
   ```

   And replace it with:

   XML

   ```
   <Midl Include="MainPage.idl">
     <DependentUpon>MainPage.xaml</DependentUpon>
   </Midl>
   ```

Migrate MainPage source code

1. In `MainPage.idl`, search for `Windows.UI.Xaml`, and change both occurrences to `Microsoft.UI.Xaml`.

2. In `MainPage.xaml.cpp`, change `#include "MainPage.h"` to `#include "MainPage.xaml.h"`.

3. Immediately below that, add `#include "MainPage.g.cpp"`.

4. For the call to the static **App::Window** method (which we're about to add) to compile, still in `MainPage.xaml.cpp`, add `#include "App.xaml.h"` immediately before `#include "Photo.h"`.

For the next step, we'll make the change that's explained in ContentDialog, and Popup.

5. So, still in `MainPage.xaml.cpp`, in the **MainPage::GetItemsAsync** method, immediately after the line `ContentDialog unsupportedFilesDialog{};`, add this line of code.

C++/WinRT

```
unsupportedFilesDialog.XamlRoot(this->Content().XamlRoot());
```

6. Make the following find/replacements (match case and whole word) in the contents of the source code in the files that you just copied and pasted.
   - In `MainPage.xaml.h` and `.xaml.cpp`, `Windows::UI::Composition` => `Microsoft::UI::Composition`
   - In `MainPage.xaml.h` and `.xaml.cpp`, `Windows::UI::Xaml` => `Microsoft::UI::Xaml`
   - In `MainPage.xaml.cpp`, `Window::Current()` => `App::Window()`
7. From `pch.h` in the source project, copy the following includes, and paste them into `pch.h` in the target project.

C++/WinRT

```
#include <winrt/Microsoft.UI.Xaml.Hosting.h>
#include <winrt/Microsoft.UI.Xaml.Media.Animation.h>
#include <winrt/Windows.Storage.Search.h>
```

Confirm that you can build the target solution (but don't run yet).

Update MainWindow

1. In `MainWindow.xaml`, delete the **StackPanel** and its contents, since we don't need any UI in **MainWindow**. That leaves only the empty **Window** element.
2. In `MainWindow.idl`, delete the placeholder `Int32 MyProperty;`, leaving only the constructor.
3. In `MainWindow.xaml.h` and `MainWindow.xaml.cpp`, delete the declarations and definitions of the placeholder **MyProperty** and **myButton_Click**, leaving only the constructor.

Migration changes needed for threading model difference

The two changes in this section are necessary due to a threading model difference between UWP and the Windows App SDK, as described in ASTA to STA threading model.

Here are brief descriptions of the causes of the issues, and then a way to resolve each.

MainPage

MainPage loads image files from your **Pictures** folder, calls StorageItemContentProperties.GetImagePropertiesAsync to get the image file's properties, creates a **Photo** model object for each image file (saving those same properties in a data member), and adds that **Photo** object to a collection. The collection of **Photo** objects is data-bound to a **GridView** in the UI. On behalf of that **GridView**, **MainPage** handles the ContainerContentChanging event, and for phase 1 that handler calls into a coroutine that calls StorageFile.GetThumbnailAsync. This call to **GetThumbnailAsync** results in messages being pumped (it doesn't return immediately, and do *all* of its work async), and that causes reentrancy. The result is that the **GridView** has its **Items** collection changed while layout is taking place, and that causes a crash.

If we comment out the call to **StorageItemContentProperties::GetImagePropertiesAsync**, then we don't get the crash. But the real fix is to make the **StorageFile.GetThumbnailAsync** call be explicitly async by cooperatively awaiting **wil::resume_foreground** immediately before calling **GetThumbnailAsync**. This works because **wil::resume_foreground** schedules the code that follows it to be a task on the **DispatcherQueue**.

Here's the code to change:

C++/WinRT

```
// MainPage.xaml.cpp
IAsyncAction MainPage::OnContainerContentChanging(ListViewBase sender,
ContainerContentChangingEventArgs args)
{
    ...
    if (args.Phase() == 1)
    {
        ...
        try
        {
            co_await wil::resume_foreground(this->DispatcherQueue());
            auto thumbnail = co_await impleType-
>GetImageThumbnailAsync(this->DispatcherQueue());
            image.Source(thumbnail);
        }
        ...
    }
}
```

Photo

The **Photo::ImageTitle** property is data-bound to the UI, so the UI calls into the accessor function for that property whenever it needs its value. But when we try to access **ImageProperties.Title** from that accessor function on the UI thread, we get an access violation.

So instead, we can just access that **Title** one-time, from the constructor of **Photo**, and store it in the *m_imageName* data member if it's not empty. Then in the **Photo::ImageTitle** accessor function we need only to access the *m_imageName* data member.

Here's the code to change:

```
// Photo.h
...
Photo(Photo(Windows::Storage::FileProperties::ImageProperties const& props,
    ...
    ) : ...
{
    if (m_imageProperties.Title() != L"")
    {
        m_imageName = m_imageProperties.Title();
    }
}
...
hstring ImageTitle() const
{
    return m_imageName;
}
...
```

Those are the last of the changes we need to make to migrate the *Photo Editor* sample app. In the **Test the migrated app** section we'll confirm that we've correctly followed the steps.

Known issues

App type issue (affects only Preview 3)

If you followed along with this case study using the project template from the VSIX for Windows App SDK version 1.0 Preview 3, then you'll need to make a small correction to `PhotoEditor.vcxproj`. Here's how to do that.

In Visual Studio, in **Solution Explorer**, right-click the project node, and click **Unload Project**. Now `PhotoEditor.vcxproj` is open for editing. As the first child of **Project**, add a **PropertyGroup** element like this:

XML

```
<Project ... >
    <PropertyGroup>
        <EnableWin32Codegen>true</EnableWin32Codegen>
    </PropertyGroup>
    <Import ... />
...
```

Save and close `PhotoEditor.vcxproj`. Right-click the project node, and click **Reload Project**. Now rebuild the project.

Test the migrated app

Now build the project, and run the app to test it. Select an image, set a zoom level, choose effects, and configure them.

Appendix: copying the contents of the Photo model's files

As we discussed earlier, you have the option to copy over source code *files* themselves, or the *contents* of source code files. We've already shown how to copy over source code *files* themselves. So this section gives an example of copying file *contents*.

In the source project in Visual Studio, locate the folder **PhotoEditor (Universal Windows)** > **Models**. That folder contains the files `Photo.idl`, `Photo.h`, and `Photo.cpp`, which together implement the **Photo** runtime class.

Add the IDL, and generate stubs

In your target project in Visual Studio, add a new **Midl File (.idl)** item to the project. Name the new item `Photo.idl`. Delete the default contents of `Photo.idl`.

From the source project in Visual Studio, copy the contents of **Models** > `Photo.idl`, and paste them into the `Photo.idl` file that you just added to your target project. In the code you pasted, search for `Windows.UI.Xaml`, and change that to `Microsoft.UI.Xaml`.

Save the file.

> **ⓘ Important**
>
> We're about to perform a build of your target solution. The build won't run to completion at this point, but it will get far enough to do necessary work for us.

Now build the target solution. Even though it won't complete, building is necessary now because it'll generate the source code files (stubs) that we need to get started implementing the **Photo** model.

In Visual Studio, right-click the target project node, and click **Open Folder in File Explorer**. This opens the target project folder in **File Explorer**. There, navigate into the `Generated Files\sources` folder (so you'll be in `\PhotoEditor\PhotoEditor\PhotoEditor\Generated Files\sources`). Copy the stub files `Photo.h` and `.cpp`, and paste them into the project folder, which is now up two folder levels in `\PhotoEditor\PhotoEditor\PhotoEditor`.

Back in **Solution Explorer**, with the target project node selected, make sure that **Show All Files** is toggled on. Right-click the stub files that you just pasted (`Photo.h` and `.cpp`), and click **Include In Project**. Toggle **Show All Files** off.

You'll see a `static_assert` at the top of the contents of `Photo.h` and `.cpp`, which you'll need to delete.

Confirm that you can build again (but don't run yet).

Migrate code into the stubs

Copy the contents of `Photo.h` and `.cpp` from the source project into the target project.

From here, the remaining steps to migrate the code that you copied are the same as those given in the Migrate **Photo** source code section.

Related topics

- Windows App SDK and supported Windows releases
- UWP Photo Editor sample app
- Overall migration strategy
- Mapping UWP APIs to the Windows App SDK
- WinUI migration

Additional migration guidance

Article • 06/24/2024

This topic contains additional migration guidance not categorized into a feature area in the feature area guides.

Conditional compilation

The info in this section might be useful if you plan to use the same source code file in both a UWP and a Windows App SDK project.

In C# source code in a Windows App SDK project, you can use preprocessor directives with the **WINDOWS_UWP** symbol to perform conditional compilation.

C#

```
#if !WINDOWS_UWP
    // Win32/Desktop code, including Windows App SDK code
#else
    // UWP code
#endif
```

In C++/WinRT source code in a Windows App SDK project, you can use preprocessor directives with **WINAPI_FAMILY_PC_APP** to do the same thing. Or you could use **WINAPI_FAMILY_DESKTOP_APP** instead. A comment in the `winapifamily.h` header file indicates that **WINAPI_FAMILY_APP** should be considered deprecated.

C++/WinRT

```
#if (WINAPI_FAMILY == WINAPI_FAMILY_DESKTOP_APP)
    // Win32/Desktop code, including Windows App SDK code
#else
    // UWP code
#endif
```

You can also use conditional compilation in XAML markup.

XAML

```
<Application
    ...
    xmlns:nouwp="condition:!WINDOWS_UWP"
    mc:Ignorable="nouwp">
    <Application.Resources>
```

```
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <!--Not Needed for UWP-->
                <nouwp:XamlControlsResources
xmlns="using:Microsoft.UI.Xaml.Controls" />
                 <!--Other merged dictionaries here-->
            </ResourceDictionary.MergedDictionaries>
             <!--Other app resources here-->
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

Also see Conditional compilation.

Debugging

During the migration process, you might find your app in a state where your XAML markup has references to XAML resource keys, but you haven't yet defined those keys. Such a condition results in a run-time crash that might not be straightforward to debug. But in a debug build, messages about missing resource keys appear in Visual Studio via debug output in the **Output** pane. So run your app under the debugger, and watch out for such messages.

Unregistering an event handler (C++/WinRT)

In a C++/WinRT project, you can manually revoke (unregister) an event handler such as **SizeChanged** (for more details, and code examples, see Revoke a registered delegate). But an alternative to manually revoking—and one that you could consider if you're having problems with manually revoking—is to use a C++/WinRT auto event revoker. Again, more details and code examples in Revoke a registered delegate.

See Also

- Windows App SDK and supported Windows releases

Migrate from UWP to the Windows App SDK with the .NET Upgrade Assistant

Article • 10/05/2023

The .NET Upgrade Assistant (see Overview of the .NET Upgrade Assistant) is a Visual Studio extension (recommended), and a command-line tool, that can assist with migrating a C# Universal Windows Platform (UWP) app to a WinUI 3 app that uses the Windows App SDK.

Our roadmap for UWP support in the .NET Upgrade Assistant includes further tooling improvements, and adding migration support for new features. If you find issues related to the .NET Upgrade Assistant, then you can file them within Visual Studio by selecting **Help** > **Send Feedback** > **Report a Problem**.

Also see the Upgrade Assistant GitHub repository. Options for running the tool on the command-line are documented there.

Install the .NET Upgrade Assistant

You can install the .NET Upgrade Assistant as a Visual Studio extension or as a .NET command-line tool. For more info, see Install the .NET Upgrade Assistant.

Summary

When you use the .NET Upgrade Assistant to migrate your UWP app, here are the high-level steps and stages in the migration process that the tool carries out.

- Optionally copies your project, and migrates the copy; leaving your original project unchanged.
- Optionally migrates your project in-place (in the same folders and files, without renaming folders); and doesn't make a copy.
- Upgrades your project from the .NET Framework project format to the latest .NET SDK project format.
- Cleans up NuGet package references. In addition to the packages referenced by your app, the `packages.config` file contains references to the dependencies of those packages. For example, if you added reference to package **A**, which depends on package **B**, then both packages would be referenced in the `packages.config` file. In the new project system, only the reference to package **A** is required. So this step analyzes the package references, and removes those that aren't required. Your

app is still referencing .NET Framework assemblies. Some of those assemblies might be available as NuGet packages. So this step analyzes those assemblies, and references the appropriate NuGet package.
- Targets .NET 6, and the Windows App SDK.
- Changes the target framework moniker (TFM) (see Target frameworks in SDK-style projects) from .NET Framework to the suggested SDK. For example, `net6.0-windows`.
- Migrates your UWP source code from WinUI 2 to WinUI 3, performing source-specific code changes.
- Adds/updates any template, config, and code files. For example, adding necessary publishing profiles, `App.xaml.cs`, `MainWindow.xaml.cs`, `MainWindow.xaml`, and others.
- Update namespaces, and adds **MainPage** navigation.
- Attempts to detect and fix APIs that are different between UWP and the Windows App SDK, and uses **Task List** TODOs to mark APIs that are no longer supported.

As it runs, the tool also aims to provide migration guidance in the form of warning messages within the tool's output, and **Task List** TODOs in the form of comments within your project's source code (for example, for cases where completely automated migration of your UWP source code isn't possible). A typical **Task List** TODO includes a link to a topic in this migration documentation. As the developer, you're always in control of the migration process.

> **Tip**
>
> To see all of the TODOs that the tool has generated, look in the **Task list** in Visual Studio.

> **Note**
>
> After the tool has finished running, there are some follow-up steps you can choose to do if needed. You can move your code from `App.xaml.old.cs` to `App.xaml.cs`; and you can restore `AssemblyInfo.cs` from the backup that the tool creates.

What the tool supports

This release of the .NET Upgrade Assistant is currently in preview, and is receiving frequent updates. The tool currently supports only the C# programming language; not

C++. And in most cases with this release, your project will require additional effort from you to complete the migration.

The tool aims to migrate your project and code so that it compiles. But some features require you to investigate and fix them (via **Task List** TODOs). For more information about what to consider before migrating, see What's supported when migrating from UWP to WinUI 3.

Because of the following limitations of the current release of the .NET Upgrade Assistant, you might choose to wait for a future release before migrating your app:

- Migrating from **ApplicationView** APIs isn't supported.
- Migrating from **AppWindow**-related APIs isn't supported.

Where possible, the tool tries to generate a warning; and it intentionally causes your code not to compile until you change it.

- Custom views aren't supported. For example, you won't receive a warning or a fix for a custom dialog that extends **MessageDialog**, and calls an API incorrectly.
- Windows Runtime Components aren't supported.

- Multi-window apps might not be migrated correctly.
- A project that follows a non-standard file structure (such as `App.xaml` and `App.xaml.cs` not being in the root folder) might not be migrated correctly.

The Upgrade Assistant GitHub repository documents troubleshooting tips and known issues. If you find any issues while using the tool, please report them in that same GitHub repository, tagging them with an area tag of `UWP`. We appreciate it!

> **Note**
>
> For guidance about the migration process—and the differences between UWP and Windows App SDK features and APIs—see Migrate from UWP to the Windows App SDK.

> **Tip**
>
> You can see what version of the tool you have by issuing the command `upgrade-assistant --version`.

Test drive the tool with the UWP PhotoLab sample

Let's take the .NET Upgrade Assistant for a test-drive.

As source material, we'll be migrating the UWP PhotoLab sample application. PhotoLab is a sample app for viewing and editing image files. It demonstrates XAML layout, data binding, and UI customization features.

> **Note**
>
> You can see a case study of the PhotoLab sample being fully manually migrated in A Windows App SDK migration of the UWP PhotoLab sample app.

1. Begin by cloning or downloading the PhotoLab sample source code from the link above.

> **Tip**
>
> Be aware that after we've used the tool to automate the migration of the app, additional manual effort will be needed to complete the migration.

1. Open the PhotoLab solution in Visual Studio.
2. Having installed the .NET Upgrade Assistant extension (see Install the .NET Upgrade Assistant earlier in this topic), right-click on the project in **Solution Explorer**, and click **Upgrade**.
3. Choose the **Upgrade project to a newer .NET version** option.
4. Choose the **In-place project upgrade** option.
5. Choose a target framework.
6. Click **Upgrade selection**.
7. The .NET Upgrade Assistant runs, and uses the Visual Studio **Output** window to print out info and status as it goes.

You can monitor the progress bar until the upgrade operation completes.

Code migration for the **PhotoLab** sample app includes:

- Changes to Content Dialog and File Save picker APIs.
- XAML update for the Animations package.
- Showing warning messages, and adding **Task List** TODOs in `DetailPage.xaml`, `DetailPage.xaml.cs`, and `MainPage.xaml.cs` for custom back button.
- Implementing the back button functionality and adding a **Task List** TODO to customize XAML button.
- A link to documentation is provided that you can use to learn more about back button implementation.

The version numbers in your resulting `.csproj` will be slightly different, but essentially it will look like this (with some of the build configuration property groups removed for brevity):

XML

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
    <Platform Condition=" '$(Platform)' == '' ">x86</Platform>
    <OutputType>WinExe</OutputType>
    <DefaultLanguage>en-US</DefaultLanguage>
    <TargetPlatformMinVersion>10.0.17763.0</TargetPlatformMinVersion>
    <RuntimeIdentifiers>win-x86;win-x64;win-arm64</RuntimeIdentifiers>
    <UseWinUI>true</UseWinUI>
    <ApplicationManifest>app.manifest</ApplicationManifest>
    <EnableMsixTooling>true</EnableMsixTooling>
    <Platforms>x86;x64;arm64</Platforms>
    <PublishProfile>win10-$(Platform).pubxml</PublishProfile>
  </PropertyGroup>
  <ItemGroup>
    <AppxManifest Include="Package.appxmanifest">
      <SubType>Designer</SubType>
    </AppxManifest>
  </ItemGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.WindowsAppSDK" Version="1.1.0" />
    <PackageReference Include="Microsoft.Graphics.Win2D" Version="1.0.0.30"
/>
    <PackageReference
Include="Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers"
Version="0.4.346201">
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.Windows.Compatibility"
Version="6.0.0" />
    <PackageReference Include="CommunityToolkit.WinUI.UI.Animations"
Version="7.1.2" />
  </ItemGroup>
  <ItemGroup>
    <Compile Remove="App.xaml.old.cs" />
```

```
  </ItemGroup>
  <ItemGroup>
    <None Include="App.xaml.old.cs" />
  </ItemGroup>
</Project>
```

As you can see, the project is now referencing the Windows App SDK, WinUI 3, and .NET 6. Now that **PhotoLab** has been migrated, you can take advantage of all of the new features that WinUI 3 apps have to offer, and grow your app with the platform.

Also, the .NET Upgrade Assistant adds analyzers to the project that assist with continuing with the upgrade process. For example, the **Microsoft.DotNet.UpgradeAssistant.Extensions.Default.Analyzers** NuGet package.

Follow-up manual migration

At this point you can open the migrated **PhotoLab** solution or project, and see the changes that have been made in the source code. The project needs a little more work to finish hooking things up before the WinUI 3 version builds, runs, and behaves like the UWP version.

See the **Task List** in Visual Studio (**View** > **Task List**) for TODOs that you should action to manually complete the migration.

It's possible that the UWP (.NET Framework) version of your app contained library references that your project isn't actually using. You'll need to analyze each reference, and determine whether or not it's required. The tool might also have added or upgraded a NuGet package reference to the wrong version.

The Upgrade Assistant doesn't edit the `Package.appxmanifest`, which will need some edits in order for the app to launch:

1. Add this namespace on the root <Package> element:

```
xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities"
```

2. Edit the <Application> element from `EntryPoint="appnamehere.App"` to `EntryPoint="$targetentrypoint$"`

3. Replace any specified `Capability` with this:

XML

```
<rescap:Capability Name="runFullTrust" />
```

In your `.csproj` file, you might need to edit the project file to set `<OutputType>WinExe</OutputType>` and `<UseMaui>False</UseMaui>`.

To use many of the XAML controls, ensure that your `app.xaml` file includes the `<XamlControlsResources>`, such as in this example:

XML

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <XamlControlsResources xmlns="using:Microsoft.UI.Xaml.Controls"
/>
            <!-- Other merged dictionaries here -->
        </ResourceDictionary.MergedDictionaries>
        <!-- Other app resources here -->
    </ResourceDictionary>
</Application.Resources>
```

Troubleshooting tips

There are several known problems that can occur when using the .NET Upgrade Assistant. In some cases, these problems are with the try-convert tool that the .NET Upgrade Assistant uses internally.

But for more troubleshooting tips and known issues, see the Upgrade Assistant GitHub repository.

See Also

- Windows App SDK and supported Windows releases

Modernize your desktop apps

Article • 10/24/2022

Windows 11 and Windows 10 offer many features you can use to deliver a modern experience in your desktop apps. Most of these features are available as modular components that you can adopt in your desktop apps at your own pace without having to rewrite your application for a different platform. You can enhance your existing desktop apps by choosing which Windows features to adopt.

This topic describes the features that you can use in your desktop apps today. For a tutorial that demonstrates how to modernize an existing app to use many of the features described in this topic, see the Modernize a WPF app tutorial.

> **Note**
>
> Do you need assistance migrating desktop apps to Windows 10 or later? The App Assure service provides direct, no-cost support to developers who are porting their apps to Windows 10 and later versions. This program is available to all ISVs and eligible enterprises. For more details on eligibility and about the program itself, visit /fasttrack/win-10-app-assure-assistance-offered. To get started now, submit your request.

Apply Windows 11 features

Windows 11 marks a visual evolution of the Windows operating system, and introduces new features that improve app fundamentals and user experience. Many of these features are enabled by default for apps, but desktop apps may require updates to integrate with some new features. These features include rounded corners of top-level windows, snap layouts, and the updated context menu in File Explorer.

For more information, see Top 11 things you can do to make your app great on Windows 11.

Windows App SDK

The Windows App SDK is a set of new developer components and tools that represent the next evolution in the Windows app development platform. The Windows App SDK provides a unified set of APIs and tools that can be used in a consistent way by any desktop app on Windows 11 and many versions of Windows 10. You can use project

templates to create new desktop apps that use the Windows App SDK with a WinUI 3-based UI, or you can use the Windows App SDK in existing desktop apps.

For more information, see Windows App SDK.

WinUI

WinUI is a native user experience framework for both Windows desktop and UWP applications. WinUI started as a toolkit that provided new and updated versions of WinRT XAML controls for UWP apps that target down-level versions of Windows. The latest version, WinUI 3, has grown in scope and is now the modern native UI platform for Windows desktop apps.

You can use WinUI in the following ways in desktop apps:

- Use WinUI 3 to create desktop apps (see Create your first WinUI 3 project) with an entirely WinUI 3-based user interface. WinUI 3 is one of many features provided by the Windows App SDK.
- You can update existing WPF, Windows Forms, and C++ desktop (Win32) apps with XAML Islands and host WinUI 2 controls.

For more information, see WinUI.

Windows Runtime APIs

You can call many Windows Runtime APIs directly in your WPF, Windows Forms, or C++ desktop app to integrate modern experiences that light up for users. For example, you can call Windows Runtime APIs to add toast notifications to your desktop app.

For more information, see Use Windows Runtime APIs in desktop apps.

MSIX deployment

MSIX is a modern Windows app package format that provides a universal packaging experience for all Windows apps, including UWP, WPF, Windows Forms and Win32 apps. MSIX brings together the best aspects of MSI, .appx, App-V and ClickOnce installation technologies to provide a modern and reliable packaging experience.

Packaging your desktop Windows apps in MSIX packages gets you access to a robust installation and updating experience, a managed security model with a flexible capability system, support for the Microsoft Store, enterprise management, and many custom distribution models.

For more information, see Building an MSIX package from your code.

Use MSIX framework packages dynamically at run time

The *dynamic dependencies* feature in the Windows App SDK and in the Windows 11 OS enables your apps to reference MSIX framework packages at run time. This feature is intended to be used primarily by unpackaged desktop apps to call APIs that are provided by MSIX framework packages.

For more information, see Use MSIX framework packages dynamically from your desktop app.

.NET

.NET (previously known as .NET Core) supports Windows desktop apps, including WinUI 3 apps created with the Windows App SDK (see Create your first WinUI 3 project). .NET also supports Windows Presentation Foundation (WPF) and Windows Forms (WinForms) apps. You can run new and existing Windows desktop apps on .NET, and enjoy all the benefits that .NET has to offer.

For more information, see What's new in .NET 6.

Host WinRT XAML controls (XAML Islands)

Starting with the Windows 10, version 1903, you can add UWP XAML controls directly to any UI element in a WPF, Windows Forms, or C++ desktop app that is associated with a window handle (HWND). This means that you can fully integrate the latest UWP features such as Windows Ink and controls that support the Fluent Design System into windows and other display surfaces in your desktop apps. This developer scenario is sometimes called *XAML islands*.

For more information, see WinRT XAML controls in desktop apps

Use the Visual layer in desktop apps

You can now use Windows Runtime APIs in non-UWP desktop apps to enhance the look, feel, and functionality of your WPF, Windows Forms, and C++ desktop apps, and take advantage of the latest Windows UI features that are only available via UWP. This is

useful when you need to create custom experiences that go beyond the built-in WinRT XAML controls you can host by using XAML Islands.

For more information, see Modernize your desktop app using the Visual layer.

Additional features available to apps with package identity

Some modern Windows experiences are available only to desktop apps that have package identity at runtime. These features include certain Windows Runtime APIs, package extensions, and UWP components. For more info, see Features that require package identity.

WinRT XAML controls optimized for desktop apps

Whether you're building a UWP app that exclusively targets the desktop device family or you want to use WinRT XAML controls in a WPF, Windows Forms, or C++ desktop app, the following new and updated WinRT XAML controls are designed to offer desktop-optimized experiences with the Fluent Design System. These controls were introduced in Windows 10, version 1809 (the October 2018 Update, or version 10.0.17763).

Expand table

Control	Description
MenuBar	Provides a quick and simple way to expose a set of commands for apps that might need more organization or grouping than a **CommandBar** allows.
DropDownButton	Shows a chevron as a visual indicator that it has an attached flyout that contains more options.
SplitButton	Provides a button has two parts that can be invoked separately. One part behaves like a standard button and invokes an immediate action. The other part invokes a flyout that contains additional options that the user can choose from.
ToggleSplitButton	Provides a button has two parts that can be invoked separately. One part behaves like a toggle button that can be on or off. The other part invokes a flyout that contains additional options that the user can choose from.
CommandBarFlyout	Lets you show common user tasks in the context of an item on your UI canvas.

Control	Description
ComboBox	You can now make a combo box editable so the user can enter values that aren't listed in the control.
TreeView	You can now configure a tree view to enable data binding, item templates, and drag and drop.
DataGridView	Provides a flexible way to display a collection of data in rows and columns. This control is available in the Windows Community Toolkit.

Other technologies for modern desktop apps

Microsoft Graph

Microsoft Graph is a collection of APIs you can use to build apps for organizations and consumers that interact with the data of millions of users. Microsoft Graph exposes REST APIs and client libraries to access data on the following:

- Azure Active Directory
- Microsoft 365 Office apps: SharePoint, OneDrive, Outlook/Exchange, Microsoft Teams, OneNote, Planner, and Excel
- Enterprise Mobility and Security services: Identity Manager, Intune, Advanced Threat Analytics, and Advanced Threat Protection.
- Windows services: activities and devices

For more information, see the Microsoft Graph docs.

Adaptive Cards

Adaptive Cards is an open, cross-platform framework that you can use to exchange card-based UI content in a common and consistent way across devices and platforms.

For more information, see the Adaptive Cards docs.

Top 11 things you can do to make your app great on Windows 11

Article • 03/06/2025

Windows 11 marks a visual evolution of the Windows operating system. As Windows moves forward, customers continue to set a higher bar for app experiences, too. We see these customer expectations manifest primarily in three areas: *app fundamentals*, *user experience*, and *added security*.

- *App fundamentals* - good performance on low-cost, highly mobile device form factors, and hassle-free app lifecycle and state rehydration/roaming.
- *User experience* - the ability to work naturally with a complete range of inputs, design and interaction patterns that look and feel at home on current and future devices, and support for modern windowing workflows and shell integration points.
- *Added security* - deploying apps on a secure platform that protects user data and privacy, and the ability to integrate features like Windows Hello and Passkeys to provide a more secure experience for users.

With entirely new visuals signaling change, signature experiences that showcase the best of Windows and Microsoft together, and a completely new Store with more apps, Windows 11 is positioned as the Windows for "what's next".

Great apps on Windows meet these customer expectations for app fundamentals, modern user experience, and added security, and we're investing in the native Windows platform to make it easier for your apps to achieve greatness through WinUI, MSIX, and the Windows App SDK family of APIs. This document provides an overview what you need to do to make your app look and behave great so that users feel like your app was made for Windows 11.

Here are the top ~~10~~ 11 things you can do to make your app shine on Windows 11.

1. Test your app to be compatible with Windows 11

Users expect solid fundamentals from any app they install on their device. They expect the app to just work when they upgrade it or migrate to a new version of the operating system. Compatibility testing helps to make sure there is no difference in functionality for your apps after upgrade and helps increase customer satisfaction with a well-tested product.

Test your app to be compatible with Windows 11

Microsoft is committed to ensuring your apps work on the latest versions of our software and Windows 11 has been built with compatibility in mind. Our promise states that apps that worked on Windows 7/8.1/10 will work on Windows 11. If you experience any issues with your apps, Microsoft will help you identify the issue at no cost so that you can then fix the issue. Visit the App Assure page and sign in to submit your request for assistance from a dedicated App Assure Manager.

2. Have a discoverable and easy to understand privacy policy

An easily discoverable and understandable privacy notice increases user trust and confidence in your application.

Your privacy policy must:

- Inform users of the personal information accessed, collected, or transmitted by your product.
  - How that information is used, stored, and secured.
  - Indicate the types of parties to whom that information is disclosed.
- Describe the controls that users have over the use and sharing of their information and how they may access their information.
- Comply with applicable laws and regulations.
- Be kept up-to-date as you add new features and functionality to your product.

Microsoft Privacy Statement

3. Make your app accessible and inclusive

Inclusive design creates better products for everyone. To make sure your app is accessible and inclusive, consider what improved functionality and usability means in relation to:

- People with disabilities (both temporary and permanent).
- Personal preferences.
- Specific work styles.
- Situational constraints (such as shared work spaces, driving, cooking, glare, and so on).

Some common solutions include providing information in alternative formats (such as captions on a video) or enabling the use of assistive technologies (such as screen

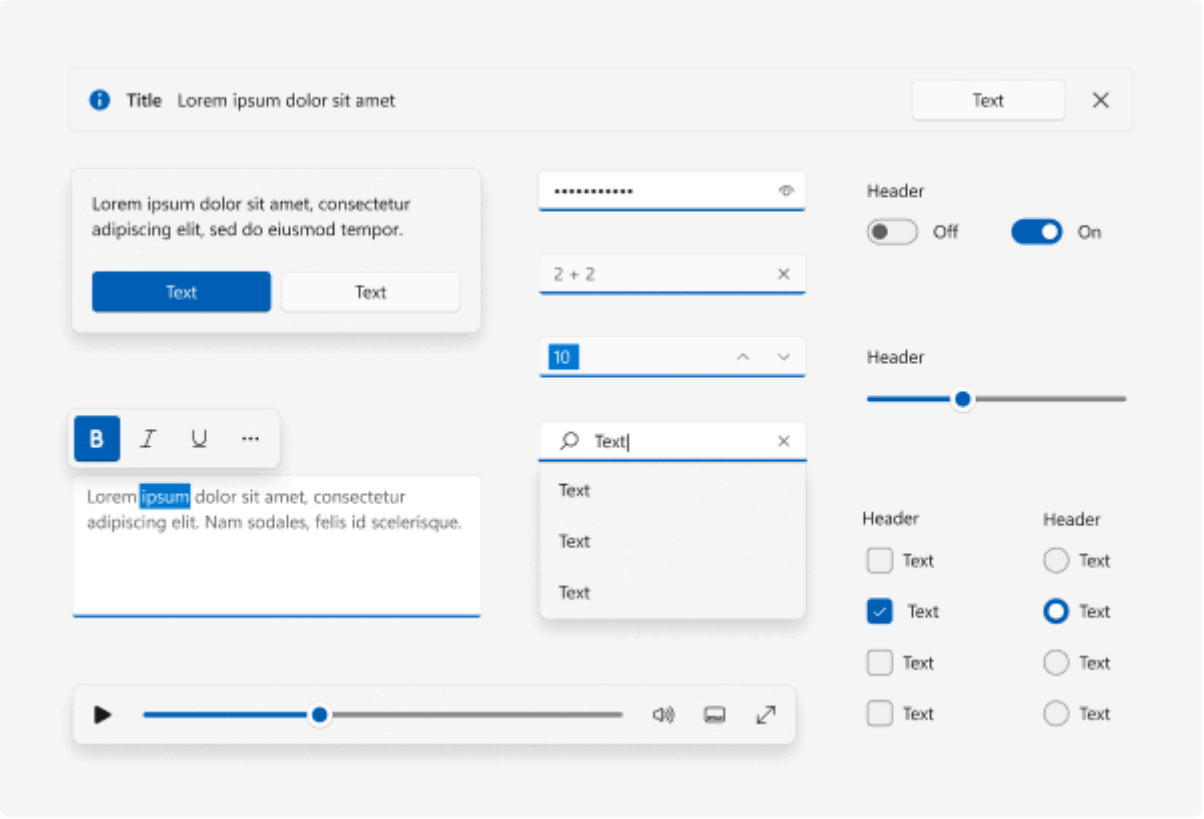
readers).

Applications designed with accessibility in mind are easier to maintain, update, and redesign. In addition to helping your app reach people with disabilities, factoring in accessibility can reduce the cost of maintaining your app.

Accessibility in Windows

4. Use the latest common controls

Windows 11 brings beautiful UI innovations to the Windows operating system that you can leverage in your apps. Common controls are one way that you can utilize these updates immediately. Use the latest common controls whenever possible to get the benefits of compatibility and accessibility for free. And these common controls are more cost effective than building your own custom controls when you factor in maintenance and testing costs.



Desktop apps (Win32)

- Use WinUI 3 in Windows App SDK 1.1 or greater to create a Win32 application that can leverage the modern common controls.
- To evaluate the controls, check out the WinUI 3 Gallery (*main branch*).
- Win32 controls that use UXTheme will automatically get a "repaint" of the Light theme for select rejuvenated control visuals. Be sure to test for any issues. For

those surfaces that do not get updated automatically, manually update the visuals so that they look coherent with the rest of the update, if possible.

UWP apps

- If your app uses UWP XAML or WinUI, update to use WinUI 2.6 or greater. WinUI provides new styles for both UWP XAML and WinUI controls, and the default styles have been updated with rejuvenated visuals and animation by default.
- See the documentation for a list of supported controls.
- Use the WinUI 2 Gallery (*winui 2 branch*) to evaluate the controls.

WebView and other platforms

- Individual WebView surfaces need to do work to adopt the style manually. We will not provide centralized styles, but the design toolkits and WinUI 2 Gallery can be used to copy the styles.

5. Use the latest design materials (Acrylic and Mica)

We are introducing a new material called *Mica* that lets the user's desktop background shine through your app. Mica is a very performant material that is meant to be used on long-lived UI surfaces like TitleBar to communicate the active or inactive state of the app. Mica is a texture that creates visual delight while saving battery life.

- Mica is to be used on the base layer of the app's UI to communicate the active state of the app; it falls back to a solid color when the app does not have focus. Thus, we recommend use of Mica on the TitleBar's background.
- Some controls, like NavigationView, already come built with the default behavior.

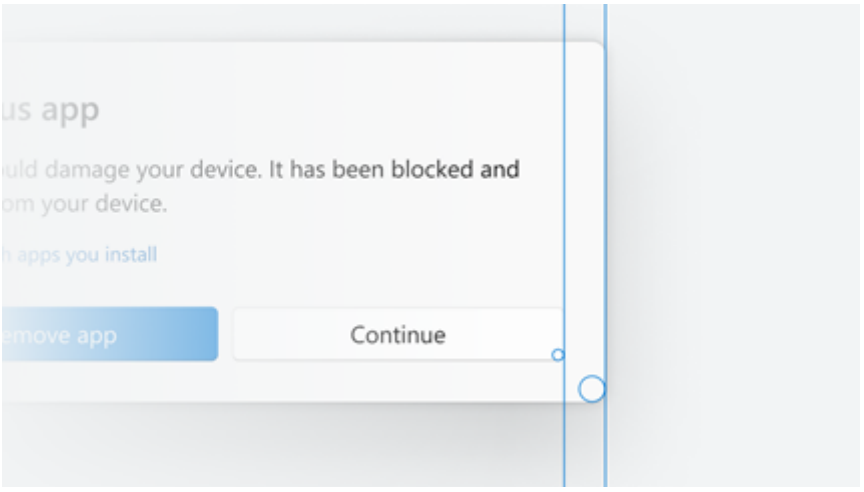
- When an app that uses Mica runs in Windows 10 or down-level, it will degrade gracefully (Mica will fallback to a solid color).
- Mica is to be used on long-lived surfaces, unlike Acrylic, which is to be used on transient surfaces.
- If you are using Acrylic material, follow the existing Acrylic guidance as we have updated the colors to be more vibrant.

Mica guidance, Acrylic guidance

6. Use rounded corners for your windows and support snap layouts

App windows have new features like rounded corners and a menu with snap layouts that your app will automatically receive in most cases. If you've customized your window or title bar, you might need to do some work to make sure these new features are supported.

We rounded the corners of window borders in Windows 11. Our user research team found that rounded geometry psychologically provides a feeling of safety and makes the app's UI much easier to scan. This makes users feel less intimidated and the app feel more engaging. The amount of rounding was also carefully chosen. We worked across the company and user research to balance between feeling professional and being softer and more inviting.



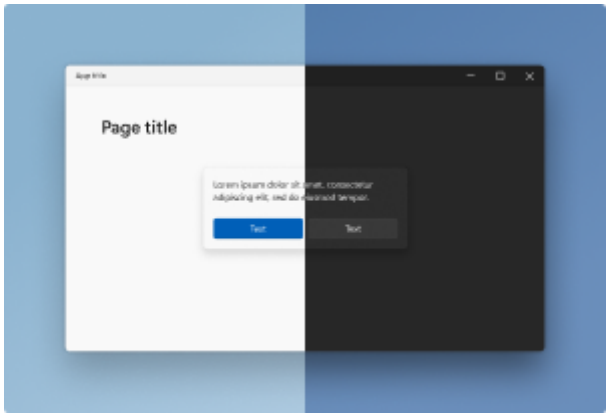
Snap layouts are a new Windows 11 feature to help introduce users to the power of window snapping. Snap layouts are easily accessible by hovering the mouse over a window's maximize button or pressing Win + Z. After invoking the menu that shows the available layouts, users can click on a zone in a layout to snap a window to that particular zone and then use Snap Assist to finish building an entire layout of windows. Snap layouts are tailored to the current screen size and orientation, including support

for three side-by-side windows on large landscape screens and top/bottom stacked windows on portrait screens.

- Most apps will automatically receive rounded corners and support for the menu with snap layouts, but in some cases you might need to do a little work to get them:
  - Allow the system to draw your border and shadow.
  - If you need to draw your own border and shadow:
    - Call our rounding API to opt in. See Apply rounded corners in desktop apps for Windows 11.
    - Use our APIs to have the platform draw and implement the caption buttons. See Support snap layouts for desktop apps on Windows 11.
- You will get these features automatically if you use UWP or you adopt Windows App SDK windowing to:
  - Configure the style of your window using the pre-defined templates.
  - Customize the title bar of your windows.

7. Support Dark and Light themes

We support Light and Dark themes, which is a great way to let the user express their personality. We are updating the color tones to be softer on the eyes by avoiding pure white and black, which makes the colors much more delightful. Dark theme support is also a great to make your app more accessible and attractive for different types of users.



- The color palette of WinUI is being updated to feel lighter (use WinUI 2.6 or greater). If your apps have hardcoded custom colors, you may need to make updates to match the overall color theory, regardless of technology.
- If you are using UXTheme based Win32 surfaces, the Light theme will have rejuvenated controls (for example, rounded buttons). You should test your apps to validate that local styling does not override updated global defaults. (For Win32 apps, see Support Dark and Light themes in Win32 apps.)

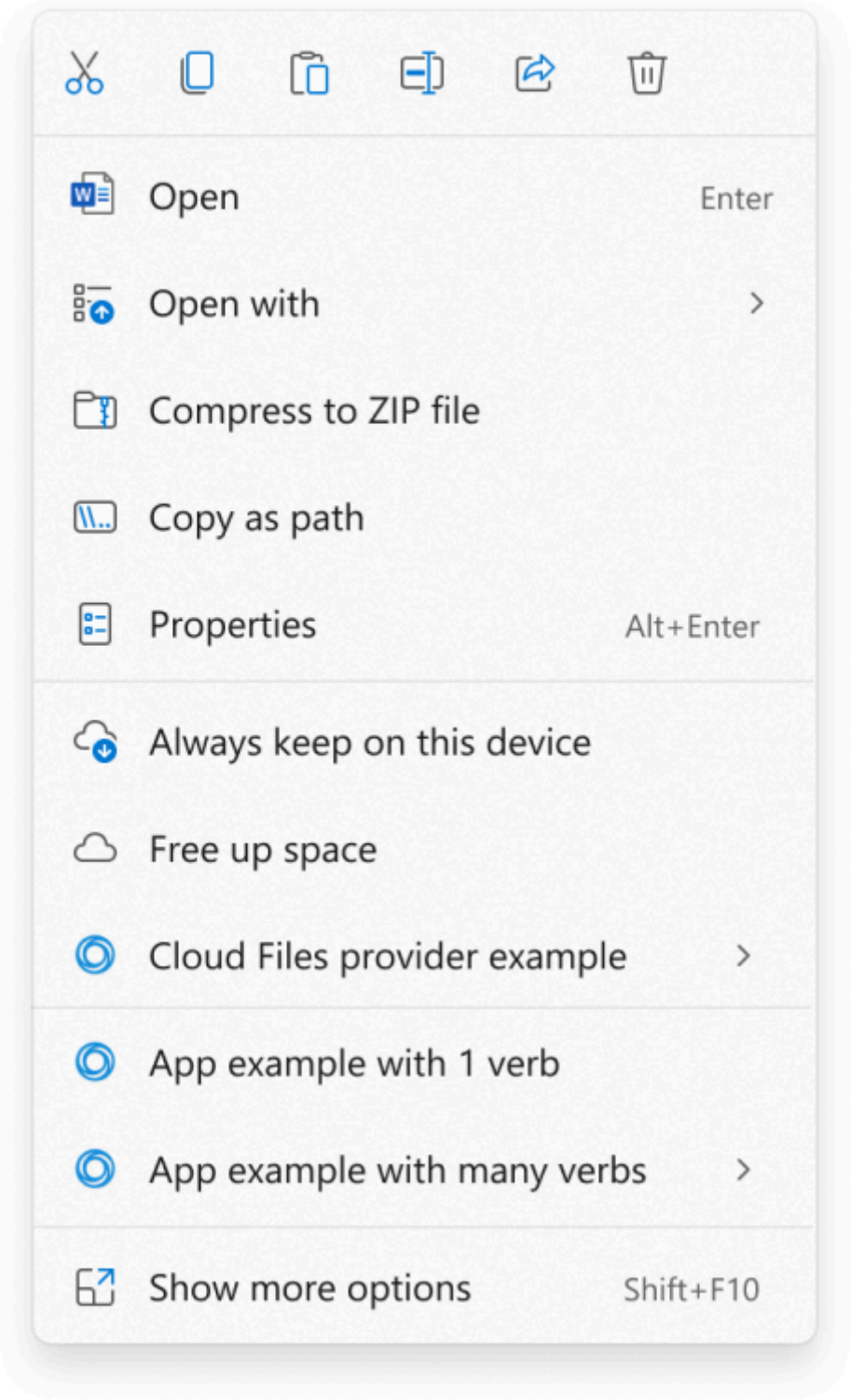
8. Optimize your app's context menu extensions and Share targets

Windows 11 refines the behavior of the contextual file operations in the right-click context menu of File Explorer and the Share dialog. If your app creates context menus or defines share targets, you may need to make some changes to ensure that these work well with Windows 11.

Context menus

For Windows 11, we improved the behavior of the context menu in File Explorer in several ways:

- Common commands, such as **Cut**, **Copy**, **Paste**, and **Delete**, have been moved to the top of the menu.
- **Open** and **Open with** are now grouped together.
- App extensions are grouped together below Shell verbs. Apps with more than one verb are grouped into a flyout with app attribution.
- Cloud files provider apps are placed next to the Shell commands to hydrate or dehydrate files.
- The older context menu from Windows 10 (along with lesser-used commands from the older context menu) is still available via the **Show more options** item at the bottom of the menu. **Shift** + **F10** or the keyboard menu key will also load the Windows 10 context menu.



If your app defines a context menu extension, the following requirements must be met for the extension to appear in the new Windows 11 context menu. Otherwise, your app's context menu extension will appear in the older context menu available via the **Show more options** item.

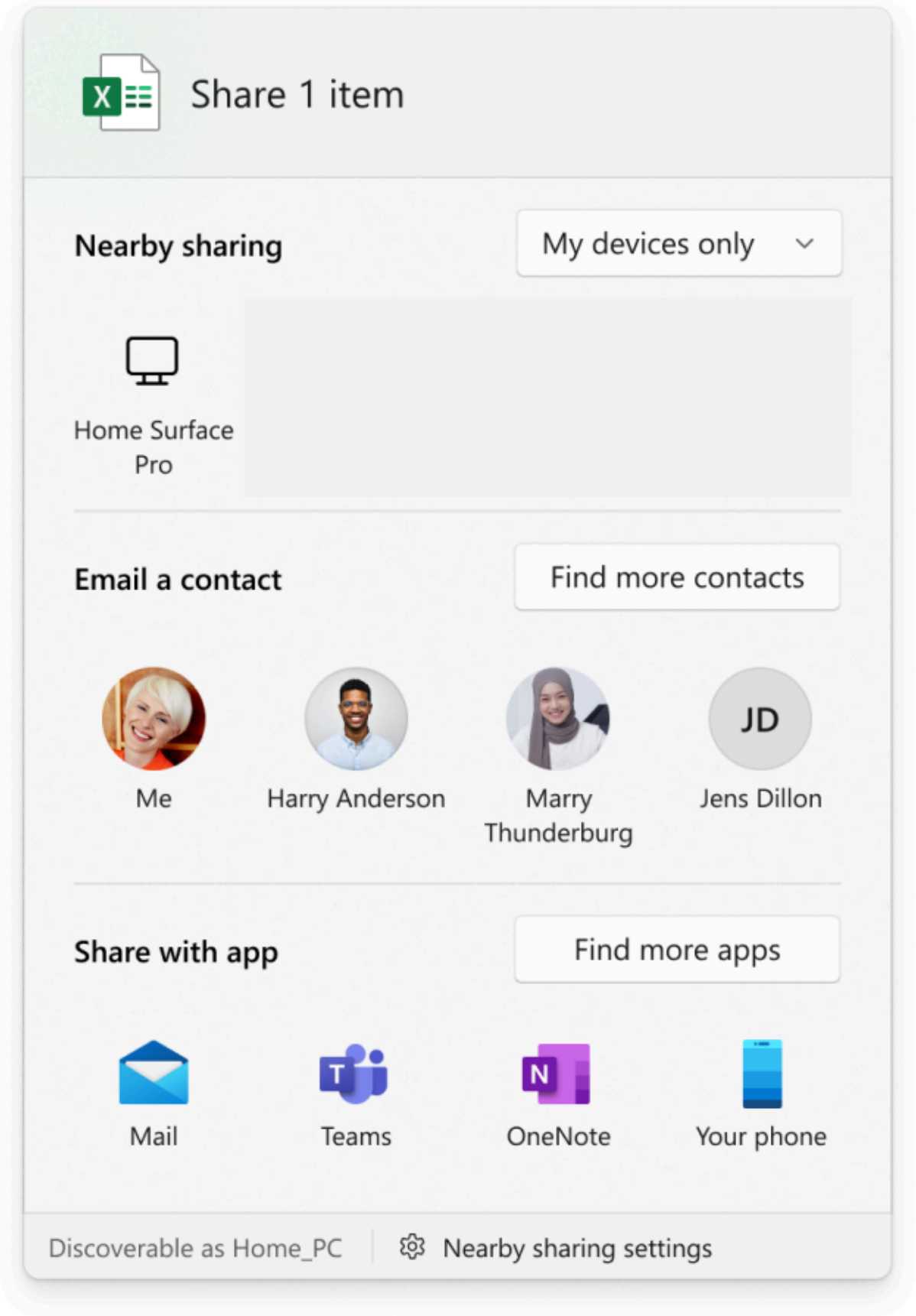
- Your context menu extension must be implemented by using the **IExplorerCommand** interface. Context menu extensions that implement **IContextMenu** will appear in the older context menu instead.

- Your app must be a *packaged app* so that it has package identity at runtime. See Features that require package identity for some options for packaging your app.

Share dialog

For Windows 11, we improved the behavior of the Share dialog in several ways.

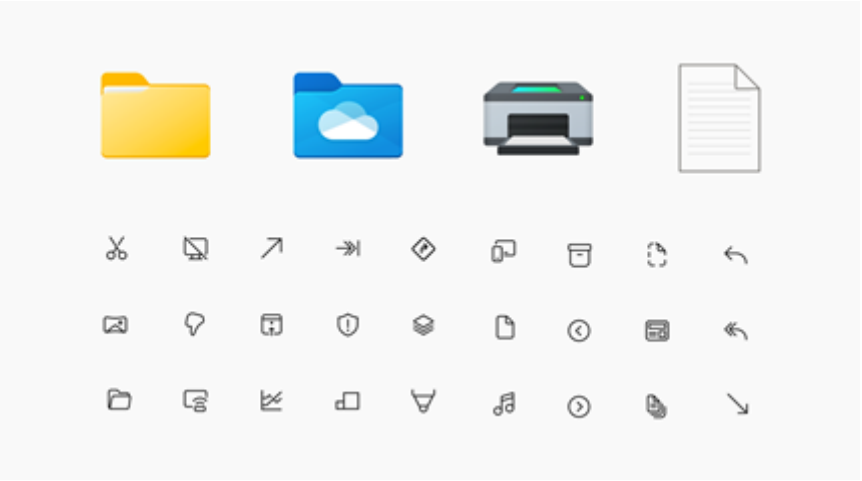
- Discoverability settings for nearby sharing are now at the top of the dialog and more settings are available at the bottom.
- All apps can now participate in the Share dialog as targets, including unpackaged desktop apps and PWAs that are installed through Microsoft Edge.
  - A previously unpackaged desktop app can participate as a target in the Share dialog if you package it with external location (see Grant package identity by packaging with external location). For sample code that demonstrates how to do that, see the SparsePackages sample app.
  - A PWA can participate in the Share dialog if it implements the Web Share Target API.



9. Use beautiful Iconography & Typography

We have updated icons and a new UI font called "Segoe UI Variable". We recommend all apps switch to using these new icons and font to be coherent on Windows 11. The new

font brings much softer geometry and makes the text much more legible.



- New icons called "Segoe Fluent Icons" are introduced for monoline icons. Controls in WinUI 2.6 and greater use the new icons and typography automatically.
- File type icons are updated. If your app is using icons in `imageres.dll` or `shell32.dll`, then icons will be updated automatically. Otherwise, a manual style update might be needed.
- App icons - Follow the latest design guidance for app icons used in places like launchers on Start and TaskBar.
- Animated icons - Lottie animation support was added to WinUI and we recommend using AnimatedIcon functionality to animate your icons in a meaningful way. Just as with other stylistic changes, you will need WinUI 2.6 or greater.
- Custom experiences written in XAML that specify `Segoe UI` in code, should instead specify `Segoe UI Variable`.

> **Note**
>
> When an app that uses the new font runs in Windows 10 or down-level, it will fallback to use the old font and degrade gracefully.

10. Make use of the innovative, secure features available in Windows

People run Windows across conventional devices as well as an increasingly diverse, modern range of devices. Devices today come not only with x86/x64-based, but also Arm-based, architectures; not only with mouse and keyboard but also touch screens, touchpads, and pens; with cameras, GPS, and sensors like gyroscopes; and with graphics and neural processing chipsets that enable not only amazing visuals but also hardware-

accelerated artificial intelligence (AI). Customers expect apps to take advantage of the hardware (that they have paid for!) and be cognizant of the device form factor to give them an appropriately optimized experience.

- Achieve AI powered productivity with Win ML - Introduction to Windows Machine Learning.
- Use AI models that power Windows Copilot Runtime on Copilot+ PCs and run locally - Windows Copilot Runtime overview.
- Utilize best practices for Notifications and content sharing - Notifications (Design basics).

Enhanced security features in Windows

Windows is built on a foundation of security and privacy, and Windows 11 is designed to be the most secure version of Windows yet, and we're committed to helping you build secure apps that take advantage of the latest security features in Windows.

- Protect your Windows apps and backend services with Windows Hello biometric sign-in - Windows Hello overview.
- Implement passkey sign-ins across online, enterprise, and government applications, and for payments - Intro to passkeys.
- Sign your apps with a digital certificate to ensure that Smart App Control can verify the integrity of your app - Introduction to code signing and Microsoft Trusted Root Program requirements.

11. Utilize the power of MSIX and Windows Store to package and distribute your application

Distribute your app wherever it makes sense for your business. Windows lets you distribute from your existing website, app management system, or the Windows Store.

- The Store lets you bring all of your Windows apps with no change, be it a native Windows app or a Progressive Web App (PWA). You can take your existing Windows app and list it on the store to drive app discovery. You choose whether you want to use the Store commerce and app distribution system, or use your existing systems. Either way, your app will be easily discoverable in the Windows Store.

  Learn more about Windows Store

- Make sure your app supports auto-updates, is efficient to install, and is manageable via MSIX.

  Keeping customers up to date with the latest version of your application is key to customer retention and appreciation. MSIX allows your app to seamlessly install and keep up to date without having to run a separate app update process. You get complete control over how this happens and on what schedule.

  What is MSIX?

Related content

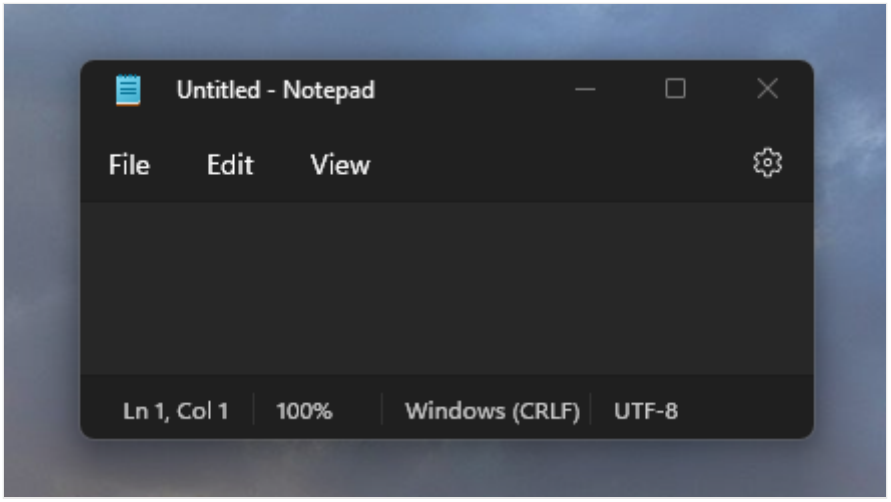
- Windows Dev Center
- What's cool in Windows 11

Apply rounded corners in desktop apps for Windows 11

Article • 08/21/2024

Rounded corners are the most immediately noticeable feature of Windows 11 Geometry. On Windows 11, the system automatically rounds top-level window corners for all inbox apps, including all UWP apps, and most other apps. However, some Win32 apps might not be rounded. This topic describes how to round your Win32 app's main window corners if the system does not round them automatically.

> **Note**
>
> By design, apps are not rounded when maximized, snapped, running in a Virtual Machine (VM), running on a Windows Virtual Desktop (WVD), or running as a Windows Defender Application Guard (WDAG) window.



Why isn't my app rounded?

If your app's main window doesn't receive automatic rounding, it's because you've customized your frame in a way that prevents it. Apps fall into three main categories from the perspective of the Desktop Window Manager (DWM):

1. Apps that are rounded by default.

   This includes apps that want a complete system-provided frame and caption-controls (min/max/close buttons), like Notepad. It also includes apps that provide enough information to the system so it can properly round them, such as setting

the WS_THICKFRAME and WS_CAPTION window styles or providing a 1-pixel non-client area border that the system can use to round the corners.

2. Apps that are not rounded by policy, but *can* be rounded.

   Apps in this category generally want to customize the majority of the window frame but still want the system-drawn border and shadow, such as Microsoft Office. If your app is not rounded by policy, it could be caused by one of the following things:

   - Lack of frame styles
   - Empty non-client area
   - Other customizations, such as extra non-child windows used for custom shadows

   Changing one of these things will break automatic rounding. Although we did try to round as many apps as possible with our system heuristics, there are some combinations of customizations that we can't predict so we provided a manual opt-in API for those cases. If you address these issues in your app or call the opt-in API, described in the following section, then it's possible for the system to round your app's window. Note, however, that the API is a hint to the system and does not guarantee rounding, depending on the customizations.

3. Apps that cannot ever be rounded, even if they call the opt-in API.

   These apps have no frame or borders, and typically have heavily customized UI. If your app does one of the following, it cannot be rounded:

   - Per-pixel alpha layering
   - Window regions

   For example, an app might use per-pixel alpha layering to draw transparent pixels around its main window to achieve a custom shadow effect, which makes the window no longer a rectangle and therefore the system cannot round it.

How to opt in to rounded corners

If your app is not rounded by policy, you can optionally use these APIs to let your app opt-in to rounded corners. You specify the corner rounding option you want for your app by passing a value of the **DWM_WINDOW_CORNER_PREFERENCE** enumeration (shown in the following table) to the **DwmSetWindowAttribute** function.

Expand table

Enum value	Description
DWMWCP_DEFAULT	Let the system decide whether or not to round window corners.
DWMWCP_DONOTROUND	Never round window corners.
DWMWCP_ROUND	Round the corners if appropriate.
DWMWCP_ROUNDSMALL	Round the corners if appropriate, with a small radius.

A pointer to the appropriate value from this enum is passed to the third parameter of **DwmSetWindowAttribute**. For the second parameter, which specifies which attribute you are setting, pass the **DWMWA_WINDOW_CORNER_PREFERENCE** value defined in the DWMWINDOWATTRIBUTE enumeration.

For C# apps

DwmSetWindowAttribute is a native Win32 API and is not exposed directly to .NET code. You'll need to use your language's implementation of P/Invoke to declare the function (C# code is given in the example below). All standard WinForms and WPF apps are rounded automatically, but if you customize your window frame or use a third party framework, you might need to opt-in to rounded corners. See the Examples section for further details.

Examples

The following examples show how you can call DwmSetWindowAttribute or DwmGetWindowAttribute to control your app's rounding experience if your app is not rounded by policy.

> ⓘ **Note**
>
> Error handling has been left out of these examples for brevity and clarity.

Example 1 - Rounding an app's main window in C# - WPF

This example shows how to call DwmSetWindowAttribute from C# by using the **[DllImport]** attribute. Note that this definition is specific to rounded corners; the DwmSetWindowAttribute function is designed to take different parameters depending on the flags provided, so this is not a general-purpose signature. The example also includes copies of the relevant enums from the dwmapi.h header file. Because the

Win32 API takes a pointer for the third parameter, make sure to use the *ref* keyword so you can pass the address of a variable when you call the function. You can do this in your MainWindow class in MainWindow.xaml.cs.

CSharp

```
using System.Runtime.InteropServices;
using System.Windows.Interop;

public partial class MainWindow : Window
{
    // The enum flag for DwmSetWindowAttribute's second parameter, which
tells the function what attribute to set.
    // Copied from dwmapi.h
    public enum DWMWINDOWATTRIBUTE
    {
        DWMWA_WINDOW_CORNER_PREFERENCE = 33
    }

    // The DWM_WINDOW_CORNER_PREFERENCE enum for DwmSetWindowAttribute's
third parameter, which tells the function
    // what value of the enum to set.
    // Copied from dwmapi.h
    public enum DWM_WINDOW_CORNER_PREFERENCE
    {
        DWMWCP_DEFAULT      = 0,
        DWMWCP_DONOTROUND   = 1,
        DWMWCP_ROUND        = 2,
        DWMWCP_ROUNDSMALL   = 3
    }

    // Import dwmapi.dll and define DwmSetWindowAttribute in C#
corresponding to the native function.
    [DllImport("dwmapi.dll", CharSet = CharSet.Unicode, PreserveSig =
false)]
    internal static extern void DwmSetWindowAttribute(IntPtr hwnd,
                                                       DWMWINDOWATTRIBUTE
attribute,
                                                       ref
DWM_WINDOW_CORNER_PREFERENCE pvAttribute,
                                                       uint cbAttribute);
    // ...
    // Various other definitions
    // ...
}
```

Next, in your MainWindow constructor, after the call to InitializeComponent, create a new instance of the WindowInteropHelper class to acquire a pointer to the underlying HWND (window handle). Make sure to use the EnsureHandle method to force the system to create an HWND for the window before it's shown, because normally the system only does so after exiting the constructor.

CSharp

```
public MainWindow()
{
    InitializeComponent();

    IntPtr hWnd = new WindowInteropHelper(GetWindow(this)).EnsureHandle();
    var attribute = DWMWINDOWATTRIBUTE.DWMWA_WINDOW_CORNER_PREFERENCE;
    var preference = DWM_WINDOW_CORNER_PREFERENCE.DWMWCP_ROUND;
    DwmSetWindowAttribute(hWnd, attribute, ref preference, sizeof(uint));

    // ...
    // Perform any other work necessary
    // ...
}
```

Example 2 - Rounding an app's main window in C# - WinForms

Like with WPF, for a WinForms app you'll first need to import dwmapi.dll and the DwmSetWindowAttribute function signature with P/Invoke. You can do this in your primary Form class.

Csharp

```
using System;
using System.Runtime.InteropServices;

public partial class Form1 : Form
{
    // The enum flag for DwmSetWindowAttribute's second parameter, which
tells the function what attribute to set.
    // Copied from dwmapi.h
    public enum DWMWINDOWATTRIBUTE
    {
        DWMWA_WINDOW_CORNER_PREFERENCE = 33
    }

    // The DWM_WINDOW_CORNER_PREFERENCE enum for DwmSetWindowAttribute's
third parameter, which tells the function
    // what value of the enum to set.
    // Copied from dwmapi.h
    public enum DWM_WINDOW_CORNER_PREFERENCE
    {
        DWMWCP_DEFAULT      = 0,
        DWMWCP_DONOTROUND   = 1,
        DWMWCP_ROUND        = 2,
        DWMWCP_ROUNDSMALL   = 3
    }
```

```
    // Import dwmapi.dll and define DwmSetWindowAttribute in C#
corresponding to the native function.
    [DllImport("dwmapi.dll", CharSet = CharSet.Unicode, PreserveSig =
false)]
    internal static extern void DwmSetWindowAttribute(IntPtr hwnd,
                                                       DWMWINDOWATTRIBUTE
attribute,
                                                       ref
DWM_WINDOW_CORNER_PREFERENCE pvAttribute,
                                                       uint cbAttribute);

    // ...
    // Various other definitions
    // ...
}
```

Calling DwmSetWindowAttribute is also the same as with a WPF app, but you don't have to use a helper class to get the HWND because it's simply a property of the Form. Call it from within your Form constructor, after the call to InitializeComponent.

CSharp

```
public Form1()
{
    InitializeComponent();

    var attribute = DWMWINDOWATTRIBUTE.DWMWA_WINDOW_CORNER_PREFERENCE;
    var preference = DWM_WINDOW_CORNER_PREFERENCE.DWMWCP_ROUND;
    DwmSetWindowAttribute(this.Handle, attribute, ref preference,
sizeof(uint));

    // ...
    // Perform any other work necessary
    // ...
}
```

Example 3 - Rounding an app's main window in C++

For a native C++ app, you can call DwmSetWindowAttribute in your message processing function after window creation to ask the system to round you.

C++

```
LRESULT ExampleWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message)
    {
    // ...
```

```
    // Handle various window messages...
    // ...

    case WM_CREATE:
        // ...
        // Perform app resource initialization after window creation
        // ...

        if(hWnd)
        {
            DWM_WINDOW_CORNER_PREFERENCE preference = DWMWCP_ROUND;
            DwmSetWindowAttribute(hWnd, DWMWA_WINDOW_CORNER_PREFERENCE,
&preference, sizeof(preference));
        }
        break;

    // ...
    // Handle various other window messages...
    // ...
    }

    return 0;
}
```

Example 4 – Rounding the corners of a menu with a small radius - C++

By default, menus are pop-up windows, which do not get rounded. If your app creates a custom menu and you want it to follow the rounding policy of other standard menus, you can call the API to let the system know that this window should be rounded, even though it doesn't appear to match the default rounding policy.

C++

```
HWND CreateCustomMenu()
{
    // Call an app-specific helper to make the window, using traditional
APIs.
    HWND hWnd = CreateMenuWindowHelper();

    if (hWnd)
    {
        // Make sure we round the window, using the small radius
        // because menus are auxiliary UI.
        DWM_WINDOW_CORNER_PREFERENCE preference = DWMWCP_ROUNDSMALL;
        DwmSetWindowAttribute(hWnd, DWMWA_WINDOW_CORNER_PREFERENCE,
&preference, sizeof(preference));
    }
```

```
    return hWnd;
}
```

Apply Mica in Win32 desktop apps for Windows 11

Article • 08/21/2024

Mica is an opaque material that incorporates the user's theme and desktop wallpaper to create a highly personalized appearance. As the user moves the window across the screen, the Mica material dynamically adapts to create a rich visualization using the wallpaper underneath the application. In addition, the material helps users focus on the current task by falling back to a neutral color when the app is inactive.

This article describes how to apply Mica as the base layer of your Win32 app, prioritizing application and visibility in the title bar area. For more information about app layering with Mica, see Mica material.

Prerequisites

To apply Mica to a Win32 app for Windows 11, you need to use the Windows App SDK. You'll need the following:

- Install the latest Windows App SDK Visual Studio Extension or Microsoft.WindowsAppSDK NuGet package. See Downloads for the Windows App SDK.
- For unpackaged apps, reference the Windows App SDK, install WinRT, and have a matching Windows App Runtime Redistributable (Microsoft.WindowsAppRuntime.Redist) installed. See Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged.

How to use Mica in Win32 Apps

To use mica in your app, you use the MicaController class. This class manages both the rendering of the system backdrop material as well as the handling of system policy for the mica material.

The MicaController reacts to the system Light and Dark themes by default. To override this behavior, you can pass the following properties to the MicaController:

- FallbackColor
- LuminosityOpacity
- TintColor

- TintOpacity

> **Tip**
>
> The code in this section is taken from the **Windows App SDK Win32 Mica sample on GitHub**. See the GitHub repo for the full code. These examples use **C++/WinRT**.

To enable Mica, you need a reference to the Windows App SDK, a Compositor, and a DispatcherQueue.

This example shows how to do the following to set up an unpackaged app:

- Initialize WinRT.
- Reference the WindowsAppSDK from an unpackaged app.
  - See Use the Windows App SDK runtime for apps packaged with external location or unpackaged for more info.
  - See `WindowsAppSDKBootstrapperContext` in Utilities.h for example code.
- Register the window class.
- Create the Mica dispatcher queue controller
- Initialize a WinRT compositor.

> From WinMain.cpp

C++

```
int __stdcall WinMain(_In_ HINSTANCE, _In_opt_ HINSTANCE,  _In_ PSTR, _In_
int)
{
    // Initialize WinRt Instance
    winrt::init_apartment();

    // Enable referencing the WindowsAppSDK from an unpackaged app.
    Utilities::WindowsAppSDKBootstrapperContext sdkContext;

    // Register Window class before making the window.
    MicaWindow::RegisterWindowClass();

    // Mica requires a compositor, which also requires a dispatcher queue.
    auto controller =
Utilities::CreateDispatcherQueueControllerForCurrentThread();
    auto compositor = winrt::Compositor();

    // Create your window...
    ...
}
```

> From MicaWindow.cpp

```
C++
```

```
void MicaWindow::RegisterWindowClass()
{
    auto instance = winrt::check_pointer(GetModuleHandleW(nullptr));
    WNDCLASSEX wcex = { sizeof(wcex) };
    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = WndProc;
    wcex.hInstance = instance;
    wcex.hIcon = LoadIconW(instance, IDI_APPLICATION);
    wcex.hCursor = LoadCursorW(nullptr, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszClassName = ClassName.c_str();
    wcex.hIconSm = LoadIconW(wcex.hInstance, IDI_APPLICATION);
    winrt::check_bool(RegisterClassExW(&wcex)); // check if the window class
was registered successfully
}
```

The `winrt::init_apartment` method is multi-threaded by default. If your app requires a single-thread, like the WebView2 Sample, you can easily set the type.

```
C++
```

```
winrt::init_apartment(winrt::apartment_type::single_threaded);
```

Now you can use the `CreateWindowEx()` function to create a window. Then, you have to create a window target and set it as the root to specify which layer to apply Mica to. Finally, assert that Mica is supported by the window and target.

The Win32 Mica sample creates the DesktopWindow and MicaWindow classes to do this work. These classes define: `ClassName`, `windowTitle`, `m_target`, `m_micaController`, and `m_isMicaSupported`.

> From WinMain.cpp

```
C++
```

```
// Mica window is inherited from the MicaWindow class, which is an extension
of the DesktopWindow Class.
// Here, we initialize the main window and set the title.
   auto window = MicaWindow(compositor, L"Hello, Mica!");
```

> From MicaWindow.cpp

C++

```
// Create the main window and enable Mica.
MicaWindow::MicaWindow(const winrt::Compositor& compositor, const
std::wstring& windowTitle)
{
    auto instance = winrt::check_pointer(GetModuleHandleW(nullptr));
    WINRT_ASSERT(!m_window); // check that window is not initialized
    WINRT_VERIFY(
        // Window Properties
        CreateWindowExW(
            WS_EX_COMPOSITED,
            ClassName.c_str(), // declared in MicaWindow.h and defined above
            windowTitle.c_str(),
            WS_OVERLAPPEDWINDOW,
            CW_USEDEFAULT,
            CW_USEDEFAULT,
            800, 600,
            nullptr,
            nullptr,
            instance,
            this
        ));

    // Check that the window was created successfully.
    WINRT_ASSERT(m_window);

    ShowWindow(m_window, SW_SHOWDEFAULT);
    UpdateWindow(m_window);

    // The Mica controller needs to set a target with a root to recognize
the visual base layer.
    m_target = CreateWindowTarget(compositor);

    // Need to set a root before we can enable Mica.
    m_target.Root(compositor.CreateContainerVisual());

    m_micaController = winrt::MicaController();
    m_isMicaSupported =
m_micaController.SetTarget(winrt::Microsoft::UI::WindowId{
reinterpret_cast<uint64_t>(m_window) }, m_target);
}
```

How to use Mica in Win32 WebView2 apps

The fundamental principles of applying Mica are consistent across most Win32 applications. The process for WebView2 follows the basic steps from the Win32 instructions shown previously. However, in this case you will need to specify a single threaded process from WinRT's `init_apartment` feature.

> **Tip**
>
> The code in this section is taken from the **Windows App SDK WebView2 Mica sample on GitHub**. See the GitHub repo for the full code.

To get started, set up the required apartment, controller, compositor, target, and root. By default, the WinRT `init_apartment` function is multi-threaded, but WebView2 is inherently single-threaded. To set `init_apartment` as a single thread, pass the `winrt::apartment_type::single_threaded` parameter. In the Mica WebView2 Sample, we simplify the syntax by creating a separate class for web view functions, referenced in the following code.

> From Main.cpp

C++

```
int __stdcall WinMain(_In_ HINSTANCE, _In_opt_ HINSTANCE, _In_ PSTR, _In_
int)
{
    winrt::init_apartment(winrt::apartment_type::single_threaded);
    // Enable referencing the WindowsAppSDK from an unpackaged app.
    // Remember to have a matching Microsoft.WindowsAppRuntime.Redist
installed.
    // https://learn.microsoft.com/windows/apps/windows-app-sdk/deploy-
unpackaged-apps
    Utilities::WindowsAppSDKBootstrapperContext sdkContext;
    CompositionWindow::RegisterWindowClass();
    // A dispatcher queue is required to be able to create a compositor.
    auto controller =
Utilities::CreateDispatcherQueueControllerForCurrentThread();
    auto compositor = winrt::Compositor();
    auto window = WebView2Window(compositor, L"Hello, WebView2!");

    ...
}
```

For a full demonstration of the **WebView2Window class** and its integration with Mica, see the Windows App SDK WebView2 Mica sample on GitHub. Note how the CompositionWindow and WebView2Window classes handle messages, initialize the web view environment, and delete the window controller once the window is closed.

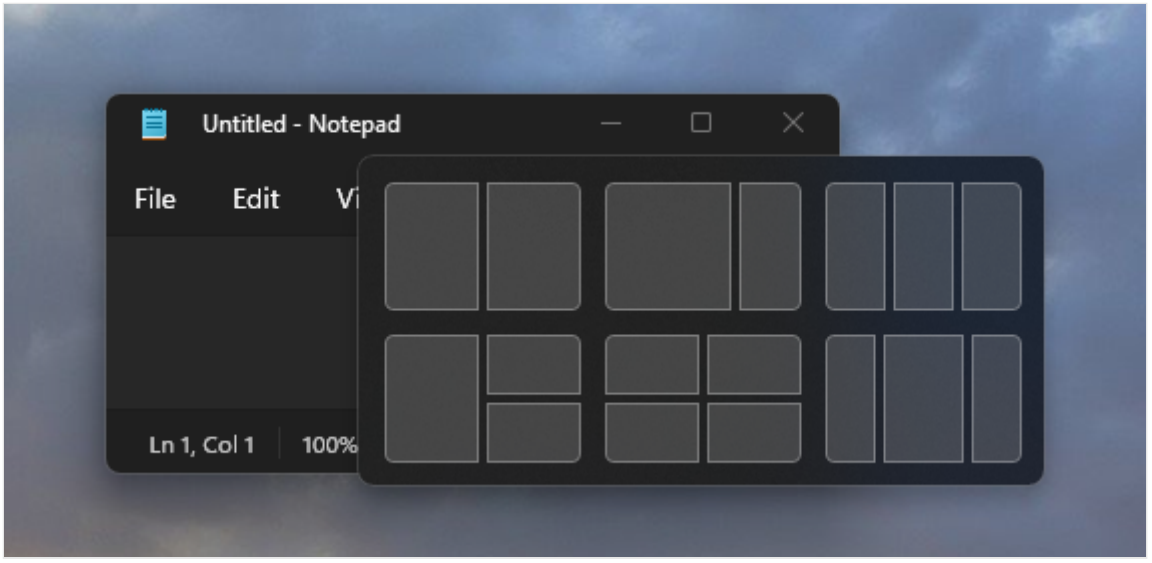
Related articles

Materials, Layering and Elevation, Windows App SDK Mica samples

Support snap layouts for desktop apps on Windows 11

Article • 08/21/2024

Snap layouts are a new Windows 11 feature to help introduce users to the power of window snapping. Snap layouts are easily accessible by hovering the mouse over a window's maximize button or pressing Win + Z. After invoking the menu that shows the available layouts, users can click on a zone in a layout to snap a window to that particular zone and then use Snap Assist to finish building an entire layout of windows. Snap layouts are tailored to the current screen size and orientation, including support for three side-by-side windows on large landscape screens and top/bottom stacked windows on portrait screens.



If the app's window has the maximize caption button available, the system will automatically show snap layouts when a user hovers the mouse over the window's maximize button. Snap layouts will appear automatically for most apps, but some desktop apps may not show snap layouts. This topic describes how to make sure your app shows the menu with snap layouts if the system does not show it automatically.

Why doesn't my app show the snap layouts menu?

If your app's main window has the maximize caption button available but does not show snap layouts, it may be because you've customized your caption buttons or title bar in a way that prevents it.

How do I fix it?

If you have a custom title bar, then you can:

- Use the Windows App SDK windowing APIs (see Manage app windows) and have the platform draw and implement the caption buttons for you.
- For Win32 apps, make sure you are responding appropriately to WM_NCHITTEST (with a return value of `HTMAXBUTTON` for the maximize/restore button).

  C++

  ```
  LRESULT CALLBACK TestWndProc(HWND window, UINT msg, WPARAM wParam,
  LPARAM lParam)
  {
      switch (msg)
      {
            case WM_NCHITTEST:
          {
              // Get the point in screen coordinates.
              // GET_X_LPARAM and GET_Y_LPARAM are defined in windowsx.h
              POINT point = { GET_X_LPARAM(lParam), GET_Y_LPARAM(lParam)
  };
              // Map the point to client coordinates.
              ::MapWindowPoints(nullptr, window, &point, 1);
              // If the point is in your maximize button then return
  HTMAXBUTTON
              if (::PtInRect(&m_maximizeButtonRect, point))
              {
                  return HTMAXBUTTON;
              }
          }
          break;
      }
      return ::DefWindowProcW(window, msg, wParam, lParam);
  }
  ```

- If your app uses Electron, update to the v13 stable release of Electron to enable snap layouts.

What if my app's window shows snap layouts but isn't snapping properly?

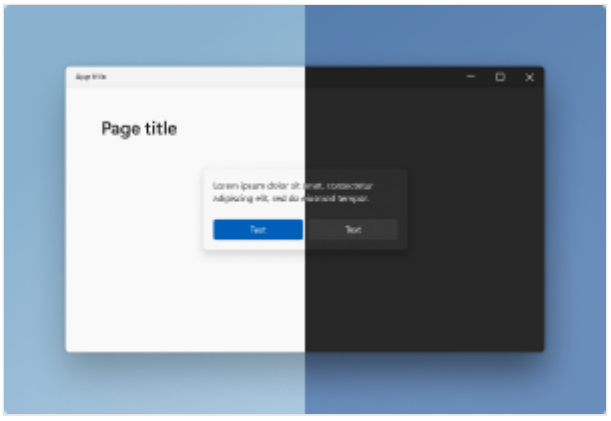
If your app can invoke the menu with snap layouts but isn't able to snap properly to the zone sizes, it's likely that your app's minimum window size is too large for the window to fit in the selected zone.

Your app should support a minimum width of at *most* 500 effective pixels (epx) to support snap layouts across the most common screen sizes. However, we recommend that you support an even smaller minimum width (330 epx or less) so that it's compatible with a larger set of devices and snap layouts.

Support Dark and Light themes in Win32 apps

Article • 08/21/2024

Windows supports *Light* and *Dark* themes as a personalization option in Windows settings. Windows uses Light mode by default, but users can choose Dark mode, which changes much of the UI to a dark color. Users might prefer this setting because it's easier on the eyes in lower-light environments, or they might simply prefer a darker interface in general. Also, darker UI colors can reduce the battery usage on some types of computer displays, such as OLED screens.

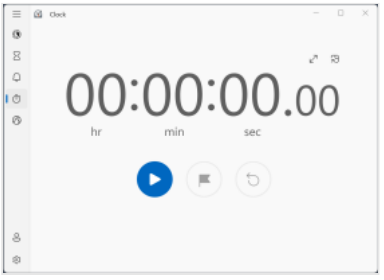


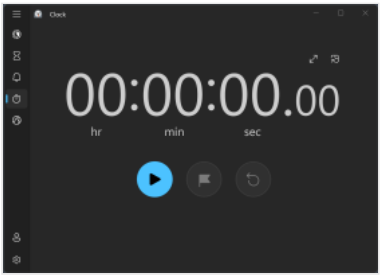
We are working hard to broaden support for Dark mode without breaking existing applications, and to that end we're providing technical guidance for updating a Win32 desktop Windows app to support both Light and Dark modes.

Dark mode vs. Light mode

The Color Mode in settings (which includes Light and Dark modes) is a setting that defines the overall *foreground* and *background* colors for the operating system and apps.

Expand table

Mode	Description	Example
Light	A light background with a contrasting dark foreground. In Light Mode, you will generally see black or dark text on white or light backgrounds.	

Mode	Description	Example
Dark	A dark background with a contrasting light foreground. In Dark mode, you will generally see white or light text on black or dark backgrounds.	

> **Note**
>
> The reason we use "*black or dark*" and "*white or light*" is because there are additional colors such as the Accent color that can tint various foreground and background colors. So you might in fact see light blue text on a dark blue background in some parts of the UI, and that would still be considered acceptable Dark mode UI.

Due to the wide diversity of UI in different apps, the color mode, and foreground and background colors are meant as more of a directional guideline than a hard rule:

- Foreground elements, highlights, and text should be closer to the foreground color than the background color.
- Large, solid background areas and text backgrounds should generally be closer to the background color than the foreground color.

In practice, this means that in Dark mode, most of the UI will be dark, and in Light mode most of the UI will be light. The concept of a background in Windows is the large area of colors in an app, or the page color. The concept of a foreground in Windows is the text color.

> **Tip**
>
> If you find it confusing that the *foreground* color is light in Dark mode and dark in Light mode, it may help to think of the foreground color as "the default text color".

Enable support for switching color modes

There are many approaches to implementing Dark mode support in an application. Some apps contain two sets of UIs (one with a light color and one with a dark color). Some Windows UI frameworks, such as WinUI 3, automatically detect a system's theme

and adjust the UI to follow the system theme. To fully support Dark mode, the entirety of an app's surface must follow the dark theme.

There are two main things you can do in your Win32 app to support both Light and Dark themes.

- **Know when Dark mode is enabled**

  Knowing when Dark mode is enabled in the system settings can help you know when to switch your app UI to a Dark mode-themed UI.

- **Enable a Dark mode title bar for Win32 applications**

  Not all Win32 applications support Dark mode, so Windows gives Win32 apps a light title bar by default. If you are prepared to support Dark mode, you can ask Windows to draw the dark title bar instead when Dark mode is enabled.

> **Note**
>
> This article provides examples of ways to detect system theme changes, and request a light or dark title bar for your Win32 application's window. It does not cover specifics of how to repaint and render your app UI using a Dark mode color set.

Know when Dark mode is enabled

The first step is to keep track of the color mode setting itself. This will let you adjust your application's painting and rendering code to use a Dark mode color set. Doing this requires the app to read the color setting at startup and to know when the color setting changes during an app session.

To do this in a Win32 application, use Windows::UI::Color and detect if a color can be classified as *light* or *dark*. To use `Windows::UI::Color`, you need to import (in `pch.h`) the `Windows.UI.ViewManagement` header from winrt.

```
C++
```

```
#include <winrt/Windows.UI.ViewManagement.h>
```

Also include that namespace in `main.cpp`.

```
C++
```

```
using namespace Windows::UI::ViewManagement;
```

In `main.cpp`, use this function to detect if a color can be classified as *light*.

C++

```
inline bool IsColorLight(Windows::UI::Color& clr)
{
    return (((5 * clr.G) + (2 * clr.R) + clr.B) > (8 * 128));
}
```

This function performs a quick calculation of the *perceived brightness* of a color, and takes into consideration ways that different channels in an RGB color value contribute to how bright it looks to the human eye. It uses all-integer math for speed on typical CPUs.

> **Note**
>
> This is not a model for real analysis of color brightness. It is good for quick calculations that require you to determine if a color can be classified as *light* or *dark*. Theme colors can often be light but not pure white, or dark but not pure black.

Now that you have a function to check whether a color is light, you can use that function to detect if Dark mode is enabled.

Dark mode is defined as a dark background with a contrasting light foreground. Since `IsColorLight` checks if a color is considered light, you can use that function to see if the foreground is light. If the foreground is light, then Dark mode is enabled.

To do this, you need to get the UI color type of the foreground from the system settings. Use this code in `main.cpp`.

C++

```
auto settings = UISettings();

auto foreground = settings.GetColorValue(UIColorType::Foreground);
```

UISettings gets all the settings of the UI including color. Call UISettings.GetColorValue(UIColorType::Foreground) to get the foreground color value from the UI settings.

Now you can run a check to see if the foreground is considered light (in `main.cpp`).

C++

```
bool isDarkMode = static_cast<bool>(IsColorLight(foreground));

wprintf(L"\nisDarkMode: %u\n", isDarkMode);
```

- If the foreground is light, then `isDarkMode` will evaluate to 1 (`true`) meaning Dark mode is enabled.
- If the foreground is dark, then `isDarkMode` will evaluate to 0 (`false`) meaning Dark mode is not enabled.

To automatically track when the Dark mode setting changes during an app session, you can wrap your checks like this.

C++

```
auto revoker = settings.ColorValuesChanged([settings](auto&&...)
{
    auto foregroundRevoker =
settings.GetColorValue(UIColorType::Foreground);
    bool isDarkModeRevoker = static_cast<bool>
(IsColorLight(foregroundRevoker));
    wprintf(L"isDarkModeRevoker: %d\n", isDarkModeRevoker);
});
```

Your full code should look like this.

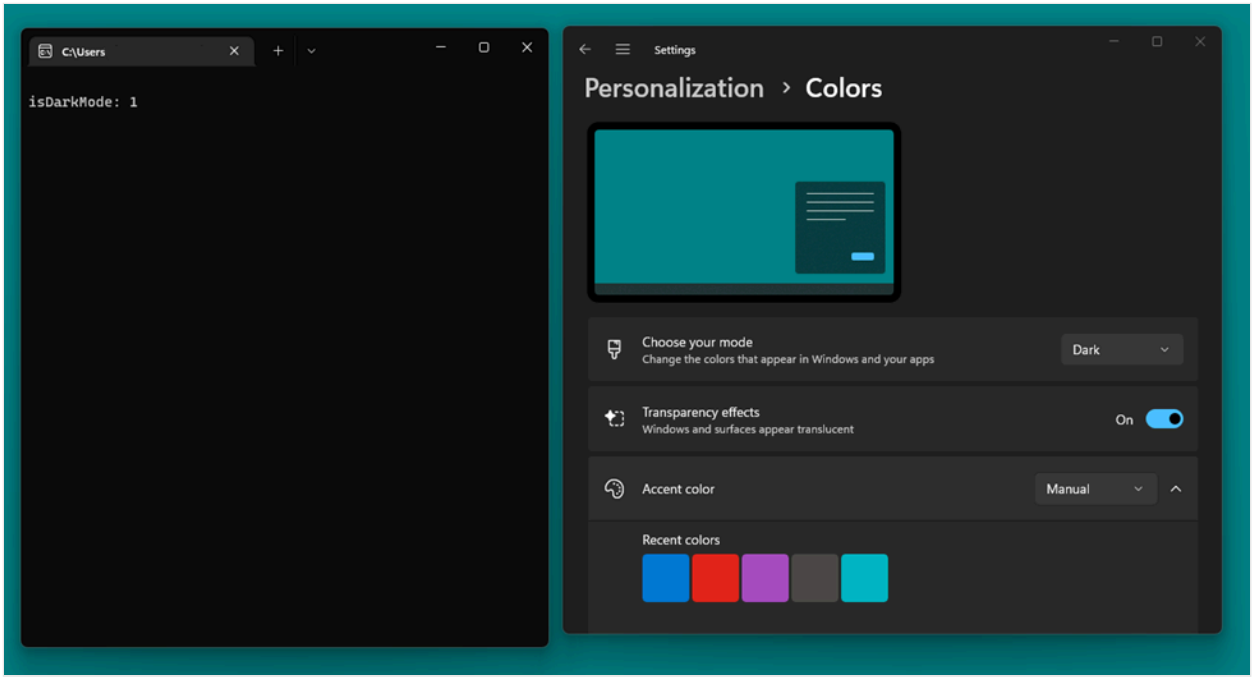
C++

```
inline bool IsColorLight(Windows::UI::Color& clr)
{
    return (((5 * clr.G) + (2 * clr.R) + clr.B) > (8 * 128));
}

int main()
{
    init_apartment();

    auto settings = UISettings();
    auto foreground = settings.GetColorValue(UIColorType::Foreground);

    bool isDarkMode = static_cast<bool>(IsColorLight(foreground));
    wprintf(L"\nisDarkMode: %u\n", isDarkMode);

    auto revoker = settings.ColorValuesChanged([settings](auto&&...)
        {
```

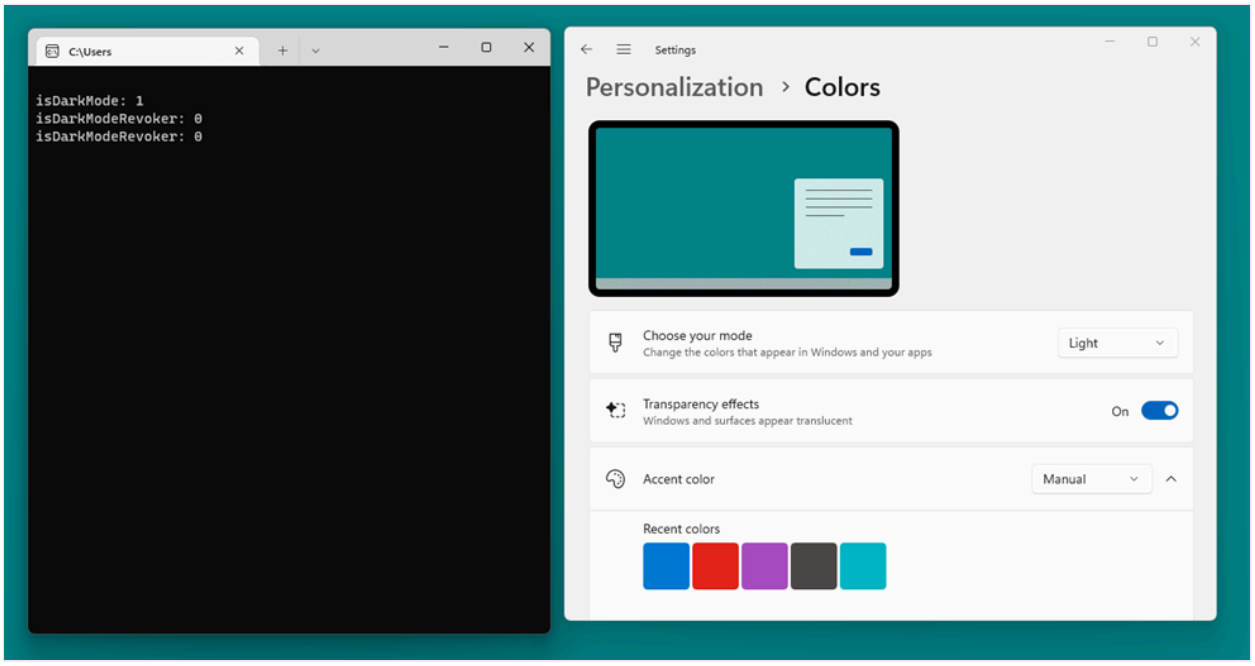
```
            auto foregroundRevoker =
settings.GetColorValue(UIColorType::Foreground);
            bool isDarkModeRevoker = static_cast<bool>
(IsColorLight(foregroundRevoker));
            wprintf(L"isDarkModeRevoker: %d\n", isDarkModeRevoker);
        });

    static bool s_go = true;
    while (s_go)
    {
        Sleep(50);
    }
}
```

When this code is run:

If Dark mode is enabled, `isDarkMode` will evaluate to 1.



Changing the setting from Dark mode to Light mode will make `isDarkModeRevoker` evaluate to 0.



Enable a Dark mode title bar for Win32 applications

Windows doesn't know if an application can support Dark mode, so it assumes that it can't for backwards compatibility reasons. Some Windows development frameworks, such as Windows App SDK, support Dark mode natively and change certain UI elements without any additional code. Win32 apps often don't support Dark mode, so Windows gives Win32 apps a light title bar by default.

However, for any app that uses the standard Windows title bar, you can enable the dark version of the title bar when the system is in Dark mode. To enable the dark title bar, call a Desktop Windows Manager (DWM) function called DwmSetWindowAttribute on your top-level window, using the window attribute DWMWA_USE_IMMERSIVE_DARK_MODE. (DWM renders attributes for a window.)

The following examples assume you have a window with with a standard title bar, like the one created by this code.

C++

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // Store instance handle in our global variable

   HWND hWnd = CreateWindowW(szWindowClass, szTitle,
WS_OVERLAPPEDWINDOW,CW_USEDEFAULT, 0,
      CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

   if (!hWnd)
```

```
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}
```

First, you need to import the DWM API, like this.

C++

```
#include <dwmapi.h>
```

Then, define the `DWMWA_USE_IMMERSIVE_DARK_MODE` macros above your `InitInstance` function.

C++

```
#ifndef DWMWA_USE_IMMERSIVE_DARK_MODE
#define DWMWA_USE_IMMERSIVE_DARK_MODE 20
#endif

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
…
```

Finally, you can use the DWM API to set the title bar to use a dark color. Here, you create a `BOOL` called `value` and set it to `TRUE`. This `BOOL` is used to trigger this Windows attribute setting. Then, you use `DwmSetWindowAttribute` to change the window attribute to use Dark mode colors.

C++

```
BOOL value = TRUE;
::DwmSetWindowAttribute(hWnd, DWMWA_USE_IMMERSIVE_DARK_MODE, &value,
sizeof(value));
```

Here's more explanation of what this call does.

The syntax block for DwmSetWindowAttribute looks like this.

C++

```
HRESULT DwmSetWindowAttribute(
       HWND    hwnd,
       DWORD   dwAttribute,
  [in] LPCVOID pvAttribute,
       DWORD   cbAttribute
);
```

After passing `hWnd` (the handle to the window you want to change) as your first parameter, you need to pass in `DWMWA_USE_IMMERSIVE_DARK_MODE` as the `dwAttribute` parameter. This is a constant in the DWM API that lets the Windows frame be drawn in Dark mode colors when the Dark mode system setting is enabled. If you switch to Light mode, you will have to change `DWMWA_USE_IMMERSIVE_DARK_MODE` from 20 to 0 for the title bar to be drawn in light mode colors.

The `pvAttribute` parameter points to a value of type `BOOL` (which is why you made the `BOOL` value earlier). You need `pvAttribute` to be `TRUE` to honor Dark mode for the window. If `pvAttribute` is `FALSE`, the window will use Light Mode.

Lastly, `cbAttribute` needs to have the size of the attribute being set in `pvAttribute`. To do easily do this, we pass in `sizeof(value)`.

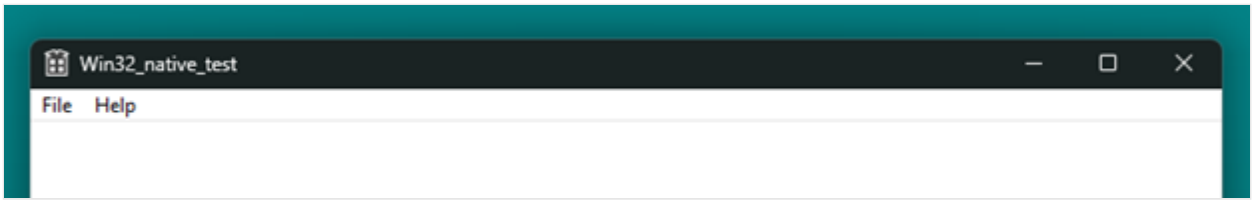
Your code to draw a dark windows title bar should look like this.

C++

```
#ifndef DWMWA_USE_IMMERSIVE_DARK_MODE
#define DWMWA_USE_IMMERSIVE_DARK_MODE 20
#endif


BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // Store instance handle in our global variable

   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance,
nullptr);

   BOOL value = TRUE;
   ::DwmSetWindowAttribute(hWnd, DWMWA_USE_IMMERSIVE_DARK_MODE, &value,
sizeof(value));

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
```

```
    UpdateWindow(hWnd);

    return TRUE;
}
```

When this code is run, the app title bar should be dark:



See also

- Make your app great on Windows 11
- Accessible text requirements
- Desktop Windows Manager (DWM)

Call Windows Runtime APIs in desktop apps

Article • 10/05/2022

This topic describes how to set up your desktop app projects to use Windows Runtime (WinRT) APIs provided by the Windows OS, and to add modern Windows 11 and Windows 10 experiences to your desktop apps.

Some Windows Runtime (WinRT) APIs are not supported in desktop apps. For more information, see Windows Runtime APIs not supported in desktop apps.

Modify a .NET project to use Windows Runtime APIs

There are several options for .NET projects:

- Starting in .NET 6, you can specify the Target Framework Moniker (TFM) in your project file to access WinRT APIs. This option is supported in projects that target Windows 10, version 1809 or later.
- For earlier versions of .NET, you can install the Microsoft.Windows.SDK.Contracts NuGet package to add all necessary references to your project. This option is supported in projects that target Windows 10, version 1803 or later.
- If your project multi-targets .NET 6 (or later) and earlier versions of .NET, then you can configure the project file to use both options.

.NET 6 and later: Use the Target Framework Moniker option

This option is supported only in projects that use .NET 6 (or later) and target Windows 10, version 1809 or a later OS release. By specifying a Windows OS version-specific TFM in the project file, a reference is added to the appropriate Windows SDK targeting package. For more background info about this scenario, see the blog post Calling Windows APIs in .NET.

1. With your project open in Visual Studio, right-click your project in **Solution Explorer** and choose **Edit Project File**. Your project file will look similar to this.

> **Note**

The example below shows an **OutputType** of *WinExe*, which specifies a Windows GUI executable (and prevents a console window from opening when the app runs). If your app doesn't have a GUI, then your **OutputType** will have a different value. You can call WinRT APIs from Windows GUI apps, console apps, and libraries. Also, your value for **TargetFramework** might not exactly match the example below.

XML

```xml
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net8.0</TargetFramework>
  </PropertyGroup>
</Project>
```

2. Leaving all other settings as they are, replace the value of the **TargetFramework** element with one of the following strings:

   - **net8.0-windows10.0.17763.0**: If your app targets Windows 10, version 1809.
   - **net8.0-windows10.0.18362.0**: If your app targets Windows 10, version 1903.
   - **net8.0-windows10.0.19041.0**: If your app targets Windows 10, version 2004.
   - **net8.0-windows10.0.22000.0**: If your app targets initial release of Windows 11.
   - **net8.0-windows10.0.22621.0**: If your app targets Windows 11, version 22H2.
   - **net8.0-windows10.0.26100.0**: If your app targets Windows 11, version 24H2.

   For example, the following element is for a project that targets Windows 10, version 2004.

   XML

   ```xml
   <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
   ```

   In later versions of .NET, you can replace the value with the relevant version, for example **net8.0-windows10.0.19041.0**.

3. Save your changes and close the project file.

WinRT APIs not supported in .NET 6 or later

In .NET 6 and later, there are several Windows Runtime (WinRT) APIs in the **Windows.UI** namespace that aren't supported. For the APIs listed below, equivalent versions of the

APIs exist in the WinUI (**Microsoft.UI**) namespace (for example, Microsoft.UI.Text). The following WinRT APIs are *not* supported on .NET 6 and later:

- Windows.UI.Colors class
- Windows.UI.ColorHelper class
- Windows.UI.Text (all classes in this namespace **except** for **Windows.UI.Text.FontStretch**, **Windows.UI.Text.FontStyle**, **Windows.UI.Text.FontWeight**, **Windows.UI.Text.UnderlineType**, and all classes under the **Windows.UI.Text.Core** namespace)
- Windows.UI.Xaml (all classes in this namespace)

Supporting multiple Windows OS versions

The Windows OS version-specific **TargetFramework** property determines the version of the Windows SDK that your app is compiled with. This property determines the set of accessible APIs at build time, and provides default values for both **TargetPlatformVersion** and **TargetPlatformMinVersion** (if not explicitly set). The **TargetPlatformVersion** property doesn't need to be explicitly defined in the project file, since it's automatically set by the **TargetFramework** OS version.

The **TargetPlatformMinVersion** can be overridden to be less than the **TargetPlatformVersion** (determined by the version in the **TargetFramework** property). This permits an app to run on earlier OS versions. For example, you can set the following in your project file to support your app downlevel to Windows 10, version 1809.

XML

```
<Project Sdk="Microsoft.NET.Sdk">
 <PropertyGroup>
   <OutputType>WinExe</OutputType>
   <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
   <TargetPlatformMinVersion>10.0.17763.0</TargetPlatformMinVersion>
 </PropertyGroup>
</Project>
```

Note that setting the **TargetPlatformMinVersion** to a version below the **TargetPlatformVersion** creates the potential for calling unavailable APIs. When calling WinRT APIs that are not available on all supported OS versions, we recommend guarding these calls with **ApiInformation** checks. For more information, see Version adaptive apps.

Earlier versions of .NET: Install the Microsoft.Windows.SDK.Contracts NuGet package

Use this option if your app uses .NET Core 3.x or .NET Framework. This option is supported in projects that target Windows 10, version 1803 or later.

1. Make sure package references are enabled:
   a. In Visual Studio, click **Tools -> NuGet Package Manager -> Package Manager Settings**.
   b. Make sure **PackageReference** is selected for **Default package management format**.

2. With your project open in Visual Studio, right-click your project in **Solution Explorer** and choose **Manage NuGet Packages**.

3. In the **NuGet Package Manager** window, select the **Browse** tab and search for `Microsoft.Windows.SDK.Contracts`.

4. After the `Microsoft.Windows.SDK.Contracts` package is found, in the right pane of the **NuGet Package Manager** window select the **Version** of the package you want to install based on the version of Windows 10 you want to target:

   - **10.0.19041.xxxx**: Choose this for Windows 10, version 2004.
   - **10.0.18362.xxxx**: Choose this for Windows 10, version 1903.
   - **10.0.17763.xxxx**: Choose this for Windows 10, version 1809.
   - **10.0.17134.xxxx**: Choose this for Windows 10, version 1803.

5. Click **Install**.

Configure projects that multi-target different versions of .NET

If your project multi-targets .NET 6 (or later) and earlier versions (including .NET Core 3.x and .NET Framework), then you can configure the project file to use the Target Framework Moniker (TFM) to automatically pull in the WinRT API references for .NET 6 (or later), and use the `Microsoft.Windows.SDK.Contracts` NuGet package for earlier versions.

1. With your project open in Visual Studio, right-click your project in **Solution Explorer** and choose **Edit Project File**. The following example demonstrates a project file for an app that uses .NET Core 3.1.

   > ⓘ **Note**
   >
   > The example below shows an **OutputType** of *WinExe*, which specifies a Windows GUI executable (and prevents a console window from opening when

the app runs). If your app doesn't have a GUI, then your **OutputType** will have a different value. You can call WinRT APIs from Windows GUI apps, console apps, and libraries. Also, your value for **TargetFramework** might not exactly match the example below.

XML

```
<Project Sdk="Microsoft.NET.Sdk.WindowsDesktop">
  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <UseWindowsForms>true</UseWindowsForms>
  </PropertyGroup>
</Project>
```

2. Replace the **TargetFramework** element in the file with a **TargetFrameworks** element (note the plural). In this element, specify the Target Framework Monikers (TFMs) for all the versions of .NET you want to target, separated by semi-colons.

   - For .NET 6 or later, use one of the following Target Framework Monikers (TFMs):
     - **net6.0-windows10.0.17763.0**: If your app targets Windows 10, version 1809.
     - **net6.0-windows10.0.18362.0**: If your app targets Windows 10, version 1903.
     - **net6.0-windows10.0.19041.0**: If your app targets Windows 10, version 2004.
     - **net6.0-windows10.0.22000.0**: If your app targets initial release of Windows 11.
     - **net6.0-windows10.0.22621.0**: If your app targets Windows 11, version 22H2.
     - **net6.0-windows10.0.26100.0**: If your app targets Windows 11, version 24H2.
   - For .NET Core 3.x, use **netcoreapp3.0** or **netcoreapp3.1**.
   - For .NET Framework, use **net46**.

   The following example demonstrates how to multi-target .NET Core 3.1 and .NET 6 (for Windows 10, version 2004).

   XML

   ```
   <TargetFrameworks>netcoreapp3.1;net6.0-
   windows10.0.19041.0</TargetFrameworks>
   ```

3. After the **PropertyGroup** element, add a **PackageReference** element that includes a conditional statement that installs the `Microsoft.Windows.SDK.Contracts` NuGet package for any versions of .NET Core 3.x or .NET Framework that your app targets. The **PackageReference** element must be a child of an **ItemGroup** element. The following example demonstrates how to do this for .NET Core 3.1.

   XML

   ```
   <ItemGroup>
     <PackageReference Condition="'$(TargetFramework)' == 'netcoreapp3.1'"
                       Include="Microsoft.Windows.SDK.Contracts"
                       Version="10.0.19041.0" />
   </ItemGroup>
   ```

   When you're done, your project file should look similar to this.

   XML

   ```
   <Project Sdk="Microsoft.NET.Sdk.WindowsDesktop">
     <PropertyGroup>
       <OutputType>WinExe</OutputType>
       <TargetFrameworks>netcoreapp3.1;net6.0-
   windows10.0.19041.0</TargetFrameworks>
       <UseWPF>true</UseWPF>
     </PropertyGroup>
     <ItemGroup>
       <PackageReference Condition="'$(TargetFramework)' ==
   'netcoreapp3.1'"
                         Include="Microsoft.Windows.SDK.Contracts"
                         Version="10.0.19041.0" />
     </ItemGroup>
   </Project>
   ```

4. Save your changes and close the project file.

Modify a C++ desktop (Win32) project to use Windows Runtime APIs

Use C++/WinRT to consume WinRT APIs. C++/WinRT is an entirely standard modern C++17 language projection for WinRT APIs, implemented as a header-file-based library, and designed to provide you with first-class access to the modern Windows API.

To configure your project for C++/WinRT:

- For new projects, you can install the C++/WinRT Visual Studio Extension (VSIX) and use one of the C++/WinRT project templates included in that extension.

- For existing projects, you can install the Microsoft.Windows.CppWinRT NuGet package in the project.

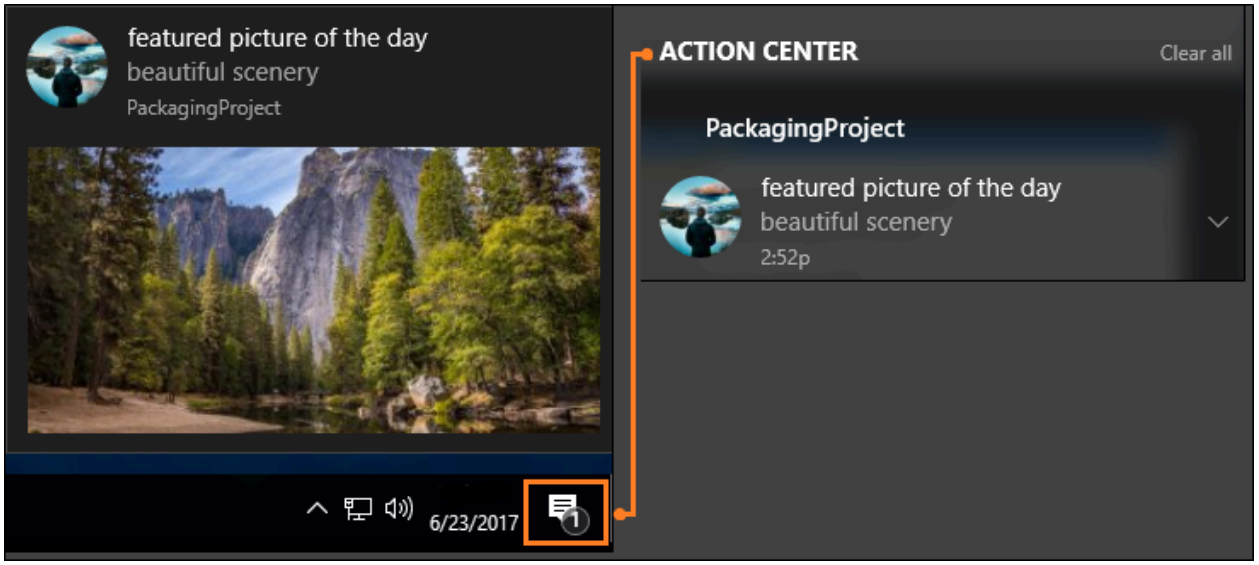
For more details about these options, see Visual Studio support for C++/WinRT, and the VSIX.

Add Windows 10 experiences

Now you're ready to add modern experiences that light up when users run your application on Windows 10. Use this design flow.

✅ **First, decide what experiences you want to add**

There's lots to choose from. For example, you can simplify your purchase order flow by using monetization APIs, or direct attention to your application when you have something interesting to share, such as a new picture that another user has posted.



Even if users ignore or dismiss your message, they can see it again in the action center, and then click on the message to open your app. This increases engagement with your application and has the added bonus of making your application appear deeply integrated with the operating system. We'll show you the code for that experience a bit later in this article.

Visit the UWP documentation for more ideas.

✅ **Decide whether to enhance or extend**

You'll often hear us use the terms *enhance* and *extend*, so we'll take a moment to explain exactly what each of these terms mean.

We use the term *enhance* to describe WinRT APIs that you can call directly from your desktop app whether or not it's a packaged app. When you've chosen a Windows 10 experience, identify the APIs that you need to create it, and then see if that API appears in this list. This is a list of APIs that you can call directly from your desktop app. If your API does not appear in this list, that's because the functionality associated with that API can run only within a UWP process. Often times, these include APIs that render UWP XAML such as a UWP map control or a Windows Hello security prompt.

> ⓘ **Note**
>
> Although APIs that render UWP XAML typically cannot be called directly from your desktop, you might be able to use alternative approaches. If you want to host UWP XAML controls or other custom visual experiences, you can use **XAML Islands** (starting in Windows 10, version 1903) and the **Visual layer** (starting in Windows 10, version 1803). These features can be used in packaged or unpackaged desktop apps.

If you have chosen to package your desktop app, then another option is to *extend* the application by adding a UWP project to your solution. The desktop project is still the entry point of your application, but the UWP project gives you access to all of the APIs that do not appear in this list. The desktop app can communicate with the UWP process by using a an app service and we have lots of guidance on how to set that up. If you want to add an experience that requires a UWP project, see Extend with UWP components.

✅ Reference API contracts

If you can call the API directly from your desktop app, open a browser and search for the reference topic for that API. Beneath the summary of the API, you'll find a table that describes the API contract for that API. Here's an example of that table:

Windows 10 requirements

Device family	Windows 10 (introduced v10.0.10240.0)
API contract	Windows.Foundation.UniversalApiContract (introduced v1)

If you have a .NET-based desktop app, add a reference to that API contract, and then set the **Copy Local** property of that file to **False**. If you have a C++-based project, add to your **Additional Include Directories**, a path to the folder that contains this contract.

✅ Call the APIs to add your experience

Here's the code that you'd use to show the notification window that we looked at earlier. These APIs appear in this list so you can add this code to your desktop app and run it right now.

C#

```
using Windows.Foundation;
using Windows.System;
using Windows.UI.Notifications;
using Windows.Data.Xml.Dom;
...

private void ShowToast()
{
    string title = "featured picture of the day";
    string content = "beautiful scenery";
    string image = "https://picsum.photos/360/180?image=104";
    string logo = "https://picsum.photos/64?image=883";

    string xmlString =
    $@"<toast><visual>
       <binding template='ToastGeneric'>
       <text>{title}</text>
       <text>{content}</text>
       <image src='{image}'/>
       <image src='{logo}' placement='appLogoOverride' hint-crop='circle'/>
       </binding>
      </visual></toast>";

    XmlDocument toastXml = new XmlDocument();
    toastXml.LoadXml(xmlString);

    ToastNotification toast = new ToastNotification(toastXml);

    ToastNotificationManager.CreateToastNotifier().Show(toast);
}
```

C++/WinRT

```
#include <sstream>
#include <winrt/Windows.Data.Xml.Dom.h>
#include <winrt/Windows.UI.Notifications.h>

using namespace winrt::Windows::Foundation;
using namespace winrt::Windows::System;
using namespace winrt::Windows::UI::Notifications;
using namespace winrt::Windows::Data::Xml::Dom;

void UWP::ShowToast()
{
    std::wstring const title = L"featured picture of the day";
    std::wstring const content = L"beautiful scenery";
```

```
    std::wstring const image = L"https://picsum.photos/360/180?image=104";
    std::wstring const logo = L"https://picsum.photos/64?image=883";

    std::wostringstream xmlString;
    xmlString << L"<toast><visual><binding template='ToastGeneric'>" <<
        L"<text>" << title << L"</text>" <<
        L"<text>" << content << L"</text>" <<
        L"<image src='" << image << L"'/>" <<
        L"<image src='" << logo << L"'" <<
        L" placement='appLogoOverride' hint-crop='circle'/>" <<
        L"</binding></visual></toast>";

    XmlDocument toastXml;

    toastXml.LoadXml(xmlString.str().c_str());


ToastNotificationManager::CreateToastNotifier().Show(ToastNotification(toast
Xml));
}
```

C++/CX

```
using namespace Windows::Foundation;
using namespace Windows::System;
using namespace Windows::UI::Notifications;
using namespace Windows::Data::Xml::Dom;

void UWP::ShowToast()
{
    Platform::String ^title = "featured picture of the day";
    Platform::String ^content = "beautiful scenery";
    Platform::String ^image = "https://picsum.photos/360/180?image=104";
    Platform::String ^logo = "https://picsum.photos/64?image=883";

    Platform::String ^xmlString =
        L"<toast><visual><binding template='ToastGeneric'>" +
        L"<text>" + title + "</text>" +
        L"<text>"+ content + "</text>" +
        L"<image src='" + image + "'/>" +
        L"<image src='" + logo + "'" +
        L" placement='appLogoOverride' hint-crop='circle'/>" +
        L"</binding></visual></toast>";

    XmlDocument ^toastXml = ref new XmlDocument();

    toastXml->LoadXml(xmlString);

    ToastNotificationManager::CreateToastNotifier()->Show(ref new
ToastNotification(toastXml));
}
```

To learn more about notifications, see Adaptive and Interactive toast notifications.

Support Windows XP, Windows Vista, and Windows 7/8 install bases

You can modernize your application for Windows 10 without having to create a new branch and maintain separate code bases.

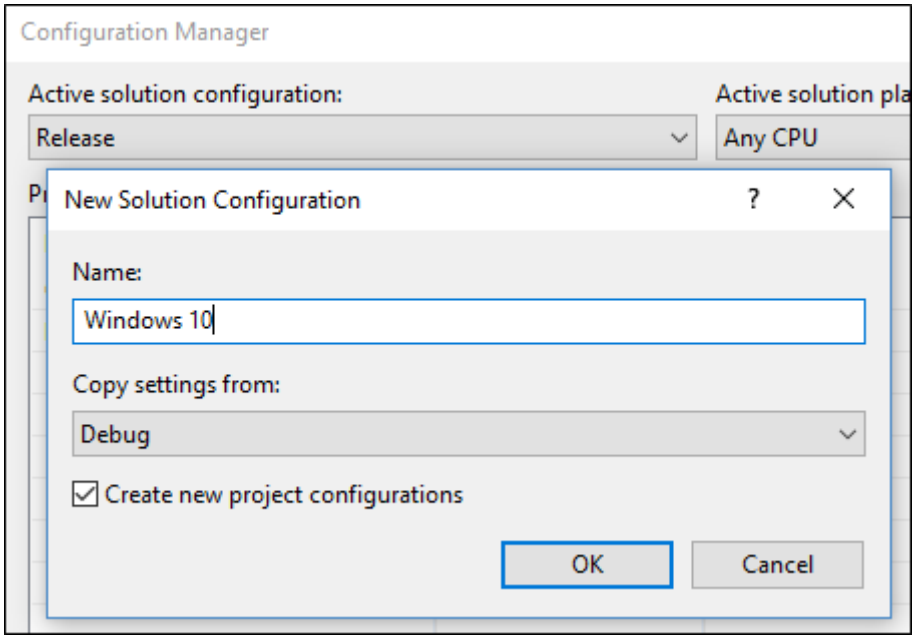
If you want to build separate binaries for Windows 10 users, use conditional compilation. If you'd prefer to build one set of binaries that you deploy to all Windows users, use runtime checks.

Let's take a quick look at each option.

Conditional compilation

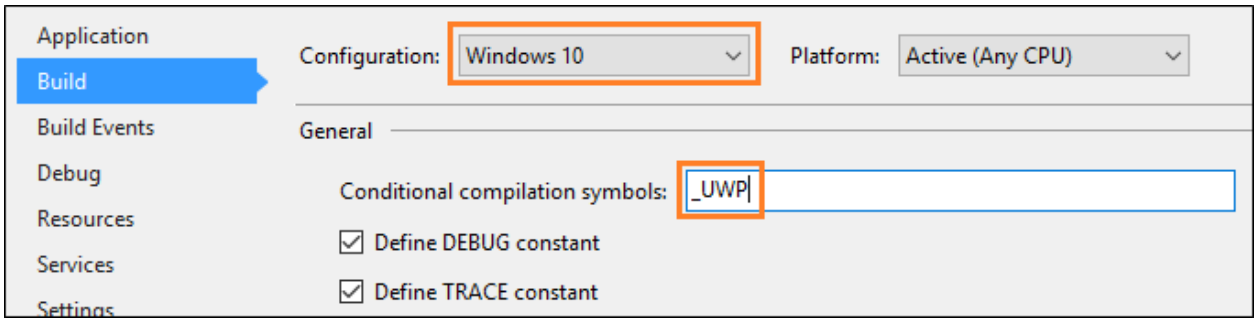
You can keep one code base and compile a set of binaries just for Windows 10 users.

First, add a new build configuration to your project.

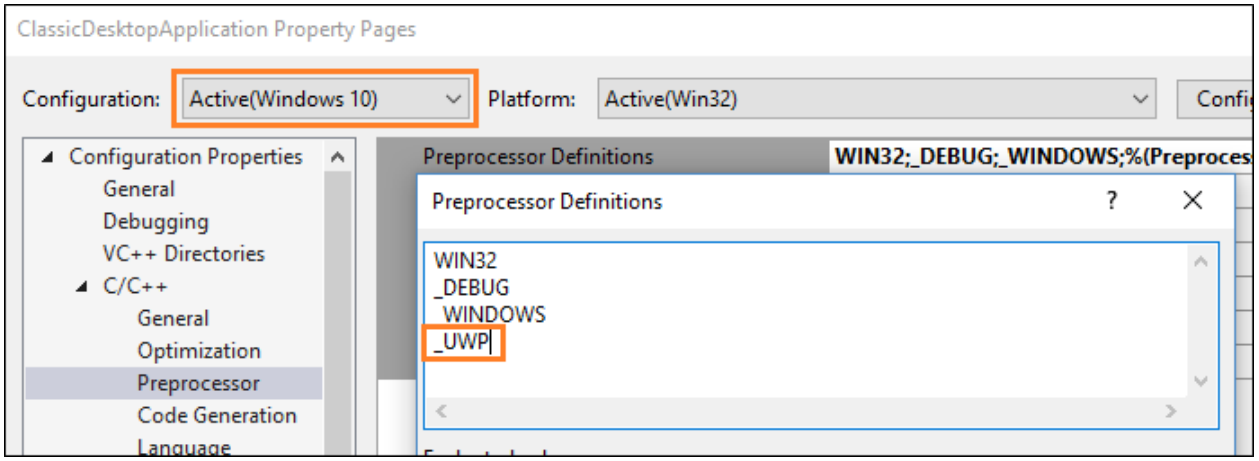


For that build configuration, create a constant that to identify code that calls WinRT APIs.

For .NET-based projects, the constant is called a **Conditional Compilation Constant**.



For C++-based projects, the constant is called a **Preprocessor Definition**.



Add that constant before any block of UWP code.

C#

```
[System.Diagnostics.Conditional("_UWP")]
private void ShowToast()
{
 ...
}
```

C++

```
#if _UWP
void UWP::ShowToast()
{
 ...
}
#endif
```

The compiler builds that code only if that constant is defined in your active build configuration.

Runtime checks

You can compile one set of binaries for all of your Windows users regardless of which version of Windows they run. Your application calls WinRT APIs only if the user is runs your application as a packaged application on Windows 10.

The easiest way to add runtime checks to your code is to install this Nuget package: Desktop Bridge Helpers and then use the `IsRunningAsUWP()` method to gate off all code that calls WinRT APIs. See this blog post for more details: Desktop Bridge - Identify the application's context.

Related Samples

- Hello World Sample
- Secondary Tile
- Store API Sample
- WinForms application that implements a UWP UpdateTask
- Desktop app bridge to UWP Samples

Find answers to your questions

Have questions? Ask us on Stack Overflow. Our team monitors these tags. You can also ask on our forums.

Windows Runtime APIs not supported in desktop apps

Article • 02/27/2024

Although you can use most Windows Runtime (WinRT) APIs (see Windows UWP namespaces) in your C# or C++ desktop app, there are two main sets of WinRT APIs that aren't supported in desktop apps, or that have restrictions:

- APIs that have dependencies on user interface (UI) features that were designed for use only in a Universal Windows Platform (UWP) app.
- APIs that require package identity (see Features that require package identity). Such APIs are supported only in desktop apps that are packaged using MSIX.

This article provides details about both of those sets of WinRT APIs. Where available, this article suggests alternative APIs to achieve the same functionality as the APIs that are unsupported in desktop apps. Most of the alternative APIs are available in WinUI 3 or via WinRT COM interfaces that are available in the Windows SDK.

> **Note**
>
> Apps using .NET can make use of provided class implementations for some of the WinRT COM interfaces listed in this article. Those classes are easier to work with than using the WinRT COM interfaces directly. For more information about the available class implementations, see **Call interop APIs from a .NET app**. Note that those classes require the .NET 6 SDK or later.

This article will be updated as more workarounds and replacements are identified. If you encounter an issue with an API not listed here, please create an issue in the microsoft-ui-xaml repo with the API name and details about what you're trying to achieve by using it.

APIs that have dependencies on UWP-only UI features

Some WinRT APIs were designed specifically for UI scenarios in a UWP app. Those APIs do not behave properly in desktop apps due to threading model and other platform differences. Those APIs, and other WinRT APIs that have dependencies on them, aren't supported for use in desktop apps.

Core unsupported classes

These WinRT classes aren't supported in desktop apps:

Expand table

Class	Alternative APIs
ApplicationView	None
CoreApplicationView	Use the Window class provided by WinUI 3 instead.
CoreApplicationViewTitleBar	Instead of the ExtendViewIntoTitleBar property, use the Window.ExtendsContentIntoTitleBar property provided by WinUI 3.
CoreDispatcher	Use the Microsoft.UI.Xaml.Window.DispatcherQueue property provided by WinUI 3 instead. Note that the Windows.UI.Xaml.Window.Dispatcher and Windows.UI.Xaml.DependencyObject.Dispatcher properties return `null` in a desktop app.
CoreWindow	Also see the Classes that implement IInitializeWithWindow section below. Instead of the GetKeyState method, use the InputKeyboardSource.GetKeyStateForCurrentThread method provided by WinUI 3. Instead of the PointerCursor property, use the UIElement.ProtectedCursor property provided by WinUI 3. You'll need to have a subclass of UIElement to access that property.
UserActivity	Use the IUserActivitySourceHostInterop COM interface instead (in `useractivityinterop.h`).

For other WinRT APIs that aren't supported in desktop apps, see Unsupported members later in this topic.

Classes with an XxxForCurrentView method

Many WinRT classes have a static **GetForCurrentView** or **CreateForCurrentView** method, such as UIViewSettings.GetForCurrentView. Those **XxxForCurrentView** methods have an implicit dependency on the ApplicationView type, which isn't supported in desktop apps. Because **ApplicationView** isn't supported in desktop apps, none of the

XxxForCurrentView methods are supported either. Some unsupported **XxxForCurrentView** methods not only return `null`, but also throw exceptions.

> **Note**
>
> CoreInputView.GetForCurrentView *is* supported in desktop apps, and it *can* be used even without a CoreWindow. You can use that method to retrieve a CoreInputView object on any thread; and if that thread has a foreground window, then that object will produce events.

The following classes *are* supported in desktop apps; but to retrieve an instance of one in a desktop app, you use a mechanism that's different from the **GetForCurrentView** or **CreateForCurrentView** methods. For the classes below that have a COM interface listed as the alternative API, C# developers can also consume those WinRT COM interfaces (see Call interop APIs from a .NET app). The list might not be comprehensive.

Expand table

Class	Alternative APIs
AccountsSettingsPane	Use the IAccountsSettingsPaneInterop COM interface instead (in `accountssettingspaneinterop.h`).
CoreDragDropManager	Use the IDragDropManagerInterop COM interface instead (in `dragdropinterop.h`).
CoreTextServicesManager	This class is currently supported in desktop apps only in Windows Insider Preview builds.
DataTransferManager	Use the IDataTransferManagerInterop COM interface instead (in `shobjidl_core.h`).
DisplayInformation	To retrieve an instance of **DisplayInformation**, use the IDisplayInformationStaticsInterop interface. Alternatively, instead of the LogicalDpi property, you can use the XamlRoot.RasterizationScale property, and listen for changes via the XamlRoot.Changed event (the XamlRoot.RasterizationScale property is provided in WinUI 3). And, instead of the RawPixelsPerViewPixel property, you have the option to use the XamlRoot.RasterizationScale property provided by WinUI 3.
InputPane	Use the IInputPaneInterop COM interface instead (in `inputpaneinterop.h`).

Class	Alternative APIs
PlayToManager	Use the IPlayToManagerInterop COM interface instead (in `playtomanagerinterop.h`).
Print3DManager	Use the IPrinting3DManagerInterop COM interface instead (in `print3dmanagerinterop.h`).
PrintManager	Use the IPrintManagerInterop COM interface instead (in `printmanagerinterop.h`).
RadialController	Use the IRadialControllerInterop COM interface instead (in `radialcontrollerinterop.h`).
RadialControllerConfiguration	Use the IRadialControllerConfigurationInterop COM interface instead (in `radialcontrollerinterop.h`).
ResourceContext	See MRT to MRT Core migration.
ResourceLoader	See MRT to MRT Core migration.
SpatialInteractionManager	Use the ISpatialInteractionManagerInterop COM interface instead (in `spatialinteractionmanagerinterop.h`).
SystemMediaTransportControls	Use the ISystemMediaTransportControlsInterop COM interface instead (in `systemmediatransportcontrolsinterop.h`).
UserActivityRequestManager	Use the IUserActivityRequestManagerInterop COM interface instead (in `useractivityinterop.h`).
UIViewSettings	Use the IUIViewSettingsInterop COM interface instead (in `uiviewsettingsinterop.h`).

The following classes are *not* supported in desktop apps because the APIs don't provide an alternative to their **GetForCurrentView** or **CreateForCurrentView** method. The list might not be comprehensive.

Expand table

Class	Alternative APIs
AppCapture	None
BrightnessOverride	None
ConnectedAnimationService	None
CoreInputView	None
CoreWindowResizeManager	None

Class	Alternative APIs
DisplayEnhancementOverride	None
EdgeGesture	None
GazeInputSourcePreview	None
HdmiDisplayInformation	None
HolographicKeyboardPlacementOverridePreview	None
KeyboardDeliveryInterceptor	None
LockApplicationHost	None
MouseDevice	None
PointerVisualizationSettings	None
ProtectionPolicyManager	None
SearchPane	None
SettingsPane	None
SystemNavigationManager	None
SystemNavigationManagerPreview	None
WebAuthenticationBroker	None. for more details, see the WebAuthenticationBroker.AuthenticateAsync throws COMException GitHub issue.

Classes that implement IInitializeWithWindow

Certain pickers, popups, dialogs, and other Windows Runtime (WinRT) objects depend on a **CoreWindow**; typically, to display a UI. Even though **CoreWindow** isn't supported in desktop apps (see Core unsupported classes above), you can still use many of those WinRT classes in your desktop app by adding a little bit of interoperation code.

For more info (including a list of affected types), and code examples, see Display WinRT UI objects that depend on CoreWindow.

Unsupported members

This section lists (or describes, where a comprehensive list isn't possible) specific members of WinRT classes that aren't supported for use in desktop apps. Unless

otherwise noted, the rest of the classes apart from these members are supported in desktop apps.

Events

The following classes are supported in desktop apps, except for the specified event(s).

Expand table

Class	Unsupported events
UISettings	ColorValuesChanged
AccessibilitySettings	HighContrastChanged

Methods

The following classes are supported in desktop apps, except for the specified method(s).

Expand table

Class	Unsupported methods
DeviceInformationPairing	PairAsync

Methods that use the Request naming pattern

Most methods that follow the **Request** naming pattern—such as AppCapability.RequestAccessAsync and StoreContext.RequestPurchaseAsync—aren't supported in desktop apps. Internally, these methods use the Windows.UI.Popups class. That class requires that the thread have a CoreWindow object, which isn't supported in desktop apps.

The full list of methods that follow the **Request** naming pattern is very long, and this article doesn't provide a comprehensive list of those methods.

APIs that require package identity

The following WinRT classes require package identity (see Features that require package identity). These APIs are supported only in desktop apps that are packaged (that is, that have package identity at runtime). The list might not be comprehensive.

- Windows.ApplicationModel.DataTransfer.DataProviderHandler
- Windows.ApplicationModel.DataTransfer.DataRequest
- Windows.ApplicationModel.DataTransfer.DataRequestDeferral
- Windows.ApplicationModel.DataTransfer.DataRequestedEventArgs
- Windows.ApplicationModel.DataTransfer.DataTransferManager
- Windows.ApplicationModel.DataTransfer.SharedStorageAccessManager
- Windows.ApplicationModel.DataTransfer.TargetApplicationChosenEventArgs
- Windows.ApplicationModel.Resources.Core.NamedResource
- Windows.ApplicationModel.Resources.Core.ResourceCandidate
- Windows.ApplicationModel.Resources.Core.ResourceCandidateVectorView
- Windows.ApplicationModel.Resources.Core.ResourceContext
- Windows.ApplicationModel.Resources.Core.ResourceContextLanguagesVectorView
- Windows.ApplicationModel.Resources.Core.ResourceManager
- Windows.ApplicationModel.Resources.Core.ResourceMap
- Windows.ApplicationModel.Resources.Core.ResourceMapIterator
- Windows.ApplicationModel.Resources.Core.ResourceMapMapView
- Windows.ApplicationModel.Resources.Core.ResourceMapMapViewIterator
- Windows.ApplicationModel.Resources.Core.ResourceQualifier
- Windows.ApplicationModel.Resources.Core.ResourceQualifierMapView
- Windows.ApplicationModel.Resources.Core.ResourceQualifierObservableMap
- Windows.ApplicationModel.Resources.Core.ResourceQualifierVectorView
- Windows.ApplicationModel.Resources.ResourceLoader
- Windows.Data.Pdf.PdfDocument
- Windows.Data.Pdf.PdfPage
- Windows.Data.Pdf.PdfPageDimensions
- Windows.Data.Pdf.PdfPageRenderOptions
- Windows.Data.Text.SelectableWordSegmentsTokenizingHandler
- Windows.Data.Text.SemanticTextQuery
- Windows.Data.Text.TextConversionGenerator
- Windows.Data.Text.TextPredictionGenerator
- Windows.Data.Text.TextReverseConversionGenerator
- Windows.Data.Text.WordSegmentsTokenizingHandler
- Windows.Data.Xml.Dom.DtdEntity
- Windows.Data.Xml.Dom.DtdNotation
- Windows.Data.Xml.Dom.XmlAttribute
- Windows.Data.Xml.Dom.XmlCDataSection
- Windows.Data.Xml.Dom.XmlComment
- Windows.Data.Xml.Dom.XmlDocument
- Windows.Data.Xml.Dom.XmlDocumentFragment
- Windows.Data.Xml.Dom.XmlDocumentType

- Windows.Data.Xml.Dom.XmlDomImplementation
- Windows.Data.Xml.Dom.XmlElement
- Windows.Data.Xml.Dom.XmlEntityReference
- Windows.Data.Xml.Dom.XmlLoadSettings
- Windows.Data.Xml.Dom.XmlNamedNodeMap
- Windows.Data.Xml.Dom.XmlNodeList
- Windows.Data.Xml.Dom.XmlProcessingInstruction
- Windows.Data.Xml.Dom.XmlText
- Windows.Data.Xml.Xsl.XsltProcessor
- Windows.Devices.Input.KeyboardCapabilities
- Windows.Devices.Input.MouseCapabilities
- Windows.Devices.Input.MouseDevice
- Windows.Devices.Input.MouseEventArgs
- Windows.Devices.Input.PointerDevice
- Windows.Devices.Input.TouchCapabilities
- Windows.Devices.Lights.Lamp
- Windows.Devices.Lights.LampAvailabilityChangedEventArgs
- Windows.Devices.Perception.Provider.PerceptionStartFaceAuthenticationHandler
- Windows.Devices.Perception.Provider.PerceptionStopFaceAuthenticationHandler
- Windows.Devices.PointOfService.MagneticStripeReader
- Windows.Devices.PointOfService.MagneticStripeReaderAamvaCardDataReceivedEventArgs
- Windows.Devices.PointOfService.MagneticStripeReaderBankCardDataReceivedEventArgs
- Windows.Devices.PointOfService.MagneticStripeReaderCapabilities
- Windows.Devices.PointOfService.MagneticStripeReaderCardTypes
- Windows.Devices.PointOfService.MagneticStripeReaderEncryptionAlgorithms
- Windows.Devices.PointOfService.MagneticStripeReaderErrorOccurredEventArgs
- Windows.Devices.PointOfService.MagneticStripeReaderReport
- Windows.Devices.PointOfService.MagneticStripeReaderStatusUpdatedEventArgs
- Windows.Devices.PointOfService.MagneticStripeReaderTrackData
- Windows.Devices.PointOfService.MagneticStripeReaderVendorSpecificCardDataReceivedEventArgs
- Windows.Devices.Portable.ServiceDevice
- Windows.Devices.Portable.StorageDevice
- Windows.Devices.Printers.Print3DDevice
- Windows.Devices.Printers.PrintSchema
- Windows.Devices.SmartCards.SmartCard
- Windows.Devices.SmartCards.SmartCardConnection
- Windows.Devices.SmartCards.SmartCardReader
- Windows.Foundation.AsyncActionCompletedHandler

- Windows.Foundation.AsyncActionProgressHandler<TProgress>
- Windows.Foundation.AsyncActionWithProgressCompletedHandler<TProgress>
- Windows.Foundation.AsyncOperationCompletedHandler<TProgress>
- Windows.Foundation.Collections.VectorChangedEventHandler<T>
- Windows.Foundation.DeferralCompletedHandler
- Windows.Foundation.Diagnostics.FileLoggingSession
- Windows.Foundation.Diagnostics.LogFileGeneratedEventArgs
- Windows.Foundation.Diagnostics.LoggingActivity
- Windows.Foundation.Diagnostics.LoggingChannel
- Windows.Foundation.Diagnostics.LoggingChannelOptions
- Windows.Foundation.Diagnostics.LoggingFields
- Windows.Foundation.Diagnostics.LoggingOptions
- Windows.Foundation.Diagnostics.LoggingSession
- Windows.Foundation.EventHandler<T>
- Windows.Foundation.MemoryBuffer
- Windows.Globalization.ApplicationLanguages
- Windows.Globalization.JapanesePhoneme
- Windows.Globalization.JapanesePhoneticAnalyzer
- Windows.Globalization.PhoneNumberFormatting.PhoneNumberFormatter
- Windows.Globalization.PhoneNumberFormatting.PhoneNumberInfo
- Windows.Graphics.Imaging.BitmapBuffer
- Windows.Graphics.Imaging.BitmapCodecInformation
- Windows.Graphics.Imaging.BitmapDecoder
- Windows.Graphics.Imaging.BitmapEncoder
- Windows.Graphics.Imaging.BitmapFrame
- Windows.Graphics.Imaging.BitmapProperties
- Windows.Graphics.Imaging.BitmapPropertiesView
- Windows.Graphics.Imaging.BitmapPropertySet
- Windows.Graphics.Imaging.BitmapTransform
- Windows.Graphics.Imaging.BitmapTypedValue
- Windows.Graphics.Imaging.ImageStream
- Windows.Graphics.Imaging.PixelDataProvider
- Windows.Graphics.Imaging.SoftwareBitmap
- Windows.Graphics.Printing3D.Print3DTaskRequestedEventArgs
- Windows.Graphics.Printing3D.Print3DTaskSourceRequestedHandler
- Windows.Graphics.Printing3D.Printing3D3MFPackage
- Windows.Graphics.Printing3D.Printing3DBaseMaterial
- Windows.Graphics.Printing3D.Printing3DBaseMaterialGroup
- Windows.Graphics.Printing3D.Printing3DColorMaterial
- Windows.Graphics.Printing3D.Printing3DColorMaterialGroup
- Windows.Graphics.Printing3D.Printing3DComponent

- Windows.Graphics.Printing3D.Printing3DComponentWithMatrix
- Windows.Graphics.Printing3D.Printing3DCompositeMaterial
- Windows.Graphics.Printing3D.Printing3DCompositeMaterialGroup
- Windows.Graphics.Printing3D.Printing3DFaceReductionOptions
- Windows.Graphics.Printing3D.Printing3DMaterial
- Windows.Graphics.Printing3D.Printing3DMesh
- Windows.Graphics.Printing3D.Printing3DMeshVerificationResult
- Windows.Graphics.Printing3D.Printing3DModel
- Windows.Graphics.Printing3D.Printing3DModelTexture
- Windows.Graphics.Printing3D.Printing3DMultiplePropertyMaterial
- Windows.Graphics.Printing3D.Printing3DMultiplePropertyMaterialGroup
- Windows.Graphics.Printing3D.Printing3DTexture2CoordMaterial
- Windows.Graphics.Printing3D.Printing3DTexture2CoordMaterialGroup
- Windows.Graphics.Printing3D.Printing3DTextureResource
- Windows.Management.Core.ApplicationDataManager
- Windows.Management.Deployment.DeploymentResult
- Windows.Management.Deployment.PackageManager
- Windows.Management.Deployment.PackageUserInformation
- Windows.Management.Deployment.PackageVolume
- Windows.Management.Workplace.MdmPolicy
- Windows.Management.Workplace.WorkplaceSettings
- Windows.Media.AudioBuffer
- Windows.Media.Capture.AdvancedCapturedPhoto
- Windows.Media.Capture.AppCaptureAlternateShortcutKeys
- Windows.Media.Capture.AppCaptureManager
- Windows.Media.Capture.AppCaptureSettings
- Windows.Media.Capture.CapturedFrame
- Windows.Media.Capture.MediaCaptureFailedEventArgs
- Windows.Media.Capture.MediaCaptureFailedEventHandler
- Windows.Media.Capture.MediaCapturePauseResult
- Windows.Media.Capture.MediaCaptureStopResult
- Windows.Media.Capture.OptionalReferencePhotoCapturedEventArgs
- Windows.Media.Capture.RecordLimitationExceededEventHandler
- Windows.Media.ClosedCaptioning.ClosedCaptionProperties
- Windows.Media.Devices.DefaultAudioCaptureDeviceChangedEventArgs
- Windows.Media.Devices.DefaultAudioRenderDeviceChangedEventArgs
- Windows.Media.Devices.MediaDevice
- Windows.Media.DialProtocol.DialApp
- Windows.Media.DialProtocol.DialAppStateDetails
- Windows.Media.DialProtocol.DialDevice
- Windows.Media.FaceAnalysis.DetectedFace

- Windows.Media.FaceAnalysis.FaceDetector
- Windows.Media.FaceAnalysis.FaceTracker
- Windows.Media.MediaExtensionManager
- Windows.Media.MediaProperties.H264ProfileIds
- Windows.Media.MediaProperties.MediaEncodingSubtypes
- Windows.Media.MediaProperties.Mpeg2ProfileIds
- Windows.Media.Ocr.OcrEngine
- Windows.Media.Ocr.OcrLine
- Windows.Media.Ocr.OcrResult
- Windows.Media.Ocr.OcrWord
- Windows.Media.Playback.PlaybackMediaMarker
- Windows.Media.Playback.PlaybackMediaMarkerReachedEventArgs
- Windows.Media.Playback.PlaybackMediaMarkerSequence
- Windows.Media.SpeechRecognition.SpeechContinuousRecognitionCompletedEventArgs
- Windows.Media.SpeechRecognition.SpeechContinuousRecognitionResultGeneratedEventArgs
- Windows.Media.SpeechRecognition.SpeechContinuousRecognitionSession
- Windows.Media.SpeechRecognition.SpeechRecognitionCompilationResult
- Windows.Media.SpeechRecognition.SpeechRecognitionGrammarFileConstraint
- Windows.Media.SpeechRecognition.SpeechRecognitionHypothesis
- Windows.Media.SpeechRecognition.SpeechRecognitionHypothesisGeneratedEventArgs
- Windows.Media.SpeechRecognition.SpeechRecognitionListConstraint
- Windows.Media.SpeechRecognition.SpeechRecognitionQualityDegradingEventArgs
- Windows.Media.SpeechRecognition.SpeechRecognitionResult
- Windows.Media.SpeechRecognition.SpeechRecognitionSemanticInterpretation
- Windows.Media.SpeechRecognition.SpeechRecognitionTopicConstraint
- Windows.Media.SpeechRecognition.SpeechRecognitionVoiceCommandDefinitionConstraint
- Windows.Media.SpeechRecognition.SpeechRecognizer
- Windows.Media.SpeechRecognition.SpeechRecognizerStateChangedEventArgs
- Windows.Media.SpeechRecognition.SpeechRecognizerTimeouts
- Windows.Media.SpeechRecognition.SpeechRecognizerUIOptions
- Windows.Media.SpeechSynthesis.SpeechSynthesisStream
- Windows.Media.SpeechSynthesis.SpeechSynthesizer
- Windows.Media.SpeechSynthesis.VoiceInformation
- Windows.Networking.PushNotifications.PushNotificationChannel
- Windows.Networking.PushNotifications.PushNotificationChannelManager
- Windows.Networking.PushNotifications.PushNotificationReceivedEventArgs

- Windows.Networking.PushNotifications.RawNotification
- Windows.Networking.Sockets.DatagramSocketMessageReceivedEventArgs
- Windows.Networking.Sockets.MessageWebSocketMessageReceivedEventArgs
- Windows.Services.Maps.Guidance.GuidanceAudioNotificationRequestedEventArgs
- Windows.Services.Maps.Guidance.GuidanceLaneInfo
- Windows.Services.Maps.Guidance.GuidanceManeuver
- Windows.Services.Maps.Guidance.GuidanceMapMatchedCoordinate
- Windows.Services.Maps.Guidance.GuidanceNavigator
- Windows.Services.Maps.Guidance.GuidanceReroutedEventArgs
- Windows.Services.Maps.Guidance.GuidanceRoadSegment
- Windows.Services.Maps.Guidance.GuidanceRoadSignpost
- Windows.Services.Maps.Guidance.GuidanceRoute
- Windows.Services.Maps.Guidance.GuidanceTelemetryCollector
- Windows.Services.Maps.Guidance.GuidanceUpdatedEventArgs
- Windows.Services.Maps.LocalSearch.LocalCategories
- Windows.Services.Maps.LocalSearch.LocalLocation
- Windows.Services.Maps.LocalSearch.LocalLocationFinder
- Windows.Services.Maps.LocalSearch.LocalLocationFinderResult
- Windows.Services.Maps.LocalSearch.LocalLocationHoursOfOperationItem
- Windows.Services.Maps.LocalSearch.LocalLocationRatingInfo
- Windows.Services.Maps.MapAddress
- Windows.Services.Maps.MapLocation
- Windows.Services.Maps.MapLocationFinder
- Windows.Services.Maps.MapLocationFinderResult
- Windows.Services.Maps.MapManager
- Windows.Services.Maps.MapRoute
- Windows.Services.Maps.MapRouteDrivingOptions
- Windows.Services.Maps.MapRouteFinder
- Windows.Services.Maps.MapRouteFinderResult
- Windows.Services.Maps.MapRouteLeg
- Windows.Services.Maps.MapRouteManeuver
- Windows.Services.Maps.MapService
- Windows.Services.Store.StoreAcquireLicenseResult
- Windows.Services.Store.StoreAppLicense
- Windows.Services.Store.StoreAvailability
- Windows.Services.Store.StoreCollectionData
- Windows.Services.Store.StoreConsumableResult
- Windows.Services.Store.StoreContext
- Windows.Services.Store.StoreImage
- Windows.Services.Store.StoreLicense

- Windows.Services.Store.StorePackageLicense
- Windows.Services.Store.StorePackageUpdate
- Windows.Services.Store.StorePackageUpdateResult
- Windows.Services.Store.StorePrice
- Windows.Services.Store.StoreProduct
- Windows.Services.Store.StoreProductPagedQueryResult
- Windows.Services.Store.StoreProductQueryResult
- Windows.Services.Store.StoreProductResult
- Windows.Services.Store.StorePurchaseProperties
- Windows.Services.Store.StorePurchaseResult
- Windows.Services.Store.StoreRequestHelper
- Windows.Services.Store.StoreSendRequestResult
- Windows.Services.Store.StoreSku
- Windows.Services.Store.StoreVideo
- Windows.Storage.AccessCache.StorageApplicationPermissions
- Windows.Storage.ApplicationData
- Windows.Storage.ApplicationDataSetVersionHandler
- Windows.Storage.CachedFileManager
- Windows.Storage.DownloadsFolder
- Windows.Storage.FileIO
- Windows.Storage.FileProperties.BasicProperties
- Windows.Storage.FileProperties.DocumentProperties
- Windows.Storage.FileProperties.ImageProperties
- Windows.Storage.FileProperties.MusicProperties
- Windows.Storage.FileProperties.StorageItemContentProperties
- Windows.Storage.FileProperties.StorageItemThumbnail
- Windows.Storage.FileProperties.VideoProperties
- Windows.Storage.KnownFolders
- Windows.Storage.PathIO
- Windows.Storage.StorageFile
- Windows.Storage.StorageFolder
- Windows.Storage.StorageLibrary
- Windows.Storage.StorageProvider
- Windows.Storage.StorageStreamTransaction
- Windows.Storage.StreamedFileDataRequest
- Windows.Storage.StreamedFileDataRequestedHandler
- Windows.Storage.Streams.Buffer
- Windows.Storage.Streams.DataReader
- Windows.Storage.Streams.DataReaderLoadOperation
- Windows.Storage.Streams.DataWriter
- Windows.Storage.Streams.DataWriterStoreOperation

- Windows.Storage.Streams.FileInputStream
- Windows.Storage.Streams.FileOutputStream
- Windows.Storage.Streams.FileRandomAccessStream
- Windows.Storage.Streams.InMemoryRandomAccessStream
- Windows.Storage.Streams.InputStreamOverStream
- Windows.Storage.Streams.OutputStreamOverStream
- Windows.Storage.Streams.RandomAccessStream
- Windows.Storage.Streams.RandomAccessStreamOverStream
- Windows.Storage.Streams.RandomAccessStreamReference
- Windows.Storage.SystemAudioProperties
- Windows.Storage.SystemGPSProperties
- Windows.Storage.SystemImageProperties
- Windows.Storage.SystemMediaProperties
- Windows.Storage.SystemMusicProperties
- Windows.Storage.SystemPhotoProperties
- Windows.Storage.SystemProperties
- Windows.Storage.SystemVideoProperties
- Windows.System.Diagnostics.ProcessCpuUsage
- Windows.System.Diagnostics.ProcessCpuUsageReport
- Windows.System.Diagnostics.ProcessDiagnosticInfo
- Windows.System.Diagnostics.ProcessDiskUsage
- Windows.System.Diagnostics.ProcessDiskUsageReport
- Windows.System.Diagnostics.ProcessMemoryUsage
- Windows.System.Diagnostics.ProcessMemoryUsageReport
- Windows.System.Profile.AnalyticsInfo
- Windows.System.Profile.AnalyticsVersionInfo
- Windows.System.Threading.Core.PreallocatedWorkItem
- Windows.System.Threading.Core.SignalHandler
- Windows.System.Threading.Core.SignalNotifier
- Windows.System.Threading.ThreadPool
- Windows.System.Threading.ThreadPoolTimer
- Windows.System.Threading.TimerDestroyedHandler
- Windows.System.Threading.TimerElapsedHandler
- Windows.System.Threading.WorkItemHandler
- Windows.System.TimeZoneSettings
- Windows.UI.Notifications.BadgeNotification
- Windows.UI.Notifications.BadgeUpdateManager
- Windows.UI.Notifications.BadgeUpdater
- Windows.UI.Notifications.ScheduledTileNotification
- Windows.UI.Notifications.ScheduledToastNotification
- Windows.UI.Notifications.TileNotification

- Windows.UI.Notifications.TileUpdateManager
- Windows.UI.Notifications.TileUpdater
- Windows.UI.Notifications.ToastNotificationHistory
- Windows.UI.StartScreen.JumpList
- Windows.UI.StartScreen.JumpListItem

In addition, when called from a desktop app that doesn't have package identity, the AdaptiveMediaSource.CreateFromUriAsync methods don't support the `ms-appx` and `ms-resource` URI formats.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Call interop APIs from a .NET app

Article • 10/24/2022

As a C# desktop application developer, in .NET you can make use of C# interop classes that represent several interoperability functions and Windows Runtime (WinRT) COM interoperability interfaces. These include C# classes representing IWindowNative, IInitializeWithWindow, the GetWindowIdFromWindow function, and many others.

This topic lists the available C# interop classes, and shows how to use them. The Background section at the end of the topic describes how interop interfaces were used in previous versions of .NET, and why the change was made.

Configure a .NET desktop project to use the C# interop classes

The C# interop classes listed in the next section (Available C# interop classes) are available in .NET either as part of the Windows App SDK, or else by using a particular Target Framework Moniker, as we'll see.

In a WinUI 3 C# desktop project

When you create a new WinUI 3 project in Visual Studio (see Create your first WinUI 3 project), your project is already configured, and you can start using all of the C# interop classes right away.

In other C# desktop project types (WPF or WinForms)

For other .NET desktop project types—such as Windows Presentation Foundation (WPF) or Windows Forms (WinForms)—you'll need to configure your project before you can access the C# interop classes. For the first set of classes listed below, you'll need to reference the Windows App SDK. For the second set, you'll need to configure a Target Framework Moniker that targets Windows 10, version 1809 or later, like this:

1. Open the project file for your C# .NET desktop project.
2. In the `.csproj` file, modify the **TargetFramework** element to target a specific .NET and Windows SDK version. For example, the following element is appropriate for a .NET 6 project that targets Windows 10, version 2004.

```
XML

<PropertyGroup>
  <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
```

```
</PropertyGroup>
```

For more information—including a list of other supported values—see Use the Target Framework Moniker option.

Available C# interop classes

> **Note**
>
> The classes below require the .NET 6 SDK or later.

Here are the available C# interop classes, mapped from their underlying interop function or WinRT COM interop interface. Each class listed implements the function/methods of its underlying interop API, and provides type-safe wrappers for parameters and return values. For example, **Windows.ApplicationModel.DataTransfer.DragDrop.Core.DragDropManagerInterop.GetForWindow** requires an **IntPtr** window handle (HWND) parameter, and returns a **CoreDragDropManager** object. All of the C# interop classes below and associated methods are static.

Available as part of the Windows App SDK

The **Microsoft.UI.Win32Interop** class implements the C# interop methods in the table below. For a code example, see Manage app windows.

Expand table

Interop function	C# interop method
GetDisplayIdFromMonitor	(**Microsoft.UI**) DisplayId **Win32Interop.GetDisplayIdFromMonitor**(IntPtr hmonitor)
GetIconFromIconId	(**Microsoft.UI**) IntPtr **Win32Interop.GetIconFromIconId**(IconId iconId)
GetIconIdFromIcon	(**Microsoft.UI**) IconId **Win32Interop.GetIconIdFromIcon**(IntPtr hicon)
GetMonitorFromDisplayId	(**Microsoft.UI**) IntPtr **Win32Interop.GetMonitorFromDisplayId**(DisplayId displayId)
GetWindowFromWindowId	(**Microsoft.UI**) IntPtr **Win32Interop.GetWindowFromWindowId**(WindowId windowId)
GetWindowIdFromWindow	(**Microsoft.UI**) WindowId **Win32Interop.GetWindowIdFromWindow**(IntPtr hwnd)

Available via Target Framework Moniker

Expand table

WinRT COM interop interface	C# interop class
IAccountsSettingsPaneInterop	(Windows.UI.ApplicationSettings) AccountsSettingsPaneInterop
IDisplayInformationStaticsInterop	Introduced with TFM `net6.0-windows10.0.22621.0` and .NET 6.0.7. (Windows.Graphics.Display) DisplayInformationInterop
IDragDropManagerInterop	(Windows.ApplicationModel.DataTransfer.DragDrop.Core) DragDropManagerInterop
IInitializeWithWindow	(WinRT.Interop) InitializeWithWindow
IInputPaneInterop	(Windows.UI.ViewManagement) InputPaneInterop
IPlayToManagerInterop	(Windows.Media.PlayTo) PlayToManagerInterop
IPrintManagerInterop	(Windows.Graphics.Printing) PrintManagerInterop
IRadialControllerConfigurationInterop	(Windows.UI.Input) RadialControllerConfigurationInterop
IRadialControllerIndependentInputSourceInterop	(Windows.UI.Input.Core) RadialControllerIndependentInputSourceInterop
IRadialControllerInterop	(Windows.UI.Input) RadialControllerInterop
ISpatialInteractionManagerInterop	(Windows.UI.Input.Spatial) SpatialInteractionManagerInterop
ISystemMediaTransportControlsInterop	(Windows.Media) SystemMediaTransportControlsInterop
IUIViewSettingsInterop	(Windows.UI.ViewManagement) UIViewSettingsInterop
IUserConsentVerifierInterop	(Windows.Security.Credentials.UI) UserConsentVerifierInterop
IWebAuthenticationCoreManagerInterop	(Windows.Security.Authentication.Web.Core) WebAuthenticationCoreManagerInterop
IWindowNative	WinUI 3 only (WinRT.Interop) WindowNative For alternatives for WPF and WinForms, see Retrieve a window handle (HWND).

Code example

This code example demonstrates how to use two of the C# interop classes in a WinUI 3 application (see Create your first WinUI 3 project). The example scenario is to display a **Windows.Storage.Pickers.FolderPicker**. But before displaying the picker in a desktop app, it's necessary to initialize it with the handle (HWND) of the owner window.

1. You can obtain a window handle (HWND) by using the **IWindowNative** WinRT COM interop interface. And (looking in the table in the previous section) that interface is represented by the **WinRT.Interop.WindowNative** C# interop class. Here, the `this` object is a reference to a **Microsoft.UI.Xaml.Window** object from the main window code-behind file.
2. To initialize a piece of UI with an owner window, you use the **IInitializeWithWindow** WinRT COM interop interface. And that interface is represented by the **WinRT.Interop.InitializeWithWindow** C# interop class.

C#

```
private async void myButton_Click(object sender, RoutedEventArgs e)
{
    // Create a folder picker.
    var folderPicker = new Windows.Storage.Pickers.FolderPicker();

    // 1. Retrieve the window handle (HWND) of the current WinUI 3 window.
    var hWnd = WinRT.Interop.WindowNative.GetWindowHandle(this);

    // 2. Initialize the folder picker with the window handle (HWND).
    WinRT.Interop.InitializeWithWindow.Initialize(folderPicker, hWnd);

    // Use the folder picker as usual.
    folderPicker.FileTypeFilter.Add("*");
    var folder = await folderPicker.PickSingleFolderAsync();
}
```

Also see Retrieve a window handle (HWND) and Display WinRT UI objects that depend on CoreWindow.

Background

Previous versions of the .NET Framework and .NET Core had built-in knowledge of WinRT. With those previous versions, you could define an interop interface directly in C# with the ComImport attribute, and then directly cast a projected class to that interop interface.

Since WinRT is a Windows-specific technology, to support the portability and efficiency goals of .NET, we lifted the WinRT projection support out of the C# compiler and .NET runtime, and moved it into the C#/WinRT toolkit (see Built-in support for WinRT is removed from .NET).

While the ComImport technique still works for **IUnknown**-based interop interfaces, it no longer works for the IInspectable-based interfaces that are used for interoperating with

WinRT.

So as a replacement, in .NET, you can make use of the C# interop classes described in this topic.

Troubleshooting and known issues

There are currently no known issues for the C# interop classes. To provide feedback, or to report other issues, add your feedback to an existing issue, or file a new issue on the WindowsAppSDK GitHub repo.

Related topics

- Create your first WinUI 3 project
- Manage app windows
- Retrieve a window handle (HWND)
- Display WinRT UI objects that depend on CoreWindow

Use MSIX framework packages dynamically from your desktop app

Article • 03/03/2023

The Windows App SDK and the Windows 11 OS both enable your apps to reference and use MSIX framework packages dynamically at run time by using a feature called *dynamic dependencies*. This feature is intended to be used primarily by unpackaged desktop apps to use APIs and other content provided by MSIX framework packages.

The most common scenario for using the dynamic dependencies feature is to reference the Windows App SDK framework package in an unpackaged app. In some scenarios, you may want to use the dynamic dependencies feature to reference a different framework package from an unpackaged app, such as the framework package for WinUI 2 or the DirectX Runtime.

For an overview of the dynamic dependencies feature and guidance about using it in your apps, see the following articles.

Article	Description
MSIX framework packages and dynamic dependencies	Introduces important concepts related to MSIX framework packages and describes the purpose of dynamic dependencies feature. This article includes details about the package graph for framework package references and the servicing model for framework packages.
Use the Windows App SDK runtime for apps packaged with external location or unpackaged	Describes how to use dynamically take a dependency on the Windows App SDK framework package in an unpackaged app at run time. This scenario enables unpackaged apps to use Windows App SDK features.
Use the dynamic dependency API to reference MSIX packages at run time	Describes how to use the *dynamic dependency API* to dynamically take a dependency on different framework packages (other than the Windows App SDK framework package) in an unpackaged app at run time.

How packaged desktop apps run on Windows

For a deeper dive on what happens to files and registry entries when you create a Windows app package for your desktop application, see Understanding how packaged desktop apps run on Windows.

Related topics

- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Runtime architecture for the Windows App SDK
- MSIX documentation

MSIX framework packages and dynamic dependencies

Article • 03/03/2023

This article introduces important concepts related to *MSIX framework packages*. The information in this article provides useful context to help you better understand the design and purpose of the *dynamic dependencies* feature in the Windows App SDK and in the Windows 11 OS. This feature enables your apps to reference and use MSIX framework packages at run time.

Framework packages and the package graph

MSIX is a package format that provides a modern packaging and deployment experience. It also provides a clean and trusted way to package redistributable libraries, content and components via *MSIX framework packages*. An MSIX framework package allows packaged apps to access components through a single shared source on the user's device, instead of bundling them into the app package. Common framework packages include the Windows App SDK (including WinUI3), WinUI2, VCLibs, and the DirectX Runtime.

Starting in Windows 8 and continuing through Windows 10 and Windows 11, every process has a *package graph* that provides the list of all the packages available to the app, including framework, resource, optional, and main packages. This graph allows the app to load DLLs, content, and run-time class declarations provided by a referenced package. Historically, this graph was fixed at process creation time, and there was no way to alter it at run time:

- For packaged apps, the graph was initialized based on the package dependencies declared in the PackageDependency element in the app's package manifest. When building a packaged app, this was typically done for you during the build process based on your project references and dependencies.
- For unpackaged apps, the package graph was empty, and couldn't be changed. Therefore, unpackaged apps were limited to standard DLL search order and couldn't access framework packages.

This static package graph restriction is lifted with the introduction of the dynamic dependencies support in both the Windows App SDK and in Windows 11. Developers can use dynamic dependencies to reference and use MSIX framework packages from their apps at run time. Dynamic dependencies removes the static package graph

restriction from apps, and developers can decide how they want to leverage framework packages.

Primary scenarios for dynamic dependencies

Although dynamic dependencies enables any app to add a package framework dependency at run time, this feature is primarily intended to be used by apps packaged with external location or unpackaged apps. Packaged apps can still continue to add static dependencies via the PackageDependency element in their package manifest.

- Most developers will use dynamic dependencies only to reference the Windows App SDK framework package in a packaged with external location or unpackaged app, so that the app can call APIs provided by the Windows App SDK runtime. For more information about this scenario, see Use the Windows App SDK runtime for apps packaged with external location or unpackaged.
- In some cases, developers may want to use dynamic dependencies to reference a different framework package (other than the Windows App SDK framework package) from an unpackaged app, such as the framework package for WinUI2 or the DirectX Runtime. For more information about this scenario, see Use the dynamic dependency API to reference MSIX packages at run time.

Servicing model for framework packages

The dynamic dependencies feature preserves the integrity of the servicing pipeline for the framework package that is being referenced and used dynamically at run time.

MSIX framework packages support servicing in a side-by-side model, meaning each version is installed in its own separate versioned folder. This allows applications in use to be able to stay up and running even when a newer app installs a newer version of the framework package. The OS has uninstall logic for when to delete older versions of a given framework package, based on the presence of *install-time references* and *run-time references* for the package.

- When an app is installed, it can create an *install-time reference* to a framework package. This reference informs the OS that the app has a dependency upon the specified framework package so that the OS won't uninstall the framework package while your app is installed.
- When an app needs to use APIs or content in a framework package, it can add a *run-time reference* to the framework package. This reference informs the OS that the framework package is in active use and to handle any version updates in a side-by-side manner. If a new version of the framework package is installed, but a

running app has an older version in use, the OS cannot remove the older version until all run-time references to the older version are removed.

For example, given this scenario:

- **App A** is running and using version 1.0.0.0 of a given framework package.
- **App B** is installed and has a dependency upon version 1.0.0.1 of the same framework package.

In this scenario, both versions of the framework package will be installed and in use by **App A** and **App B**. However, when **App A** is closed by the user and then restarted, it will pick up the newer version 1.0.0.1 of the framework package. At this point, the run-time reference requirement is no longer valid for version 1.0.0.0 of the framework package, and the OS can safely remove the 1.0.0.0 version. Later, when **App A** and **App B** are uninstalled by the user, then the install-time reference requirement is no longer valid and it is safe for the OS to remove the framework package entirely.

For packaged apps that use the PackageDependency element to specify static references to framework packages, the install-time references for framework packages are tracked by the OS when the app is installed or uninstalled. For run-time references that are managed by using the dynamic dependencies feature, the OS knows when a packaged app is running and will avoid removing its in-use framework packages when a newer one is available.

Related topics

- Use the Windows App SDK runtime for apps packaged with external location or unpackaged
- Use the dynamic dependency API to reference MSIX packages at run time
- Windows App SDK deployment guide for framework-dependent apps packaged with external location or unpackaged
- Runtime architecture for the Windows App SDK
- Tutorial: Use the bootstrapper API in an app packaged with external location or unpackaged that uses the Windows App SDK

Use the dynamic dependency API to reference MSIX packages at run time

Article • 07/27/2023

The two implementations

There are two implementations of the *dynamic dependency API* that you can choose from, depending on your target platform and scenario:

- **The Windows App SDK's dynamic dependency API**. The Windows App SDK provides C and C++ functions (in msixdynamicdependency.h), and Windows Runtime (WinRT) types (in the **Microsoft.Windows.ApplicationModel.DynamicDependency** namespace) that implement the dynamic dependency API. You can use this implementation of the API on any version of Windows that supports the Windows App SDK.
- **Windows 11's dynamic dependency API**. Windows 11 also provides C and C++ functions that implement the dynamic dependency API (in appmodel.h). This implementation of the API can be used only by apps that target Windows 11, version 22H2 (10.0; Build 22621), and later.

Also see Differences between the two implementations.

> **Note**
>
> As you'll see in this topic, the Windows App SDK (C/C++) APIs have the same names as the Windows 11 (C/C++) APIs with an additional **Mdd** prefix. **Mdd** stands for *Microsoft Dynamic Dependencies*.

And there are different kinds of MSIX packages, including *framework*, *resource*, *optional*, and *main* packages. The dynamic dependency API enables unpackaged apps to reference and to use *framework* packages such as WinUI 2 and the DirectX Runtime. For more info about framework package dependencies, see MSIX framework packages and dynamic dependencies.

Specifically, the dynamic dependency API provides ways to manage the *install-time references* and *run-time references* for MSIX packages. For more info, see Servicing model for framework packages.

Use the dynamic dependency API

To use the dynamic dependency API in your unpackaged app to take a dependency on an MSIX package, follow this general pattern in your code:

1. Create an install-time reference

In your app's installer, or during the first run of your app, call one of the following functions or methods to specify a set of criteria for the MSIX package that you want to use. This informs the operating system (OS) that your app has a dependency on an MSIX package that meets the specified criteria. If one or more MSIX packages are installed that meet the criteria, then Windows ensures that at least one of them remains installed until the install-time reference is deleted.

- Windows 11 (C/C++): TryCreatePackageDependency
- Windows App SDK (C/C++): MddTryCreatePackageDependency
- Windows App SDK (WinRT): PackageDependency.Create

The criteria you specify include the package family name, minimum version, and architectures; but you can't indicate a specific MSIX package. When you add a run-time reference to the MSIX package, the API chooses the highest version that satisfies the specified criteria.

You must also specify a *lifetime artifact*, which can be the current process, a file, or a Windows Registry key that indicates to the system that the app is still available. If the specified artifact no longer exists, then the OS can assume that the dependency is no longer needed, and it can uninstall the MSIX package if no other apps have declared a dependency on it. That feature is useful for scenarios where an app neglects to remove the install-time pin when it is uninstalled.

This API returns a dependency ID that must be used in other calls to create run-time references, and to delete the install-time reference.

2. Add a run-time reference

When your app needs to use the MSIX package, call one of the following functions or methods to request access to the specified MSIX package, and add a run-time reference for it. Calling this API informs the OS that the MSIX package is in active use, and to handle any version updates in a side-by-side manner (effectively delay-uninstalling or otherwise servicing the older version until after the app is done using it). If successful, the app might activate classes and use content from the MSIX package.

- Windows 11 (C/C++): AddPackageDependency
- Windows App SDK (C/C++): MddAddPackageDependency
- Windows App SDK (WinRT): PackageDependency.Add

When you call this API, you must pass in the dependency ID that was returned when you created the install-time reference, and the desired rank to use for the MSIX package in the process's package graph. This API returns the full name of the MSIX package that was referenced, and a handle that is used to keep track of the active-use dependency. If there are multiple MSIX packages installed that meet the criteria that you specified when you created the install-time reference, then the API chooses the highest version that satisfies the criteria.

3. Remove the run-time reference

When your app is done using the MSIX package, call one of the following functions or methods to remove the run-time reference. Typically, your app will call this API during shutdown. This API informs the OS that it is safe to remove any unnecessary versions of the MSIX package.

- Windows 11 (C/C++): RemovePackageDependency
- Windows App SDK (C/C++): MddRemovePackageDependency
- Windows App SDK (WinRT): PackageDependencyContext.Remove

When you call this API, you must pass in the handle that was returned when you added the run-time reference.

4. Delete the install-time reference

When your app is uninstalled, call one of the following functions or methods to delete the install-time reference. This API informs the OS that it is safe to remove the MSIX package if no other apps have a dependency on it.

- Windows 11 (C/C++): DeletePackageDependency
- Windows App SDK (C/C++): MddDeletePackageDependency
- Windows App SDK (WinRT): PackageDependency.Delete

When you call this API, you must pass in the dependency ID that was returned when you created the install-time reference.

Differences between the two implementations

The need for a lifetime manager (Windows App SDK limitation)

When you use the Windows App SDK's dynamic dependency API to take a dependency on an MSIX package, the API requires help via another installed package and running process to inform Windows that the MSIX package is in use, and to block servicing the framework while it is being used. That component is called a *lifetime manager*.

For its framework package, the Windows App SDK provides a lifetime manager component called the Dynamic Dependency Lifetime Manager (DDLM). However, no other framework packages currently provide a similar lifetime manager component from Microsoft.

Windows 11's dynamic dependency API doesn't have this limitation.

Reference and use a main package (Windows App SDK limitation)

A dynamic dependency can always target a *framework* package. But only Windows 11's dynamic dependency API can reference and use *main* packages as well.

The main package must have correctly configured its app package manifest source file (the `Package.appxmanifest` file in Visual Studio). Specifically, the main package (the target, not the caller) needs to set `<uap15:DependencyTarget>true</>` (see uap15:DependencyTarget). So the purpose of `<uap15::DependencyTarget>` is to enable a dynamic dependency to target a *main* package. In other words, the main package has to opt in to allow itself to be used as a dynamic dependency (whereas framework packages always implicitly allow that).

Reference the Windows App SDK framework package (Windows App SDK limitation)

In an unpackaged app, you can't use the Windows App SDK's dynamic dependency API to reference the Windows App SDK framework package (like you *can* reference other MSIX packages with it). Instead, you need to use the bootstrapper API provided by the Windows App SDK. The bootstrapper API is a specialized form of the dynamic dependency API that's designed to take dependencies on the Windows App SDK framework package. For more info, see Use the Windows App SDK runtime for apps packaged with external location or unpackaged.

Windows 11's dynamic dependency API doesn't have this limitation.

Related topics

- MSIX framework packages and dynamic dependencies
- Dynamic dependencies specification
- Runtime architecture for the Windows App SDK
- Use the Windows App SDK runtime for apps packaged with external location or unpackaged

Host WinRT XAML controls in desktop apps (XAML Islands)

Article • 08/21/2024

> **Important**
>
> This topic uses or mentions types from the CommunityToolkit/Microsoft.Toolkit.Win32 GitHub repo. For important info about XAML Islands support, please see the XAML Islands Notice in that repo.

Starting in Windows 10, version 1903, you can host WinRT XAML controls in non-UWP desktop applications using a feature called *XAML Islands*. This feature enables you to enhance the look, feel, and functionality of your existing WPF, Windows Forms, and C++ desktop (Win32) applications with the latest Windows UI features that are only available via WinRT XAML controls. This means that you can use UWP features such as Windows Ink and controls that support the Fluent Design System in your existing WPF, Windows Forms, and C++ desktop applications.

You can host any WinRT XAML control that derives from Windows.UI.Xaml.UIElement, including:

- Most first-party WinRT XAML control provided by the Windows SDK or the WinUI 2 library (see exceptions).
- Any custom WinRT XAML control (for example, a user control that consists of several WinRT XAML controls that work together). You must have the source code for the custom control so you can compile it with your application.

Fundamentally, XAML Islands are created by using the *WinRT XAML hosting API*. This API consists of several Windows Runtime classes and COM interfaces that were introduced in the Windows 10, version 1903 SDK. We also provide a set of XAML Island .NET controls in the Windows Community Toolkit that use the WinRT XAML hosting API internally and provide a more convenient development experience for WPF and Windows Forms apps.

The way you use XAML Islands depends on your application type and the types of WinRT XAML controls you want to host.

> **Note**

If you have feedback about XAML Islands, create a new issue in the Microsoft.Toolkit.Win32 repo and leave your comments there.

Requirements

XAML Islands have these run time requirements:

- Windows 10, version 1903, or a later release.
- If your application is not packaged in an MSIX package for deployment, the computer must have the Visual C++ Runtime installed.

WPF and Windows Forms applications

> **Note**
>
> Using XAML Islands to host WinRT XAML controls in WPF and Windows Forms apps is currently supported only in apps that target .NET Core 3.x. XAML Islands are not yet supported in apps that target .NET, or in apps that any version of the .NET Framework.

We recommend that WPF and Windows Forms applications use the XAML Island .NET controls that are available in the Windows Community Toolkit. These controls provide an object model that mimics (or provides access to) the properties, methods, and events of the corresponding WinRT XAML controls. They also handle behavior such as keyboard navigation and layout changes.

There are two sets of XAML Island controls for WPF and Windows Forms applications: *wrapped controls* and *host controls*.

Wrapped controls

WPF and Windows Forms applications can use a selection of XAML Island controls that wrap the interface and functionality of a specific WinRT XAML control. You can add these controls directly to the design surface of your WPF or Windows Forms project and then use them like any other WPF or Windows Forms control in the designer.

The following wrapped WinRT XAML controls are currently available in the Windows Community Toolkit.

Expand table

Control	Minimum supported OS	Description
InkCanvas InkToolbar	Windows 10, version 1903	Provide a surface and related toolbars for Windows Ink-based user interaction in your Windows Forms or WPF desktop application.
MediaPlayerElement	Windows 10, version 1903	Embeds a view that streams and renders media content such as video in your Windows Forms or WPF desktop application.
MapControl	Windows 10, version 1903	Enables you to display a symbolic or photorealistic map in your Windows Forms or WPF desktop application.

For a walkthrough that demonstrates how to use the wrapped WinRT XAML controls, see Use XAML Islands to host a UWP XAML control in a C# WPF app.

Host controls

For custom controls and other scenarios beyond those covered by the available wrapped controls, WPF and Windows Forms applications can also use the WindowsXamlHost control that is available in the Windows Community Toolkit.

Expand table

Control	Minimum supported OS	Description
WindowsXamlHost	Windows 10, version 1903	Can host any WinRT XAML control that derives from Windows.UI.Xaml.UIElement, including any first-party WinRT XAML control provided by the Windows SDK as well as custom controls.

For walkthroughs that demonstrate how to use the **WindowsXamlHost** control, see Use XAML Islands to host a UWP XAML control in a C# WPF app and Host a custom WinRT XAML control in a WPF app using XAML Islands.

Configure your project to use the XAML Island .NET controls

The XAML Island .NET controls require Windows 10, version 1903, or a later version. To use these controls, install one of the NuGet packages listed below. These packages

provide everything you need to use the XAML Island wrapped controls and host controls, and they include other related NuGet packages that are also required.

Expand table

Type of control	NuGet package	Related articles
Wrapped controls	Version 6.0.0 or later of these packages: • WPF: Microsoft.Toolkit.Wpf.UI.Controls • Windows Forms: Microsoft.Toolkit.Forms.UI.Controls	Use XAML Islands to host a UWP XAML control in a C# WPF app
Host control	Version 6.0.0 or later of these packages: • WPF: Microsoft.Toolkit.Wpf.UI.XamlHost • Windows Forms: Microsoft.Toolkit.Forms.UI.XamlHost	Use XAML Islands to host a UWP XAML control in a C# WPF app Host a custom WinRT XAML control in a WPF app

Be aware of the following details:

- The host control packages are also included in the wrapped control packages. You can install the wrapped control packages if you want to use both sets of controls.
- If you're hosting a custom WinRT XAML control, you'll also need to perform some additional steps to reference the custom control. For more info, see Host a custom WinRT XAML control in a WPF app using XAML Islands.

Web view controls

The Windows Community Toolkit also provides the following .NET controls for hosting web content in WPF and Windows Forms applications. These controls are often used in similar desktop app modernization scenarios as the XAML Island controls, and they are maintained in the same Microsoft.Toolkit.Win32 repo as the XAML Island controls.

Expand table

Control	Minimum supported OS	Description
WebView	Windows 10, version 1803	Uses the Microsoft Edge rendering engine to show web content.

Control	Minimum supported OS	Description
WebViewCompatible	Windows 7	Provides a version of **WebView** that is compatible with more OS versions. This control uses the Microsoft Edge rendering engine to show web content on Windows 10 version 1803 and later, and the Internet Explorer rendering engine to show web content on earlier versions of Windows 10, Windows 8.x, and Windows 7.

To use these controls, install one of these NuGet packages:

- WPF: Microsoft.Toolkit.Wpf.UI.Controls.WebView
- Windows Forms: Microsoft.Toolkit.Forms.UI.Controls.WebView

C++ desktop (Win32) applications

The XAML Island .NET controls are not supported in C++ desktop applications. These applications must instead use the *WinRT XAML hosting API* provided by the Windows 10 SDK (version 1903 and later).

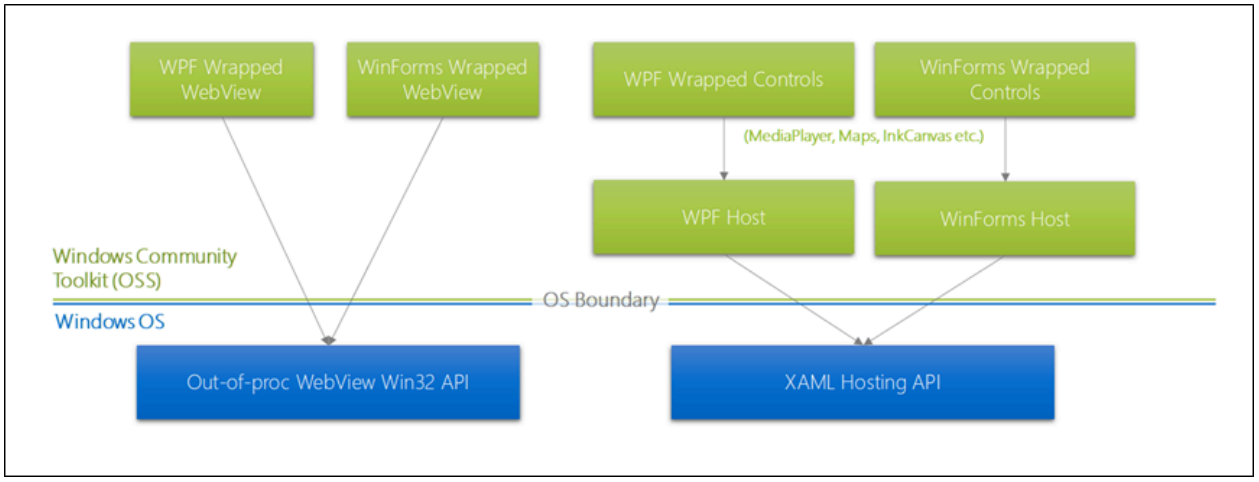
The WinRT XAML hosting API consists of several Windows Runtime classes and COM interfaces that your C++ desktop application can use to host any WinRT XAML control that derives from Windows.UI.Xaml.UIElement. You can host WinRT XAML controls in any UI element in your application that has an associated window handle (HWND). For more information about this API, see the following articles.

- Using the WinRT XAML hosting API in a C++ desktop app
- Host a standard WinRT XAML control in a C++ desktop app
- Host a custom WinRT XAML control in a C++ desktop app

> **Note**
>
> The wrapped controls and host controls in the Windows Community Toolkit use the WinRT XAML hosting API internally and implement all of the behavior you would otherwise need to handle yourself if you used the WinRT XAML hosting API directly, including keyboard navigation and layout changes. For WPF and Windows Forms applications, we strongly recommend that you use these controls instead of the WinRT XAML hosting API directly because they abstract away many of the implementation details of using the API.

Architecture of XAML Islands

Here's a quick look at how the different types of XAML Island controls are organized architecturally on top of the WinRT XAML hosting API.



The APIs that appear at the bottom of this diagram ship with the Windows SDK. The wrapped controls and host controls are available via NuGet packages in the Windows Community Toolkit.

Limitations and workarounds

The following sections discuss limitations and workarounds for certain UWP development scenarios in desktop apps that use XAML Islands.

Supported only with workarounds

✔ Hosting controls from the WinUI 2 Library in a XAML Island is supported conditionally in the current release of XAML Islands. If your desktop app uses an MSIX package for deployment, you can host WinUI controls from prerelease or release versions of the Microsoft.UI.Xaml NuGet package. If your desktop app is not packaged using MSIX, you can host WinUI controls only if you install a prerelease version of the Microsoft.UI.Xaml NuGet package, or if you use the Dynamic Dependencies API. Support for hosting controls from the WinUI 3.0 Library is coming in a later release.

✔ To access the root element of a tree of XAML content in a XAML Island and get related information about the context in which it is hosted, do not use the CoreWindow, ApplicationView, and Window classes. Instead, use the XamlRoot class. For more information, see this section.

✔ To support the Share contract from a WPF, Windows Forms, or C++ desktop (Win32) app, your app must use the IDataTransferManagerInterop interface to get the

DataTransferManager object to initiate the share operation for a specific window. For a sample that demonstrates how to use this interface in a WPF app, see the ShareSource sample.

✔ Using `x:Bind` with hosted controls in XAML Islands is not supported. You'll have to declare the data model in a .NET Standard library.

Not supported

🚫 Using XAML Islands in WPF and Windows Forms apps that target the .NET Framework. XAML Islands are supported only in apps that target .NET Core 3.x.

🚫 UWP XAML content in XAML Islands doesn't respond to Windows theme changes from dark to light or vice versa at run time. Content does respond to high contrast changes at run time.

🚫 Adding a Windows.UI.Xaml.WebView control. For WPF and WinForms apps, see these alternatives.

🚫 The MediaPlayer control and MediaPlayerElement host control are not supported in full screen mode.

🚫 Text input with the handwriting view. For more information about this feature, see this article.

🚫 Text controls that use `@Places` and `@People` content links. For more information about this feature, see this article.

🚫 XAML Islands do not support hosting a ContentDialog that contains a control that accepts text input, such as a TextBox, RichEditBox, or AutoSuggestBox. If you do this, the input control will not properly respond to key presses. To achieve similar functionality using a XAML Island, we recommend that you host a Popup that contains the input control.

🚫 XAML Islands do not currently support displaying SVG files in a hosted Windows.UI.Xaml.Controls.Image control or by using an Windows.UI.Xaml.Media.Imaging.SvgImageSource object. As a workaround, convert the image files you want to display to raster-based formats such as JPG or PNG.

Window host context for XAML Islands

When you host XAML Islands in a desktop app, you can have multiple trees of XAML content running on the same thread at the same time. To access the root element of a

tree of XAML content in a XAML Island and get related information about the context in which it is hosted, use the XamlRoot class. The CoreWindow, ApplicationView, and Window classes won't provide the correct information for XAML Islands. CoreWindow and Window objects do exist on the thread and are accessible to your app, but they won't return meaningful bounds or visibility (they are always invisible and have a size of 1x1). For more information, see Windowing hosts.

For example, to get the bounding rectangle of the window that contains a WinRT XAML control that is hosted in a XAML Island, use the XamlRoot.Size property of the control. Because every WinRT XAML control that can be hosted in a XAML Island derives from Windows.UI.Xaml.UIElement, you can use the XamlRoot property of the control to access the **XamlRoot** object.

```
C#

Size windowSize = myUWPControl.XamlRoot.Size;
```

Do not use the CoreWindows.Bounds property to get the bounding rectangle.

```
C#

// This will return incorrect information for a WinRT XAML control that is
hosted in a XAML Island.
Rect windowSize = CoreWindow.GetForCurrentThread().Bounds;
```

For a table of common windowing-related APIs that you should avoid in the context of XAML Islands and the recommended XamlRoot replacements, see the table in this section.

For a sample that demonstrates how to use this interface in a WPF app, see the ShareSource sample.

Additional resources

For more background information and tutorials about using XAML Islands, see the following articles and resources:

- Modernize a WPF app tutorial: This tutorial provides step-by-step instructions for using the wrapped controls and host controls in the Windows Community Toolkit to add WinRT XAML controls to an existing WPF line-of-business application. This tutorial includes the complete code for the WPF application as well as detailed instructions for each step in the process.

- XAML Islands code samples: This repo contains Windows Forms, WPF, and C++ desktop (Win32) samples that demonstrate how to use XAML Islands.
- XAML Islands v1 - Updates and Roadmap: This blog post discusses many common questions about XAML Islands and provides a detailed development roadmap.

Use XAML Islands to host a UWP XAML control in a C# WPF app

Article • 08/21/2024

> **Important**
>
> This topic uses or mentions types from the CommunityToolkit/Microsoft.Toolkit.Win32 GitHub repo. For important info about XAML Islands support, please see the XAML Islands Notice in that repo.

This topic shows how to build a C# Windows Presentation Foundation (WPF) app (targeting .NET Core 3.1) that uses XAML Islands to host a Universal Windows Platform (UWP) XAML control (that is, a first-party control provided by the Windows SDK). We show how to do that in two ways:

- We show how to host UWP **InkCanvas** and **InkToolbar** controls by using wrapped controls (available in the Windows Community Toolkit). Wrapped controls wrap the interface and functionality of a small set of useful UWP XAML controls. You can add a wrapped control directly to the design surface of your WPF or Windows Forms project, and then use it in the designer like any other WPF or Windows Forms control.
- We also show how to host a UWP **CalendarView** control by using the **WindowsXamlHost** control (available in the Windows Community Toolkit). Because only a small set of UWP XAML controls are available as wrapped controls, you can use **WindowsXamlHost** to host any UWP XAML control.

The process for hosting a UWP XAML control in a WPF app is similar for a Windows Forms app.

> **Important**
>
> Using XAML Islands (wrapped controls or WindowsXamlHost) to host UWP XAML controls is supported only in apps that target .NET Core 3.x. XAML Islands are not supported in apps that target .NET, or in apps that target any version of the .NET Framework.

Recommended components

To host a UWP XAML control in a WPF or Windows Forms app, we recommend that you have the following components in your solution. This topic provides instructions for creating each of these components:

- **The project and source code for your WPF or Windows Forms app**.
- **A UWP project that defines a root Application class that derives from XamlApplication**. The Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication class is available in the Windows Community Toolkit). We recommended that you define your **XamlApplication**-derived **Application** class a separate UWP app project that's part of your WPF or Windows Forms Visual Studio solution.

  > **Note**
  >
  > Making a **XamlApplication**-derived object available to your WPF or Windows Forms project isn't actually required in order to host a first-party UWP XAML control. But it *is* necessary in order to discover, load, and host custom UWP XAML controls. So—to support the full range of XAML Island scenarios—we recommend that you always define a **XamlApplication**-derived object in any solution in which you use XAML Islands.

  > **Note**
  >
  > Your solution can contain only one project that defines a **XamlApplication**-derived object. That one project must reference any other libraries and projects that host UWP XAML controls via XAML Islands.

Create a WPF project

You can follow these instructions to create a new WPF project, and configure it to host XAML Islands. If you have an existing WPF project, then you can adapt these steps and code examples for your project.

1. If you haven't done so already, then install the latest version of .NET Core 3.1.
2. In Visual Studio, create a new C# project from the **WPF Application** project template. Set the **Project name** to *MyWPFApp* so that you won't need to edit any of the steps or source code in this topic. Set the **Framework** to *.NET Core 3.1**, and click **Create**.

> **Important**
>
> Be sure not to use the **WPF App (.NET Framework)** project template.

Configure your WPF project

1. These steps enable package references:
   a. In Visual Studio, click **Tools** > **NuGet Package Manager** > **Package Manager Settings**.
   b. On the right, find the **Package Management** > **Default package management format** setting, and set it to **PackageReference**.

2. Use these steps to install the **Microsoft.Toolkit.Wpf.UI.Controls** NuGet Package:

   a. Right-click the *MyWPFApp* project node in **Solution Explorer**, and choose **Manage NuGet Packages...**.

   b. On the **Browse** tab, type or paste *Microsoft.Toolkit.Wpf.UI.Controls* into the search box. Select the latest stable version, and click **Install**. That package provides everything you need to use the wrapped UWP XAML controls for WPF (including **InkCanvas**, **InkToolbar**, and the **WindowsXamlHost** control).

   > **Note**
   >
   > For a Windows Forms app, reference the Microsoft.Toolkit.Forms.UI.Controls package instead.

3. Most XAML Islands scenarios aren't supported in projects that target **Any CPU**. So to target a specific architecture (such as **x86** or **x64**), do the following:
   a. Right-click the solution node (*not* the project node) in **Solution Explorer**, and choose **Properties**.
   b. Select **Configuration Properties** on the left.
   c. Click the **Configuration Manager...** button.
   d. Under **Active solution platform**, select **New**.
   e. In the **New Solution Platform** dialog, select **x64** or **x86**, and press **OK**.
   f. Close the open dialog boxes.

Define a XamlApplication class in a new UWP project

In this section we'll be adding a UWP project to the solution, and revising the default **App** class in that project to derive from the Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication class provided by the Windows Community Toolkit. That class supports the IXamlMetadataProvider interface, which enables your app to discover and load metadata for custom UWP XAML controls in assemblies in the current directory of your application at run time. That class also initializes the UWP XAML framework for the current thread.

1. In **Solution Explorer**, right-click the solution node, and select **Add** > **New Project...**.

2. Select the C# **Blank App (Universal Windows)** project template. Set the **Project name** to *MyUWPApp* so that you won't need to edit any of the steps or source code in this topic. Set the **Target version** and **Minimum version** to either **Windows 10, version 1903 (Build 18362)** or later.

   > **Note**
   >
   > Be sure not to create *MyUWPApp* in a subfolder of *MyWPFApp*. If you do that, then *MyWPFApp* will try to build the UWP XAML markup as if it were WPF XAML.

3. In *MyUWPApp*, install the Microsoft.Toolkit.Win32.UI.XamlApplication NuGet package (latest stable version). The process for installng a NuGet package was described in the previous section.

4. In *MyUWPApp*, open the `App.xaml` file, and replace its contents with the following XAML:

   XML

   ```
   <xaml:XamlApplication
       x:Class="MyUWPApp.App"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       xmlns:xaml="using:Microsoft.Toolkit.Win32.UI.XamlHost"
       xmlns:local="using:MyUWPApp">
   </xaml:XamlApplication>
   ```

5. Similarly, open `App.xaml.cs`, and replace its contents with the following code:

   C#

   ```
   namespace MyUWPApp
   {
   ```

```
    public sealed partial class App :
Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication
    {
        public App()
        {
            this.Initialize();
        }
    }
}
```

6. Delete the `MainPage.xaml` and `MainPage.xaml.cs` files.

7. Build the *MyUWPApp* project.

In MyWPFApp, add a reference to the MyUWPApp project

1. Specify the compatible framework version in *MyWPFApp*'s project file like this:

   a. In **Solution Explorer**, click the *MyWPFApp* project node to open the project file in the editor.

   b. Inside the first **PropertyGroup** element, add the following child element. Change the `19041` portion of the value as necessary to match the target and minimum OS build of the *MyWPFApp* project.

   XML

   ```
   <AssetTargetFallback>uap10.0.19041</AssetTargetFallback>
   ```

2. In **Solution Explorer**, right-click *MyWPFApp* > **Dependencies**, choose **Add Project Reference...**, and add a reference to the *MyUWPApp* project.

Instantiate the XamlApplication object in the entry point of MyWPFApp

Next, add code to the entry point of *MyWPFApp* to create an instance of the **App** class that you just defined in *MyUWPApp* (the class that derives from **XamlApplication**).

1. Right-click the *MyWPFApp* project node, select **Add** > **New Item...**, and then select **Class**. Set **Name** to *Program.cs*, and click **Add**.

2. Replace the contents of `Program.cs` with the following XAML (and then save the file and build the *MyWPFApp* project):

```
C#

namespace MyWPFApp
{
    public class Program
    {
        [System.STAThreadAttribute()]
        public static void Main()
        {
            using (new MyUWPApp.App())
            {
                var app = new MyWPFApp.App();
                app.InitializeComponent();
                app.Run();
            }
        }
    }
}
```

3. Right-click the *MyWPFApp* project node, and choose **Properties**.

4. In **Application** > **General**, click the **Startup object** drop-down, and choose *MyWPFApp.Program* (which is the fully-qualified name of the **Program** class that you just added). If you don't see it, then try closing and reopening Visual Studio.

   > **Note**
   >
   > By default, a WPF project defines a **Main** entry point function in a generated code file that isn't intended to be modified. The step above changes the entry point for your project to the **Main** method of the new **Program** class, which enables you to add code that runs as early in the startup process of the app as possible.

5. Save your changes to the project properties.

Use wrapped controls to host an InkCanvas and InkToolbar

Now that you've configured your project to use UWP XAML Islands, you're ready to add the InkCanvas and InkToolbar wrapped UWP XAML controls to the app.

1. In *MyWPFApp*, open the `MainWindow.xaml` file.

2. In the **Window** element near the top of the XAML file, add the following attribute. This attribute references the XAML namespace for the **InkCanvas** and **InkToolbar** wrapped UWP XAML controls, and maps it to the **controls** XML namespace.

```
XML

xmlns:controls="clr-
namespace:Microsoft.Toolkit.Wpf.UI.Controls;assembly=Microsoft.Toolkit.
Wpf.UI.Controls"
```

3. Still in `MainWindow.xaml`, edit the existing **Grid** element so that it looks like the XAML below. This XAML adds to the **Grid** an **InkCanvas** and an **InkToolbar** control (prefixed by the **controls** XML namespace that you defined in the previous step).

```
XML

<Grid Margin="10,50,10,10">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <controls:InkToolbar x:Name="myInkToolbar" TargetInkCanvas="
{x:Reference myInkCanvas}" Grid.Row="0" Width="300"
        Height="50" Margin="10,10,10,10" HorizontalAlignment="Left"
VerticalAlignment="Top" />
    <controls:InkCanvas x:Name="myInkCanvas" Grid.Row="1"
HorizontalAlignment="Left" Width="600" Height="400"
        Margin="10,10,10,10" VerticalAlignment="Top" />
</Grid>
```

> **Note**
>
> You can also add these and other wrapped controls to the **Window** by dragging them to the designer from the **Windows Community Toolkit** section of the **Toolbox**.

4. Save `MainWindow.xaml`.

   If you have a device that supports a digital pen (such as a Surface), and you're following along on a physical machine, then you could now build and run the app, and draw digital ink on the screen with the pen. But if you try to write with your mouse, then nothing will happen, because by default **InkCanvas** is enabled only for digital pens. Here's how to enable **InkCanvas** for the mouse.

5. Still in *MyWPFApp*, open `MainWindow.xaml.cs`.

6. Add the following namespace directive to the top of the file:

```
C#

using Microsoft.Toolkit.Win32.UI.Controls.Interop.WinRT;
```

7. Locate the **MainWindow** constructor. Immediately after the call to **InitializeComponent**, add the following line of code:

```
C#

myInkCanvas.InkPresenter.InputDeviceTypes = CoreInputDeviceTypes.Mouse
| CoreInputDeviceTypes.Pen;
```

   You can use the **InkPresenter** object to customize the default inking experience. The code above uses the **InputDeviceTypes** property to enable mouse as well as pen input.

8. Save, build, and run. If you're using a computer with a mouse, then confirm that you can draw something in the ink canvas space with the mouse.

Host a CalendarView by using the host control

In this section, we'll use the WindowsXamlHost control to add a CalendarView to the app.

> **Note**
>
> The WindowsXamlHost control is provided by the Microsoft.Toolkit.Wpf.UI.XamlHost package. That package is included with the Microsoft.Toolkit.Wpf.UI.Controls package that you installed earlier.

1. In **Solution Explorer**, in *MyWPFApp*, open the `MainWindow.xaml` file.

2. In the **Window** element near the top of the XAML file, add the following attribute. This attribute references the XAML namespace for the **WindowsXamlHost** control, and maps it to the **xamlhost** XML namespace.

```
XML
```

```
xmlns:xamlhost="clr-
namespace:Microsoft.Toolkit.Wpf.UI.XamlHost;assembly=Microsoft.Toolkit.
Wpf.UI.XamlHost"
```

3. Still in `MainWindow.xaml`, edit the existing **Grid** element so that it looks like the XAML below. This XAML adds to the **Grid** a **WindowsXamlHost** control (prefixed by the **xamlhost** XML namespace that you defined in the previous step). To host a UWP **CalendarView** control, this XAML sets the **InitialTypeName** property to the fully-qualified name of the control. The XAML also defines an event handler for the **ChildChanged** event, which is raised when the hosted control has been rendered.

```
XML

<Grid Margin="10,50,10,10">
    <xamlhost:WindowsXamlHost x:Name="myCalendar"
InitialTypeName="Windows.UI.Xaml.Controls.CalendarView"
          Margin="10,10,10,10" Width="600" Height="300"
ChildChanged="MyCalendar_ChildChanged" />
</Grid>
```

4. Save `MainWindow.xaml`, and open `MainWindow.xaml.cs`.

5. Delete this line of code, which we added in the previous section: `myInkCanvas.InkPresenter.InputDeviceTypes = CoreInputDeviceTypes.Mouse | CoreInputDeviceTypes.Pen;`.

6. Add the following namespace directive to the top of the file:

```
C#

using Microsoft.Toolkit.Wpf.UI.XamlHost;
```

7. Add the following **ChildChanged** event handler method to the **MainWindow** class. When the host control has been rendered, this event handler runs and creates a simple event handler for the **SelectedDatesChanged** event of the calendar control.

```
C#

private void MyCalendar_ChildChanged(object sender, EventArgs e)
{
    WindowsXamlHost windowsXamlHost = (WindowsXamlHost)sender;

    var calendarView =
        (Windows.UI.Xaml.Controls.CalendarView)windowsXamlHost.Child;
```

```
    if (calendarView != null)
    {
        calendarView.SelectedDatesChanged += (obj, args) =>
        {
            if (args.AddedDates.Count > 0)
            {
                MessageBox.Show("The user selected a new date: " +
                    args.AddedDates[0].DateTime.ToString());
            }
        };
    }
}
```

8. Save, build, and run. Confirm that the calendar control is shown in the window, and that a message box is displayed when you select a date.

Package the app

You can optionally package your WPF app in an MSIX package for deployment. MSIX is the modern and reliable app packaging technology for Windows.

The following instructions show you how to package all the components in the solution into an MSIX package by using the **Windows Application Packaging Project** in Visual Studio (see Set up your desktop application for MSIX packaging in Visual Studio). These steps are necessary only if you want to package the WPF app in an MSIX package.

> ⓘ **Note**
>
> If you choose not to package your application in an MSIX package for deployment, then computers that run your app must have the Visual C++ Runtime installed.

1. Add a new project to your solution created from the **Windows Application Packaging Project** project template. As you create the project, select the same **Target version** and **Minimum version** as you selected for the UWP project.

2. In the packaging project, right-click the **Dependencies** node, and choose **Add Project Reference...**. In the list of projects, select *MyWPFApp*, and click **OK**.

   > ⓘ **Note**
   >
   > If you want to publish your app in the Microsoft Store, then you also have to add a reference to the UWP project in the packaging project.

3. If you followed the steps up to this point, then all of the projects in your solution will target the same specific platform (x86 or x64). And that's necessary in order to build the WPF app into an MSIX package using the Windows Application Packaging Project. To confirm that, you can follow these steps:
   a. Right-click the solution node (*not* the project node) in **Solution Explorer**, and choose **Properties**.
   b. Select **Configuration Properties** on the left.
   c. Click the **Configuration Manager...** button.
   d. Confirm that all of the listed projects have the same value under **Platform**: either *x86* or *x64*.

4. Right-click the project node for the packaging project that you just added, and click **Set as Startup project**.

5. Build and run the packaging project. Confirm that the WPF app runs, and that the UWP control(s) display as expected.

6. For info about distributing/deploying the package, see Manage your MSIX deployment.

Related topics

- Host WinRT XAML controls in desktop apps (XAML Islands)
- InkCanvas class (UWP)
- InkToolbar class (UWP)
- CalendarView class (UWP)
- WindowsXamlHost control for Windows Forms and WPF
- .NET Core 3.1
- PackageReference in project files
- InkCanvas control for Windows Forms and WPF
- InkToolbar control for Windows Forms and WPF
- IXamlMetadataProvider interface
- MSIX documentation
- Set up your desktop application for MSIX packaging in Visual Studio
- Microsoft Visual C++ Redistributable latest supported downloads

Host a custom WinRT XAML control in a WPF app using XAML Islands

Article • 08/21/2024

> **Important**
>
> This topic uses or mentions types from the CommunityToolkit/Microsoft.Toolkit.Win32 GitHub repo. For important info about XAML Islands support, please see the XAML Islands Notice in that repo.

This article demonstrates how to use the WindowsXamlHost control in the Windows Community Toolkit to host a custom WinRT XAML control in a WPF app that targets .NET Core 3.1. The custom control contains several first-party controls from the Windows SDK and binds a property in one of the WinRT XAML controls to a string in the WPF app. This article also demonstrates how to also host a control from the WinUI library.

Although this article demonstrates how to do this in a WPF app, the process is similar for a Windows Forms app. For an overview about hosting WinRT XAML controls in WPF and Windows Forms apps, see this article.

> **Note**
>
> Using XAML Islands to host WinRT XAML controls in WPF and Windows Forms apps is currently supported only in apps that target .NET Core 3.x. XAML Islands are not yet supported in apps that target .NET, or in apps that any version of the .NET Framework.

Required components

To host a custom WinRT XAML control in a WPF (or Windows Forms) app, you'll need the following components in your solution. This article provides instructions for creating each of these components.

- **The project and source code for your app**. Using the WindowsXamlHost control to host custom controls is supported only in apps that target .NET Core 3.x.

- **The custom WinRT XAML control**. You'll need the source code for the custom control you want to host so you can compile it with your app. Typically, the custom control is defined in a UWP class library project that you reference in the same solution as your WPF or Windows Forms project.
- **A UWP app project that defines a root Application class that derives from XamlApplication**. Your WPF or Windows Forms project must have access to an instance of the Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication class provided by the Windows Community Toolkit so that it can discover and load custom UWP XAML controls. The recommended way to do this is to define this object in a separate UWP app project that is part of the solution for your WPF or Windows Forms app.

  > **Note**
  >
  > Your solution can contain only one project that defines a `XamlApplication` object. All custom WinRT XAML controls in your app share the same `XamlApplication` object. The project that defines the `XamlApplication` object must include references to all other WinRT libraries and projects that are used host to controls on the XAML Island.

Create a WPF project

Before getting started, follow these instructions to create a WPF project and configure it to host XAML Islands. If you have an existing WPF project, you can adapt these steps and code examples for your project.

> **Note**
>
> If you have an existing project that targets the .NET Framework, you'll need to migrate your project to .NET Core 3.1. For more information, see this blog series.

1. If you haven't done so already, install the latest version of the .NET Core 3.1 SDK.
2. In Visual Studio 2019, create a new **WPF App (.NET Core)** project.
3. Make sure package references are enabled:
   a. In Visual Studio, click **Tools -> NuGet Package Manager -> Package Manager Settings**.

b. Make sure **PackageReference** is selected for **Default package management format**.

4. Right-click your WPF project in **Solution Explorer** and choose **Manage NuGet Packages**.

5. Select the **Browse** tab, search for the Microsoft.Toolkit.Wpf.UI.XamlHost package and install the latest stable version. This package provides everything you need to use the **WindowsXamlHost** control to host a WinRT XAML control, including other related NuGet packages.

   > **Note**
   >
   > Windows Forms apps must use the Microsoft.Toolkit.Forms.UI.XamlHost package.

6. Configure your solution to target a specific platform such as x86 or x64. Custom WinRT XAML controls are not supported in projects that target **Any CPU**.
   a. In **Solution Explorer**, right-click the solution node and select **Properties** -> **Configuration Properties** -> **Configuration Manager**.
   b. Under **Active solution platform**, select **New**.
   c. In the **New Solution Platform** dialog, select **x64** or **x86** and press **OK**.
   d. Close the open dialog boxes.

Define a XamlApplication class in a UWP app project

Next, add a UWP app project to your solution and revise the default `App` class in this project to derive from the Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication class provided by the Windows Community Toolkit. This class supports the IXamlMetadataProvider interface, which enables your app to discover and load metadata for custom UWP XAML controls in assemblies in the current directory of your application at run time. This class also initializes the UWP XAML framework for the current thread.

1. In **Solution Explorer**, right-click the solution node and select **Add** -> **New Project**.

2. Add a **Blank App (Universal Windows)** project to your solution. Make sure the target version and minimum version are both set to **Windows 10, version 1903 (Build 18362)** or a later release.

3. In the UWP app project, install the Microsoft.Toolkit.Win32.UI.XamlApplication NuGet package (latest stable version).

4. Open the **App.xaml** file and replace the contents of this file with the following XAML. Replace `MyUWPApp` with the namespace of your UWP app project.

```xml
<xaml:XamlApplication
    x:Class="MyUWPApp.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:xaml="using:Microsoft.Toolkit.Win32.UI.XamlHost"
    xmlns:local="using:MyUWPApp">
</xaml:XamlApplication>
```

5. Open the **App.xaml.cs** file and replace the contents of this file with the following code. Replace `MyUWPApp` with the namespace of your UWP app project.

```csharp
namespace MyUWPApp
{
    public sealed partial class App :
Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication
    {
        public App()
        {
            this.Initialize();
        }
    }
}
```

6. Delete the **MainPage.xaml** file from the UWP app project.

7. Clean the UWP app project and then build it.

Add a reference to the UWP project in your WPF project

1. Specify the compatible framework version in the WPF project file.

    a. In **Solution Explorer**, double-click the WPF project node to open the project file in the editor.

b. In the first **PropertyGroup** element, add the following child element. Change the `19041` portion of the value as necessary to match the target and minimum OS build of the UWP project.

```
XML

<AssetTargetFallback>uap10.0.19041</AssetTargetFallback>
```

When you're done, the **PropertyGroup** element should look similar to this.

```
XML

<PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <UseWPF>true</UseWPF>
    <Platforms>AnyCPU;x64</Platforms>
    <AssetTargetFallback>uap10.0.19041</AssetTargetFallback>
</PropertyGroup>
```

2. In **Solution Explorer**, right-click the **Dependencies** node under the WPF project and add a reference to your UWP app project.

Instantiate the XamlApplication object in the entry point of your WPF app

Next, add code to the entry point for your WPF app to create an instance of the `App` class you just defined in the UWP project (this is the class that now derives from `XamlApplication`).

1. In your WPF project, right-click the project node, select **Add** -> **New Item**, and then select **Class**. Name the class **Program** and click **Add**.

2. Replace the generated `Program` class with the following code and then save the file. Replace `MyUWPApp` with the namespace of your UWP app project, and replace `MyWPFApp` with the namespace of your WPF app project.

```
C#

public class Program
{
    [System.STAThreadAttribute()]
    public static void Main()
    {
```

```
        using (new MyUWPApp.App())
        {
            MyWPFApp.App app = new MyWPFApp.App();
            app.InitializeComponent();
            app.Run();
        }
    }
}
```

3. Right-click the project node and choose **Properties**.
4. On the **Application** tab of the properties, click the **Startup object** drop-down and choose the fully qualified name of the `Program` class you added in the previous step.

   > **Note**
   >
   > By default, WPF projects define a `Main` entry point function in a generated code file that isn't intended to be modified. This step changes the entry point for your project to the `Main` method of the new `Program` class, which enables you to add code that runs as early in the startup process of the app as possible.

5. Save your changes to the project properties.

Create a custom WinRT XAML control

To host a custom WinRT XAML control in your WPF app, you must have the source code for the control so you can compile it with your app. Typically custom controls are defined in a UWP class library project for easy portability.

In this section, you will define a simple custom control in a new class library project. You can alternatively define the custom control in the UWP app project you created in the previous section. However, these steps do this in a separate class library project for illustrative purposes because this is typically how custom controls are implemented for portability.

If you already have a custom control, you can use it instead of the control shown here. However, you'll still need to configure the project that contains the control as shown in these steps.

1. In **Solution Explorer**, right-click the solution node and select **Add** -> **New Project**.

2. Add a **Class Library (Universal Windows)** project to your solution. Make sure the target version and minimum version are both set to the same target and minimum OS build as the UWP project.

3. Right-click the project file and select **Unload Project**. Right-click the project file again and select **Edit**.

4. Before the closing `</Project>` element, add the following XML to disable several properties and then save the project file. These properties must be enabled to host the custom control in a WPF (or Windows Forms) app.

```
XML

<PropertyGroup>
  <EnableTypeInfoReflection>false</EnableTypeInfoReflection>
  <EnableXBindDiagnostics>false</EnableXBindDiagnostics>
</PropertyGroup>
```

5. Right-click the project file and select **Reload Project**.

6. Delete the default **Class1.cs** file and add a new **User Control** item to the project.

7. In the XAML file for the user control, add the following `StackPanel` as a child of the default `Grid`. This example adds a `TextBlock` control and then binds the `Text` attribute of that control to the `XamlIslandMessage` field.

```
XML

<StackPanel Background="LightCoral">
    <TextBlock>This is a simple custom WinRT XAML control</TextBlock>
    <Rectangle Fill="Blue" Height="100" Width="100"/>
    <TextBlock Text="{x:Bind XamlIslandMessage}" FontSize="50">
</TextBlock>
</StackPanel>
```

8. In the code-behind file for the user control, add the `XamlIslandMessage` field to the user control class as shown below.

```
C#

public sealed partial class MyUserControl : UserControl
{
    public string XamlIslandMessage { get; set; }

    public MyUserControl()
    {
        this.InitializeComponent();
```

```
    }
}
```

9. Build the UWP class library project.
10. In your WPF project, right-click the **Dependencies** node and add a reference to the UWP class library project.
11. In the UWP app project you configured earlier, right-click the **References** node and add a reference to the UWP class library project.
12. Rebuild the entire solution and make sure all the projects build successfully.

Host the custom WinRT XAML control in your WPF app

1. In **Solution Explorer**, expand the WPF project and open the MainWindow.xaml file or some other window in which you want to host the custom control.
2. In the XAML file, add the following namespace declaration to the `<Window>` element.

```
XML

xmlns:xaml="clr-
namespace:Microsoft.Toolkit.Wpf.UI.XamlHost;assembly=Microsoft.Toolkit.
Wpf.UI.XamlHost"
```

3. In the same file, add the following control to the `<Grid>` element. Change the `InitialTypeName` attribute to the fully qualified name of the user control in your UWP class library project.

```
XML

<xaml:WindowsXamlHost InitialTypeName="UWPClassLibrary.MyUserControl"
ChildChanged="WindowsXamlHost_ChildChanged" />
```

4. Open the code-behind file and add the following code to the `Window` class. This code defines a `ChildChanged` event handler that assigns the value of the `XamlIslandMessage` field of the UWP custom control to the value of the `WPFMessage` field in the WPF app. Change `UWPClassLibrary.MyUserControl` to the fully qualified name of the user control in your UWP class library project.

C#

```
private void WindowsXamlHost_ChildChanged(object sender, EventArgs e)
{
    // Hook up x:Bind source.
    global::Microsoft.Toolkit.Wpf.UI.XamlHost.WindowsXamlHost
windowsXamlHost =
        sender as
global::Microsoft.Toolkit.Wpf.UI.XamlHost.WindowsXamlHost;
    global::UWPClassLibrary.MyUserControl userControl =
        windowsXamlHost.GetUwpInternalObject() as
global::UWPClassLibrary.MyUserControl;

    if (userControl != null)
    {
        userControl.XamlIslandMessage = this.WPFMessage;
    }
}

public string WPFMessage
{
    get
    {
        return "Binding from WPF to UWP XAML";
    }
}
```

5. Build and run your app and confirm that the UWP user control displays as expected.

Add a control from the WinUI 2 library to the custom control

Traditionally, WinRT XAML controls have been released as part of the Windows OS and made available to developers through the Windows SDK. The WinUI library is an alternative approach, where updated versions of WinRT XAML controls from the Windows SDK are distributed in a NuGet package that is not tied to Windows SDK releases. This library also includes new controls that aren't part of the Windows SDK and the default UWP platform.

This section demonstrates how to add a WinRT XAML control from the WinUI 2 library to your user control.

> **Note**

Currently, XAML Islands only supports hosting controls from the WinUI 2 library. Support for hosting controls from the WinUI 3 library is coming in a later release.

1. In the UWP app project, install the latest release or prerelease version of the Microsoft.UI.Xaml NuGet package.

   > **Note**
   >
   > If your desktop app is packaged in an **MSIX package**, you can use either a prerelease or release version of the **Microsoft.UI.Xaml** NugGet package. If your desktop app is not packaged using MSIX, you must install a prerelease version of the **Microsoft.UI.Xaml** NuGet package.

2. In the App.xaml file in this project, add the following child element to the `<xaml:XamlApplication>` element.

   XML

   ```
   <Application.Resources>
       <XamlControlsResources xmlns="using:Microsoft.UI.Xaml.Controls" />
   </Application.Resources>
   ```

   After adding this element, the contents of this file should now look similar to this.

   XML

   ```
   <xaml:XamlApplication
       x:Class="MyUWPApp.App"
       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       xmlns:xaml="using:Microsoft.Toolkit.Win32.UI.XamlHost"
       xmlns:local="using:MyUWPApp">
       <Application.Resources>
           <XamlControlsResources xmlns="using:Microsoft.UI.Xaml.Controls"
   />
       </Application.Resources>
   </xaml:XamlApplication>
   ```

3. In the UWP class library project, install the latest version of the Microsoft.UI.Xaml NuGet package (the same version that you installed in the UWP app project).

4. In the same project, open the XAML file for the user control and add the following namespace declaration to the `<UserControl>` element.

XML

```
xmlns:winui="using:Microsoft.UI.Xaml.Controls"
```

5. In the same file, add a `<winui:RatingControl />` element as a child of the `<StackPanel>`. This element adds an instance of the RatingControl class from the WinUI library. After adding this element, the `<StackPanel>` should now look similar to this.

XML

```
<StackPanel Background="LightCoral">
    <TextBlock>This is a simple custom WinRT XAML control</TextBlock>
    <Rectangle Fill="Blue" Height="100" Width="100"/>
    <TextBlock Text="{x:Bind XamlIslandMessage}" FontSize="50">
</TextBlock>
    <winui:RatingControl />
</StackPanel>
```

6. Build and run your app and confirm that the new rating control displays as expected.

Package the app

You can optionally package the WPF app in an MSIX package for deployment. MSIX is the modern app packaging technology for Windows, and it is based on a combination of MSI, .appx, App-V, and ClickOnce installation technologies.

The following instructions show you how to package the all the components in the solution in an MSIX package by using the Windows Application Packaging Project in Visual Studio 2019. These steps are necessary only if you want to package the WPF app in an MSIX package.

> ⓘ **Note**
>
> If you choose to not package your application in an MSIX package for deployment, then computers that run your app must have the Visual C++ Runtime installed.

1. Add a new Windows Application Packaging Project to your solution. As you create the project, select the same **Target version** and **Minimum version** as you selected for the UWP project.

2. In the packaging project, right-click the **Applications** node and choose **Add reference**. In the list of projects, select the WPF project in your solution and click **OK**.

   > ⓘ **Note**
   >
   > If you want to publish your app in the Microsoft Store, you have to add reference to the UWP project in the packaging project.

3. Configure your solution to target a specific platform such as x86 or x64. This is required to build the WPF app into an MSIX package using the Windows Application Packaging Project.
   a. In **Solution Explorer**, right-click the solution node and select **Properties** -> **Configuration Properties** -> **Configuration Manager**.
   b. Under **Active solution platform**, select **x64** or **x86**.
   c. In the row for your WPF project, in the **Platform** column select **New**.
   d. In the **New Solution Platform** dialog, select **x64** or **x86** (the same platform you selected for **Active solution platform**) and click **OK**.
   e. Close the open dialog boxes.

4. Build and run the packaging project. Confirm that the WPF runs and the UWP custom control displays as expected.

5. For info about distributing/deploying the package, see Manage your MSIX deployment.

Resolve "Cannot find a Resource" error when hosting a WinUI control

If you're hosting a custom control that contains a control from the WinUI library, you may encounter a problem where the control cannot be loaded in a packaged app and debugging the code shows the following error.

To resolve this error, copy the **App.xbf** file from the build output folder of the WPF project to the **\AppX\<WPF project>** build output folder of the packaging project.

For example, if the WPF project is named **WPFXamlIslandsApp** and targets x86 platform, copy **App.xbf** from **\WPFXamlIslandsApp\bin\x86\Release\netcoreapp3.1** to **\WPFXamlIslandsApp.Pack\bin\x86\Release\AppX\WPFXamlIslandsAPP**.

Related topics

- Host UWP XAML controls in desktop apps (XAML Islands)
- XAML Islands code samples
- WindowsXamlHost

Using the WinRT XAML hosting API in a C++ desktop (Win32) app

Article • 08/21/2024

> **Important**
>
> This topic uses or mentions types from the CommunityToolkit/Microsoft.Toolkit.Win32 GitHub repo. For important info about XAML Islands support, please see the XAML Islands Notice in that repo.

Starting in Windows 10, version 1903, non-UWP desktop apps (including C++ desktop (Win32), WPF, and Windows Forms apps) can use the *WinRT XAML hosting API* to host WinRT XAML controls in any UI element that is associated with a window handle (HWND). This API enables non-UWP desktop apps to use the latest Windows UI features that are only available via WinRT XAML controls. For example, non-UWP desktop apps can use this API to host WinRT XAML controls that use the Fluent Design System and support Windows Ink.

The WinRT XAML hosting API provides the foundation for a broader set of controls that we are providing to enable developers to bring Fluent UI to non-UWP desktop apps. This feature is called *XAML Islands*. For an overview of this feature, see Host WinRT XAML controls in desktop apps (XAML Islands).

> **Note**
>
> If you have feedback about XAML Islands, create a new issue in the Microsoft.Toolkit.Win32 repo and leave your comments there.

Is the WinRT XAML hosting API the right choice for your desktop app?

The WinRT XAML hosting API provides the low-level infrastructure for hosting WinRT XAML controls in desktop apps. Some types of desktop apps have the option of using alternative, more convenient APIs to accomplish this goal.

- If you have a C++ desktop app and you want to host WinRT XAML controls in your app, you must use the WinRT XAML hosting API. There are no alternatives for these types of apps.

- For WPF and Windows Forms apps, we strongly recommend that you use the XAML Island .NET controls in the Windows Community Toolkit instead of using the WinRT XAML hosting API directly. These controls use the WinRT XAML hosting API internally and implement all of the behavior you would otherwise need to handle yourself if you used the WinRT XAML hosting API directly, including keyboard navigation and layout changes.

Because we recommend that only C++ desktop apps use the WinRT XAML hosting API, this article primarily provides instructions and examples for C++ desktop apps. However, you can use the WinRT XAML hosting API in WPF and Windows Forms apps if you choose. This article points to relevant source code for the host controls for WPF and Windows Forms in the Windows Community Toolkit so you can see how the WinRT XAML hosting API is used by those controls.

Learn how to use the XAML Hosting API

To follow step-by-step instructions with code examples for using the XAML Hosting API in C++ desktop apps, see these articles:

- Host a standard WinRT XAML control
- Host a custom WinRT XAML control
- Advanced scenarios

Samples

The way you use the WinRT XAML hosting API in your code depends on your app type, the design of your app, and other factors. To help illustrate how to use this API in the context of a complete app, this article refers to code from the following samples.

C++ desktop (Win32)

The following samples demonstrate how to use the WinRT XAML hosting API in a C++ desktop app:

- Simple XAML Island sample. This sample demonstrates a basic implementation of hosting a WinRT XAML control in an unpackaged C++ desktop app.
- XAML Island with custom control sample. This sample demonstrates a complete implementation of hosting a custom WinRT XAML control in a packaged C++ desktop app, as well as handling other behavior such as keyboard input and focus navigation.

WPF and Windows Forms

The WindowsXamlHost control in the Windows Community Toolkit serves as a reference sample for using the WinRT XAML hosting API in WPF and Windows Forms apps. The source code is available at the following locations:

- For the WPF version of the control, go here. The WPF version derives from System.Windows.Interop.HwndHost.
- For the Windows Forms version of the control, go here. The Windows Forms version derives from System.Windows.Forms.Control.

> **Note**
>
> We strongly recommend that you use the **XAML Island .NET controls** in the Windows Community Toolkit instead of using the WinRT XAML hosting API directly in WPF and Windows Forms apps. The WPF and Windows Forms sample links in this article are for illustrative purposes only.

Architecture of the API

The WinRT XAML hosting API includes these main Windows Runtime types and COM interfaces.

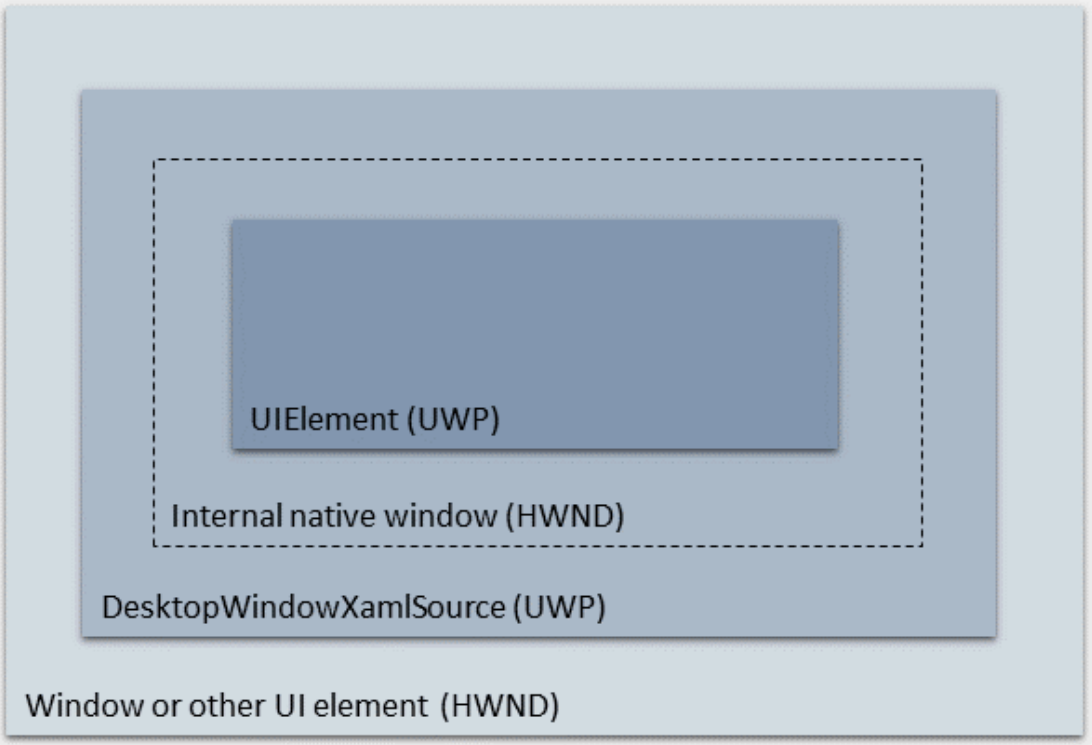
Expand table

Type or interface	Description
WindowsXamlManager	This class represents the UWP XAML framework. This class provides a single static InitializeForCurrentThread method that initializes the UWP XAML framework on the current thread in the desktop app.
DesktopWindowXamlSource	This class represents an instance of UWP XAML content that you are hosting in your desktop app. The most important member of this class is the Content property. You assign this property to a Windows.UI.Xaml.UIElement that you want to host. This class also has other members for routing keyboard focus navigation into and out of the XAML Islands.
IDesktopWindowXamlSourceNative	This COM interface provides the AttachToWindow method, which you use to attach a XAML Island in your app to a

Type or interface	Description
	parent UI element. Every **DesktopWindowXamlSource** object implements this interface.
IDesktopWindowXamlSourceNative2	This COM interface provides the PreTranslateMessage method, which enables the UWP XAML framework to process certain Windows messages correctly. Every **DesktopWindowXamlSource** object implements this interface.

The following diagram illustrates the hierarchy of objects in a XAML Island that is hosted in a desktop app.

- At the base level is the UI element in your app where you want to host the XAML Island. This UI element must have a window handle (HWND). Examples of UI elements in which you can host a XAML Island include a window for C++ desktop apps, a System.Windows.Interop.HwndHost for WPF apps, and a System.Windows.Forms.Control for Windows Forms apps.
- At the next level is a **DesktopWindowXamlSource** object. This object provides the infrastructure for hosting the XAML Island. Your code is responsible for creating this object and attaching it to the parent UI element.
- When you create a **DesktopWindowXamlSource**, this object automatically creates a native child window to host your WinRT XAML control. This native child window is mostly abstracted away from your code, but you can access its handle (HWND) if necessary.
- Finally, at the top level is the WinRT XAML control you want to host in your desktop app. This can be any UWP object that derives from Windows.UI.Xaml.UIElement, including any WinRT XAML control provided by the Windows SDK as well as custom user controls.



Desktop application

> **Note**
>
> When you host XAML Islands in a desktop app, you can have multiple trees of XAML content running on the same thread at the same time. To access the root element of a tree of XAML content in a XAML Island and get related information about the context in which it is hosted, use the **XamlRoot** class. The **CoreWindow**, **ApplicationView**, and **Window** APIs won't provide the correct information for XAML Islands. For more information, see **this section**.

Best practices

When using the WinRT XAML hosting API, follow these best practices for each thread that hosts WinRT XAML controls:

- Create a dedicated WindowsXamlManager for the thread.
- For each WinRT XAML control you want to host, create a DesktopWindowXamlSource.
- Destroy each DesktopWindowXamlSource after it is no longer needed.
- Before exiting the thread, destroy the dedicated WindowsXamlManager for the thread. Note that destruction of this WindowsXamlManager is asynchronous, and

requires draining the Windows message queue before exiting the thread. For examples of how to do this, see the XAML Islands samples.

- After destroying the WindowsXamlManager for a given thread, creating a new WindowsXamlManager on the same thread is not supported and will result in unpredictable behavior.

Troubleshooting

Error using WinRT XAML hosting API in a UWP app

Expand table

Issue	Resolution
Your app receives a **COMException** with the following message: "Cannot activate DesktopWindowXamlSource. This type cannot be used in a UWP app." or "Cannot activate WindowsXamlManager. This type cannot be used in a UWP app."	This error indicates you are trying to use the WinRT XAML hosting API (specifically, you are trying to instantiate the DesktopWindowXamlSource or WindowsXamlManager types) in a UWP app. The WinRT XAML hosting API is only intended to be used in non-UWP desktop apps, such as WPF, Windows Forms, and C++ desktop applications.

Error trying to use the WindowsXamlManager or DesktopWindowXamlSource types

Expand table

Issue	Resolution
Your app receives an exception with the following message: "WindowsXamlManager and DesktopWindowXamlSource are supported for apps targeting Windows version 10.0.18226.0 and later. Please check either the application manifest or package manifest and ensure the MaxTestedVersion property is updated."	This error indicates that your application tried to use the **WindowsXamlManager** or **DesktopWindowXamlSource** types in the WinRT XAML hosting API, but the OS can't determine whether the app was built to target Windows 10, version 1903 or later. The WinRT XAML hosting API was first introduced as a preview in an earlier version of Windows 10, but it is only supported starting in Windows 10, version 1903. To resolve this issue, either create an MSIX package for the app and run it from the package, or install the Microsoft.Toolkit.Win32.UI.SDK NuGet package in your project.

Error attaching to a window on a different thread

Expand table

Issue	Resolution
Your app receives a **COMException** with the following message: "AttachToWindow method failed because the specified HWND was created on a different thread."	This error indicates that your application called the **IDesktopWindowXamlSourceNative::AttachToWindow** method and passed it the HWND of a window that was created on a different thread. You must pass this method the HWND of a window that was created on the same thread as the code from which you are calling the method.

Error attaching to a window on a different top-level window

Expand table

Issue	Resolution
Your app receives a **COMException** with the following message: "AttachToWindow method failed because the specified HWND descends from a different top-level window than the HWND that was previously passed to AttachToWindow on the same thread."	This error indicates that your application called the **IDesktopWindowXamlSourceNative::AttachToWindow** method and passed it the HWND of a window that descends from a different top-level window than a window you specified in a previous call to this method on the same thread. After your application calls **AttachToWindow** on a particular thread, all other DesktopWindowXamlSource objects on the same thread can only attach to windows that are descendants of the same top-level window that was passed in the first call to **AttachToWindow**. When all the **DesktopWindowXamlSource** objects are closed for a particular thread, the next **DesktopWindowXamlSource** is then free to attach to any window again. To resolve this issue, either close all **DesktopWindowXamlSource** objects that are bound to other top-level windows on this thread, or create a new thread for this **DesktopWindowXamlSource**.

Related topics

- Host UWP XAML controls in desktop apps (XAML Islands)
- Host a standard WinRT XAML control in a C++ desktop app
- Host a custom WinRT XAML control in a C++ desktop app

- Advanced scenarios for XAML Islands in C++ desktop apps
- XAML Islands code samples
- C++ desktop XAML Islands sample

Host a standard WinRT XAML control in a C++ desktop (Win32) app

Article • 08/21/2024

> **Important**
>
> This topic uses or mentions types from the CommunityToolkit/Microsoft.Toolkit.Win32 GitHub repo. For important info about XAML Islands support, please see the XAML Islands Notice in that repo.

This article demonstrates how to use the WinRT XAML hosting API to host a standard WinRT XAML control (that is, a control provided by the Windows SDK) in a new C++ desktop app. The code is based on the simple XAML Island sample, and this section discusses some of the most important parts of the code. If you have an existing C++ desktop app project, you can adapt these steps and code examples for your project.

> **Note**
>
> The scenario demonstrated in this article doesn't support directly editing XAML markup for WinRT XAML controls hosted in your app. This scenario restricts you to modifying the appearance and behavior of hosted controls via code. For instructions that enable you to directly edit XAML markup when hosting WinRT XAML controls, see Host a custom WinRT XAML control in a C++ desktop app.

Create a desktop application project

1. In Visual Studio 2019 with the Windows 10, version 1903 SDK (build 10.0.18362) or a later release installed, create a new **Windows Desktop Application** project and name it **MyDesktopWin32App**. This project type is available under the **C++**, **Windows**, and **Desktop** project filters.

2. In **Solution Explorer**, right-click the solution node, click **Retarget solution**, select the **10.0.18362.0** or a later SDK release, and then click **OK**.

3. Install the Microsoft.Windows.CppWinRT NuGet package to enable support for C++/WinRT in your project:
   a. Right-click your project in **Solution Explorer** and choose **Manage NuGet Packages**.

b. Select the **Browse** tab, search for the Microsoft.Windows.CppWinRT package, and install the latest version of this package.

> **Note**
>
> For new projects, you can alternatively install the C++/WinRT Visual Studio Extension (VSIX) and use one of the C++/WinRT project templates included in that extension. For more details, see Visual Studio support for C++/WinRT, and the VSIX.

4. On the **Browse** tab of the **NuGet Package Manager** window, search for the Microsoft.Toolkit.Win32.UI.SDK NuGet package and install the latest stable version of this package. This package provides several build and run time assets that enable XAML Islands to work in your app.

5. Set the `maxversiontested` value in your application manifest to specify that your application is compatible with Windows 10, version 1903.

   a. If you don't already have an application manifest in your project, add a new XML file to your project and name it **app.manifest**.

   b. In your application manifest, include the **compatibility** element and the child elements shown in the following example. Replace the **Id** attribute of the **maxversiontested** element with the version number of Windows you are targeting (this must be 10.0.18362.0 or a later release). Note that setting a higher value means older versions of Windows won't run the app properly because every Windows release only knows of versions before it. If you want the app to run on Windows 10, version 1903 (build 10.0.18362), you should either leave the 10.0.18362.0 value as is, or add multiple **maxversiontested** elements for the different values the app supports.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1"
manifestVersion="1.0">
    <compatibility xmlns="urn:schemas-microsoft-
com:compatibility.v1">
        <application>
            <!-- Windows 10 -->
            <maxversiontested Id="10.0.18362.0"/>
            <supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a15a9a}"
/>
        </application>
```

```
    </compatibility>
</assembly>
```

6. Add a reference to the Windows Runtime metadata:
   a. In **Solution Explorer**, right-click on your project **References** node and select **Add Reference**.
   b. Click the **Browse** button at the bottom of the page and navigate to the UnionMetadata folder in your SDK install path. By default the SDK will be installed to `C:\Program Files (x86)\Windows Kits\10\UnionMetadata`.
   c. Then, select the folder named after the Windows version you are targeting (e.g. 10.0.18362.0) and inside of that folder pick the `Windows.winmd` file.
   d. Click **OK** to close the **Add Reference** dialog.

Use the XAML hosting API to host a WinRT XAML control

The basic process of using the XAML hosting API to host a WinRT XAML control follows these general steps:

1. Initialize the WinRT XAML framework for the current thread before your app creates any of the Windows.UI.Xaml.UIElement objects that it will host. There are several ways to do this, depending on when you plan to create the DesktopWindowXamlSource object that will host the controls.

   - If your application creates the **DesktopWindowXamlSource** object before it creates any of the **Windows.UI.Xaml.UIElement** objects that it will host, this framework will be initialized for you when you instantiate the **DesktopWindowXamlSource** object. In this scenario, you don't need to add any code of your own to initialize the framework.

   - However, if your application creates the **Windows.UI.Xaml.UIElement** objects before it creates the **DesktopWindowXamlSource** object that will host them, your application must call the static WindowsXamlManager.InitializeForCurrentThread method to explicitly initialize the WinRT XAML framework before the **Windows.UI.Xaml.UIElement** objects are instantiated. Your application should typically call this method when the parent UI element that hosts the **DesktopWindowXamlSource** is instantiated.

> ⓘ **Note**

This method returns a WindowsXamlManager object that contains a reference to the WinRT XAML framework. You can create as many **WindowsXamlManager** objects as you want on a given thread. However, because each object holds a reference to the WinRT XAML framework, you should dispose the objects to ensure that XAML resources are eventually released.

2. Create a DesktopWindowXamlSource object and attach it to a parent UI element in your application that is associated with a window handle.

   To do this, you'll need to follow these steps:

   a. Create a **DesktopWindowXamlSource** object and cast it to the IDesktopWindowXamlSourceNative or IDesktopWindowXamlSourceNative2 COM interface.

   b. Call the AttachToWindow method of the **IDesktopWindowXamlSourceNative** or **IDesktopWindowXamlSourceNative2** interface, and pass in the window handle of the parent UI element in your application.

   > **Important**
   >
   > Make sure that your code calls the **AttachToWindow** method only once per DesktopWindowXamlSource object. Calling this method more than once for a **DesktopWindowXamlSource** object could result in a memory leak.

   c. Set the initial size of the internal child window contained in the **DesktopWindowXamlSource**. By default, this internal child window is set to a width and height of 0. If you don't set the size of the window, any WinRT XAML controls you add to the **DesktopWindowXamlSource** will not be visible. To access the internal child window in the **DesktopWindowXamlSource**, use the **WindowHandle** property of the **IDesktopWindowXamlSourceNative** or **IDesktopWindowXamlSourceNative2** interface.

3. Finally, assign the **Windows.UI.Xaml.UIElement** you want to host to the Content property of your **DesktopWindowXamlSource** object.

The following steps and code examples demonstrate how to do implement the above process:

1. In the **Source Files** folder of the project, open the default **MyDesktopWin32App.cpp** file. Delete the entire contents of the file and add the

following `include` and `using` statements. In addition to standard C++ and UWP headers and namespaces, these statements include several items specific to XAML Islands.

C++/WinRT

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>

#include <winrt/Windows.Foundation.Collections.h>
#include <winrt/Windows.system.h>
#include <winrt/windows.ui.xaml.hosting.h>
#include <windows.ui.xaml.hosting.desktopwindowxamlsource.h>
#include <winrt/windows.ui.xaml.controls.h>
#include <winrt/Windows.ui.xaml.media.h>

using namespace winrt;
using namespace Windows::UI;
using namespace Windows::UI::Composition;
using namespace Windows::UI::Xaml::Hosting;
using namespace Windows::Foundation::Numerics;
```

2. Copy the following code after the previous section. This code defines the **WinMain** function for the app. This function initializes a basic window and uses the XAML hosting API to host a simple UWP **TextBlock** control in the window.

C++/WinRT

```
LRESULT CALLBACK WindowProc(HWND, UINT, WPARAM, LPARAM);

HWND _hWnd;
HWND _childhWnd;
HINSTANCE _hInstance;

int CALLBACK WinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE
hPrevInstance, _In_ LPSTR lpCmdLine, _In_ int nCmdShow)
{
    _hInstance = hInstance;

    // The main window class name.
    const wchar_t szWindowClass[] = L"Win32DesktopApp";
    WNDCLASSEX windowClass = { };

    windowClass.cbSize = sizeof(WNDCLASSEX);
    windowClass.lpfnWndProc = WindowProc;
    windowClass.hInstance = hInstance;
    windowClass.lpszClassName = szWindowClass;
    windowClass.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);

    windowClass.hIconSm = LoadIcon(windowClass.hInstance,
```

```
IDI_APPLICATION);

    if (RegisterClassEx(&windowClass) == NULL)
    {
        MessageBox(NULL, L"Windows registration failed!", L"Error",
NULL);
        return 0;
    }

    _hWnd = CreateWindow(
        szWindowClass,
        L"Windows c++ Win32 Desktop App",
        WS_OVERLAPPEDWINDOW | WS_VISIBLE,
        CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
        NULL,
        NULL,
        hInstance,
        NULL
    );
    if (_hWnd == NULL)
    {
        MessageBox(NULL, L"Call to CreateWindow failed!", L"Error",
NULL);
        return 0;
    }


    // Begin XAML Island section.

    // The call to winrt::init_apartment initializes COM; by default,
in a multithreaded apartment.
    winrt::init_apartment(apartment_type::single_threaded);

    // Initialize the XAML framework's core window for the current
thread.
    WindowsXamlManager winxamlmanager =
WindowsXamlManager::InitializeForCurrentThread();

    // This DesktopWindowXamlSource is the object that enables a non-
UWP desktop application
    // to host WinRT XAML controls in any UI element that is associated
with a window handle (HWND).
    DesktopWindowXamlSource desktopSource;

    // Get handle to the core window.
    auto interop = desktopSource.as<IDesktopWindowXamlSourceNative>();

    // Parent the DesktopWindowXamlSource object to the current window.
    check_hresult(interop->AttachToWindow(_hWnd));

    // This HWND will be the window handler for the XAML Island: A
child window that contains XAML.
    HWND hWndXamlIsland = nullptr;

    // Get the new child window's HWND.
```

```
    interop->get_WindowHandle(&hWndXamlIsland);

    // Update the XAML Island window size because initially it is 0,0.
    SetWindowPos(hWndXamlIsland, 0, 200, 100, 800, 200,
SWP_SHOWWINDOW);

    // Create the XAML content.
    Windows::UI::Xaml::Controls::StackPanel xamlContainer;
    xamlContainer.Background(Windows::UI::Xaml::Media::SolidColorBrush{
Windows::UI::Colors::LightGray() });

    Windows::UI::Xaml::Controls::TextBlock tb;
    tb.Text(L"Hello World from Xaml Islands!");
    tb.VerticalAlignment(Windows::UI::Xaml::VerticalAlignment::Center);

tb.HorizontalAlignment(Windows::UI::Xaml::HorizontalAlignment::Center);
    tb.FontSize(48);

    xamlContainer.Children().Append(tb);
    xamlContainer.UpdateLayout();
    desktopSource.Content(xamlContainer);

    // End XAML Island section.

    ShowWindow(_hWnd, nCmdShow);
    UpdateWindow(_hWnd);

    //Message loop:
    MSG msg = { };
    while (GetMessage(&msg, NULL, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }

    return 0;
}
```

3. Copy the following code after the previous section. This code defines the window procedure for the window.

C++/WinRT

```
LRESULT CALLBACK WindowProc(HWND hWnd, UINT messageCode, WPARAM wParam,
LPARAM lParam)
{
    PAINTSTRUCT ps;
    HDC hdc;
    wchar_t greeting[] = L"Hello World in Win32!";
    RECT rcClient;

    switch (messageCode)
    {
```

```
        case WM_PAINT:
            if (hWnd == _hWnd)
            {
                hdc = BeginPaint(hWnd, &ps);
                TextOut(hdc, 300, 5, greeting, wcslen(greeting));
                EndPaint(hWnd, &ps);
            }
            break;
        case WM_DESTROY:
            PostQuitMessage(0);
            break;

        // Create main window
        case WM_CREATE:
            _childhWnd = CreateWindowEx(0, L"ChildWClass", NULL,
WS_CHILD | WS_BORDER, 0, 0, 0, 0, hWnd, NULL, _hInstance, NULL);
            return 0;

        // Main window changed size
        case WM_SIZE:
            // Get the dimensions of the main window's client
            // area, and enumerate the child windows. Pass the
            // dimensions to the child windows during enumeration.
            GetClientRect(hWnd, &rcClient);
            MoveWindow(_childhWnd, 200, 200, 400, 500, TRUE);
            ShowWindow(_childhWnd, SW_SHOW);

            return 0;

            // Process other messages.

        default:
            return DefWindowProc(hWnd, messageCode, wParam, lParam);
            break;
    }

    return 0;
}
```

4. Save the code file, and build and run the app. Confirm that you see the UWP **TextBlock** control in the app window.

> **Note**
>
> You may see the several build warnings, including `warning C4002: too many arguments for function-like macro invocation 'GetCurrentTime'` and `manifest authoring warning 81010002: Unrecognized Element "maxversiontested" in namespace "urn:schemas-microsoft-`

`com:compatibility.v1"`. These warnings are known issues with the current tools and NuGet packages, and they can be ignored.

For complete examples that demonstrate using the XAML hosting API to host a WinRT XAML control, see the following code files:

- **C++ desktop (Win32):** See the XamlBridge.cpp file in the XAML Islands code samples repo.
- **WPF:** See the WindowsXamlHostBase.cs and WindowsXamlHost.cs files in the Windows Community Toolkit.
- **Windows Forms:** See the WindowsXamlHostBase.cs and WindowsXamlHost.cs files in the Windows Community Toolkit.

Package the app

You can optionally package the app in an MSIX package for deployment. MSIX is the modern app packaging technology for Windows, and it is based on a combination of MSI, .appx, App-V and ClickOnce installation technologies.

The following instructions show you how to package the all the components in the solution in an MSIX package by using the Windows Application Packaging Project in Visual Studio 2019. These steps are necessary only if you want to package the app in an MSIX package.

> **Note**
>
> If you choose to not package your application in an MSIX package for deployment, then computers that run your app must have the Visual C++ Runtime installed.

1. Add a new Windows Application Packaging Project to your solution. As you create the project, select **Windows 10, version 1903 (10.0; Build 18362)** for both the **Target version** and **Minimum version**.

2. In the packaging project, right-click the **Applications** node and choose **Add reference**. In the list of projects, select the C++ desktop application project in your solution and click **OK**.

3. Build and run the packaging project. Confirm that the app runs and displays the WinRT XAML controls as expected.

4. For info about distributing/deploying the package, see Manage your MSIX deployment.

Next steps

The code examples in this article get you started with the basic scenario of hosting a standard WinRT XAML control in a C++ desktop app. The following sections introduce additional scenarios that your application may need to support.

Host a custom WinRT XAML control

For many scenarios, you may need to host a custom WinRT XAML control that contains several individual controls that work together. The process for hosting a custom control (either a control you define yourself or a control provided by a 3rd party) in a C++ desktop app is more complex than hosting a standard control, and requires additional code.

For a complete walkthrough, see Host a custom WinRT XAML control in a C++ desktop app using the XAML Hosting API.

Advanced scenarios

Many desktop applications that host XAML Islands will need to handle additional scenarios in order to provide a smooth user experience. For example, desktop applications may need to handle keyboard input in XAML Islands, focus navigation between XAML Islands and other UI elements, and layout changes.

For more information about handling these scenarios and pointers to related code samples, see Advanced scenarios for XAML Islands in C++ desktop apps.

Related topics

- Host WinRT XAML controls in desktop apps (XAML Islands)
- Using the WinRT XAML hosting API in a C++ desktop app
- Host a custom WinRT XAML control in a C++ desktop app
- Advanced scenarios for XAML Islands in C++ desktop apps
- XAML Islands code samples

Host a custom WinRT XAML control in a C++ desktop (Win32) app

Article • 02/27/2025

> **Important**
>
> This topic uses or mentions types from the CommunityToolkit/Microsoft.Toolkit.Win32 GitHub repo. For important info about XAML Islands support, please see the XAML Islands Notice in that repo.

This article demonstrates how to use the WinRT XAML hosting API to host a custom WinRT XAML control in a new C++ desktop app. If you have an existing C++ desktop app project, you can adapt these steps and code examples for your project.

To host a custom WinRT XAML control, you'll create the following projects and components as part of this walkthrough:

- **Windows Desktop Application project**. This project implements a native C++ desktop app. You'll add code to this project that uses the WinRT XAML hosting API to host a custom WinRT XAML control.
- **UWP app project (C++/WinRT)**. This project implements a custom WinRT XAML control. It also implements a root metadata provider for loading metadata for custom WinRT XAML types in the project.

Requirements

- Visual Studio 2019 version 16.4.3 or later.
- Windows 10, version 1903 SDK (version 10.0.18362) or later.
- C++/WinRT Visual Studio Extension (VSIX) installed with Visual Studio. C++/WinRT is an entirely standard modern C++17 language projection for Windows Runtime (WinRT) APIs, implemented as a header-file-based library, and designed to provide you with first-class access to the modern Windows API. For more information, see C++/WinRT.

Create a desktop application project

1. In Visual Studio, create a new **Windows Desktop Application** project named **MyDesktopWin32App**. This project template is available under the **C++**, **Windows**,

and **Desktop** project filters.

2. In **Solution Explorer**, right-click the solution node, click **Retarget solution**, select the **10.0.18362.0** or a later SDK release, and then click **OK**.

3. Install the Microsoft.Windows.CppWinRT NuGet package to enable support for C++/WinRT in your project:
   a. Right-click the **MyDesktopWin32App** project in **Solution Explorer** and choose **Manage NuGet Packages**.
   b. Select the **Browse** tab, search for the Microsoft.Windows.CppWinRT package, and install the latest version of this package.

4. In the **Manage NuGet Packages** window, install the following additional NuGet packages:

   - Microsoft.Toolkit.Win32.UI.SDK (latest stable version). This package provides several build and run time assets that enable XAML Islands to work in your app.
   - Microsoft.Toolkit.Win32.UI.XamlApplication (latest stable version). This package defines the Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication class, which you will use later in this walkthrough.
   - Microsoft.VCRTForwarders.140.

5. Add a reference to the Windows Runtime metadata:
   a. In **Solution Explorer**, right-click on your project **References** node and select **Add Reference**.
   b. Click the **Browse** button at the bottom of the page and navigate to the UnionMetadata folder in your SDK install path. By default the SDK will be installed to `C:\Program Files (x86)\Windows Kits\10\UnionMetadata`.
   c. Then, select the folder named after the Windows version you are targetting (e.g. 10.0.18362.0) and inside of that folder pick the `Windows.winmd` file.
   d. Click **OK** to close the **Add Reference** dialog.

6. Build the solution and confirm that it builds successfully.

Create a UWP app project

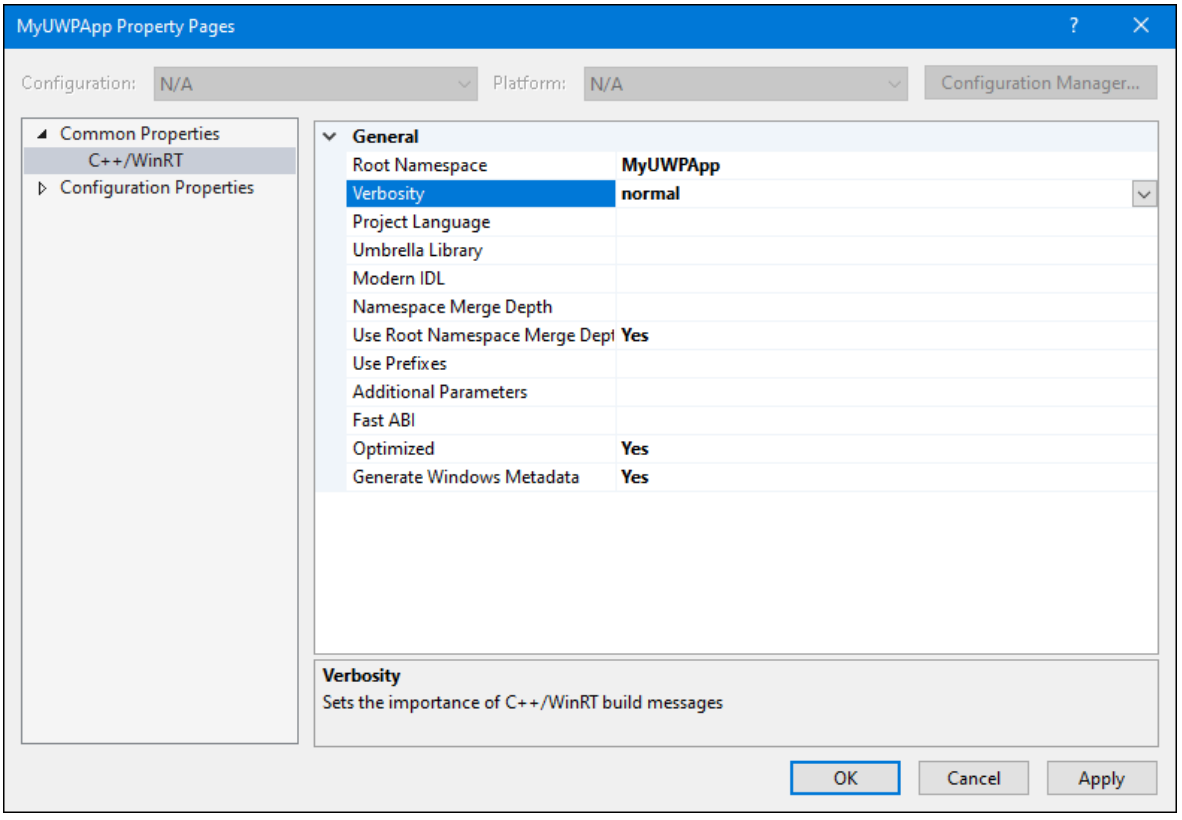
Next, add a **UWP (C++/WinRT)** app project to your solution and make some configuration changes to this project. Later in this walkthrough, you'll add code to this project to implement a custom WinRT XAML control and define an instance of the Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication class.

1. In **Solution Explorer**, right-click the solution node and select **Add** -> **New Project**.

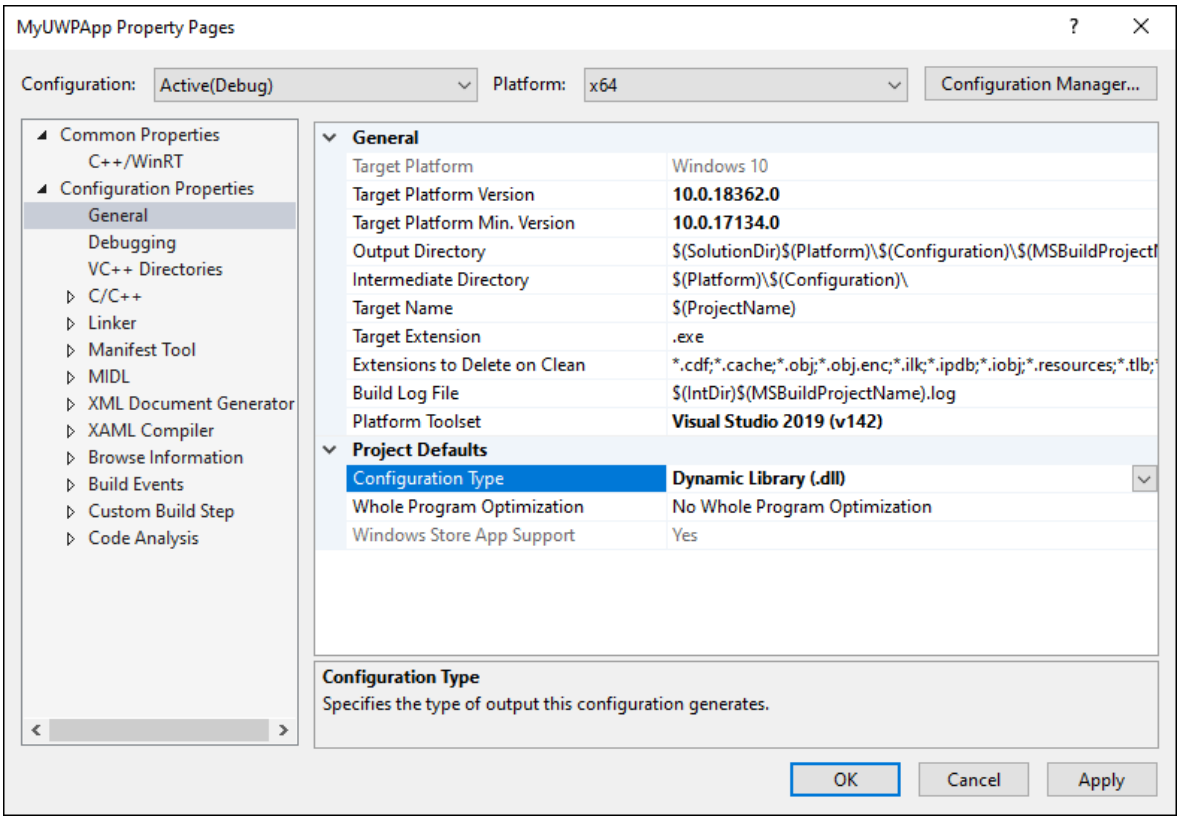
2. Add a **Blank App (C++/WinRT)** project to your solution. Name the project **MyUWPApp** and make sure the target version and minimum version are both set to **Windows 10, version 1903** or later.

3. Install the Microsoft.Toolkit.Win32.UI.XamlApplication NuGet package in the **MyUWPApp** project. This package defines the Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication class, which you will use later in this walkthrough.
   a. Right-click the **MyUWPApp** project and choose **Manage NuGet Packages**.
   b. Select the **Browse** tab, search for the Microsoft.Toolkit.Win32.UI.XamlApplication package, and install the latest stable version of this package.

4. Right-click the **MyUWPApp** node and select **Properties**. On the **Common Properties** -> **C++/WinRT** page, set the **Verbosity** property to **normal** and then click **Apply**. When you are done, the properties page should look like this.



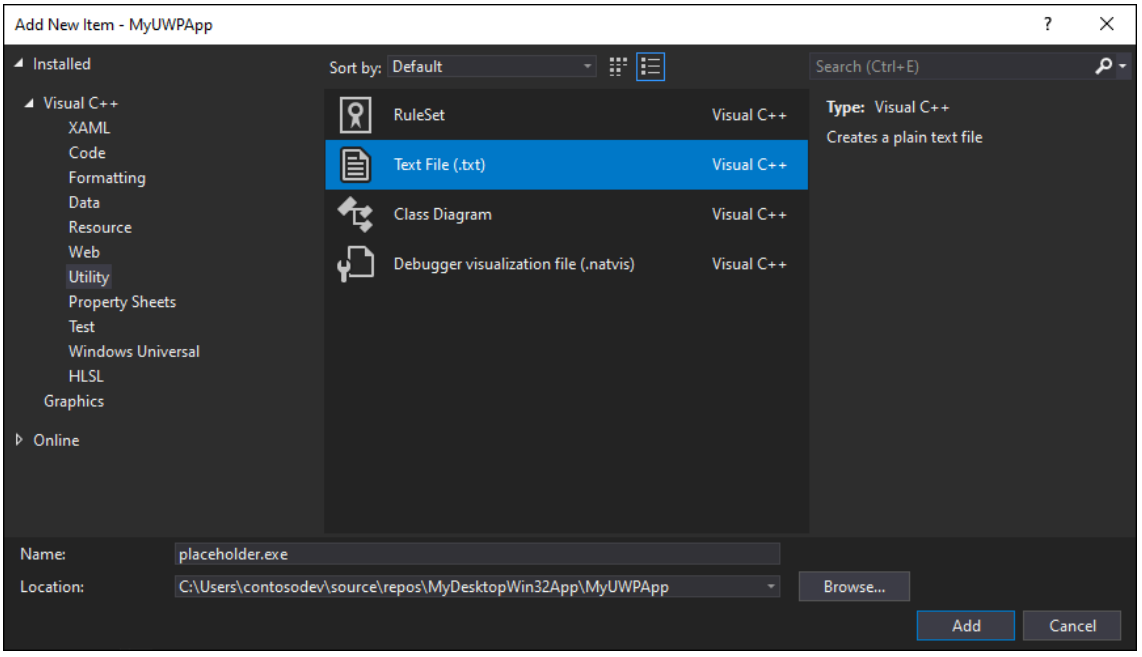
5. On the **Configuration Properties** -> **General** page of the properties window, set **Configuration Type** to **Dynamic Library (.dll)**, and then click **OK** to close the properties window.



6. Add a placeholder executable file to the **MyUWPApp** project. This placeholder executable file is required for Visual Studio to generate the required project files and properly build the project.

   a. In **Solution Explorer**, right-click the **MyUWPApp** project node and select **Add -> New Item**.

   b. In the **Add New Item** dialog, select **Utility** in the left page, and then select **Text File (.txt)**. Enter the name **placeholder.exe** and click **Add**.



   c. In **Solution Explorer**, select the **placeholder.exe** file. In the **Properties** window, make sure the **Content** property is set to **True**.

d. In **Solution Explorer**, right-click the **Package.appxmanifest** file in the **MyUWPApp** project, select **Open With**, and select **XML (Text) Editor**, and click **OK**.

e. Find the **<Application>** element and change the **Executable** attribute to the value `placeholder.exe`. When you are done, the **<Application>** element should look similar to this.

```
XML

<Application Id="App" Executable="placeholder.exe"
EntryPoint="MyUWPApp.App">
  <uap:VisualElements DisplayName="MyUWPApp" Description="Project
for a single page C++/WinRT Universal Windows Platform (UWP) app
with no predefined layout"
    Square150x150Logo="Assets\Square150x150Logo.png"
Square44x44Logo="Assets\Square44x44Logo.png"
BackgroundColor="transparent">
    <uap:DefaultTile Wide310x150Logo="Assets\Wide310x150Logo.png">
    </uap:DefaultTile>
    <uap:SplashScreen Image="Assets\SplashScreen.png" />
  </uap:VisualElements>
</Application>
```

f. Save and close the **Package.appxmanifest** file.

7. In **Solution Explorer**, right-click the **MyUWPApp** node and select **Unload Project**.

8. Right-click the **MyUWPApp** node and select **Edit MyUWPApp.vcxproj**.

9. Find the `<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />` element and replace it with the following XML. This XML adds several new properties immediately before the element.

```
XML

<PropertyGroup Label="Globals">
    <WindowsAppContainer>true</WindowsAppContainer>
    <AppxGeneratePriEnabled>true</AppxGeneratePriEnabled>
    <ProjectPriIndexName>App</ProjectPriIndexName>
    <AppxPackage>true</AppxPackage>
</PropertyGroup>
<Import Project="$(VCTargetsPath)\Microsoft.Cpp.Default.props" />
```

10. Save and close the project file.

11. In **Solution Explorer**, right-click the **MyUWPApp** node and select **Reload Project**.

Configure the solution

In this section, you'll update the solution that contains both projects to configure project dependencies and build properties required for the projects to build correctly.

1. In **Solution Explorer**, right-click the solution node and add a new XML file named **Solution.props**.

2. Add the following XML to the **Solution.props** file.

```XML
<?xml version="1.0" encoding="utf-8"?>
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>

<IntDir>$(SolutionDir)\obj\$(Platform)\$(Configuration)\$(MSBuildProjec
tName)\</IntDir>

<OutDir>$(SolutionDir)\bin\$(Platform)\$(Configuration)\$(MSBuildProjec
tName)\</OutDir>
    <GeneratedFilesDir>$(IntDir)Generated Files\</GeneratedFilesDir>
  </PropertyGroup>
</Project>
```

3. From the **View** menu, click **Property Manager** (depending on your configuration, this may be under **View** -> **Other Windows**).

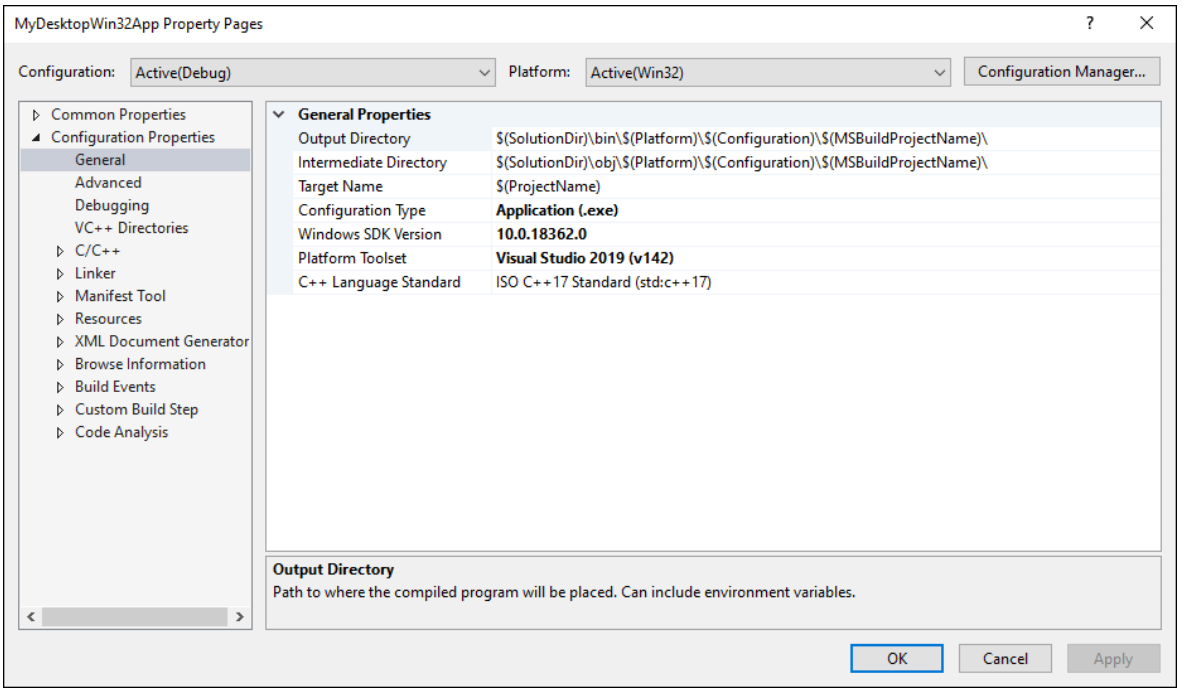
4. In the **Property Manager** window, right-click **MyDesktopWin32App** and select **Add Existing Property Sheet**. Navigate to the **Solution.props** file you just added and click **Open**.

5. Repeat the previous step to add the **Solution.props** file to the **MyUWPApp** project in the **Property Manager** window.

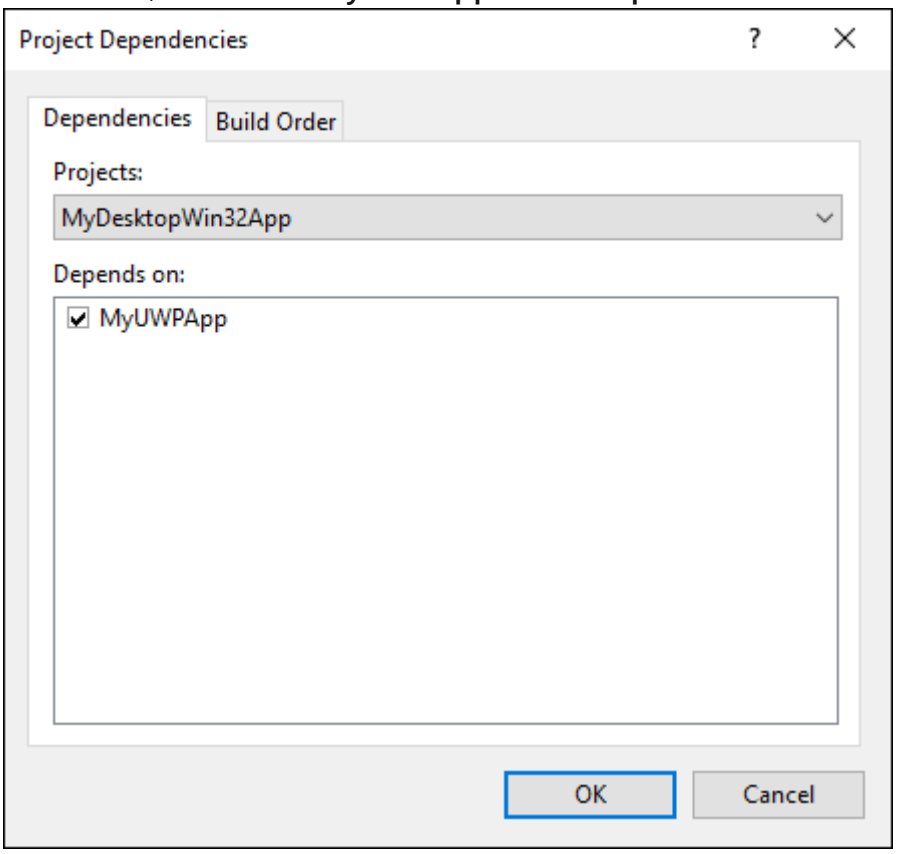
6. Close the **Property Manager** window.

7. Confirm that the property sheet changes were saved properly. In **Solution Explorer**, right-click the **MyDesktopWin32App** project and choose **Properties**. Click **Configuration Properties** -> **General**, and confirm that the **Output Directory** and **Intermediate Directory** properties have the values you added to the

Solution.props file. You can also confirm the same for the **MyUWPApp** project.



8. In **Solution Explorer**, right-click the solution node and choose **Project Dependencies**. In the **Projects** drop-down, make sure that **MyDesktopWin32App** is selected, and select **MyUWPApp** in the **Depends On** list.



9. Click **OK**.

Add code to the UWP app project

You're now ready to add code to the **MyUWPApp** project to perform these tasks:

- Implement a custom WinRT XAML control. Later in this walkthrough, you'll add code that hosts this control in the **MyDesktopWin32App** project.
- Define a type that derives from the Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication class in the Windows Community Toolkit.

Define a custom WinRT XAML control

1. In **Solution Explorer**, right-click **MyUWPApp** and select **Add** -> **New Item**. Select the **Visual C++** node in the left pane, select **Blank User Control (C++/WinRT)**, name it **MyUserControl**, and click **Add**.

2. In the XAML editor, replace the contents of the **MyUserControl.xaml** file with the following XAML and then save the file.

XML

```
<UserControl
    x:Class="MyUWPApp.MyUserControl"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MyUWPApp"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
    mc:Ignorable="d">

    <StackPanel HorizontalAlignment="Center" Spacing="10"
                Padding="20" VerticalAlignment="Center">
        <TextBlock HorizontalAlignment="Center" TextWrapping="Wrap"
                       Text="Hello from XAML Islands" FontSize="30" />
        <TextBlock HorizontalAlignment="Center" Margin="15"
TextWrapping="Wrap"
                       Text="😍❤💋🌹🎉😎🏴" FontSize="16" />
        <Button HorizontalAlignment="Center"
                x:Name="Button" Click="ClickHandler">Click Me</Button>
    </StackPanel>
</UserControl>
```

Define a XamlApplication class

Next, revise the default **App** class in the **MyUWPApp** project to derive from the Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication class provided by the Windows Community Toolkit. This class supports the IXamlMetadataProvider interface, which enables your app to discover and load metadata for custom WinRT XAML controls in assemblies in the current directory of your application at run time. This class also

initializes the WinRT XAML framework for the current thread. Later in this walkthrough you'll update the desktop project to create an instance of this class.

> ⓘ **Note**
>
> Each solution that uses XAML Islands can contain only one project that defines a `XamlApplication` object. All custom WinRT XAML controls in your app share the same `XamlApplication` object.

1. In **Solution Explorer**, right-click the **MainPage.xaml** file in the **MyUWPApp** project. Click **Remove** and then **Delete** to delete this file permanently from the project.
2. In the **MyUWPApp** project, expand **App.xaml** file.
3. Replace the contents of the **App.xaml**, **App.cpp**, **App.h**, and **App.idl** files with the following code.

   - **App.xaml**:

     XML

     ```
     <Toolkit:XamlApplication
         x:Class="MyUWPApp.App"
         xmlns:Toolkit="using:Microsoft.Toolkit.Win32.UI.XamlHost"

     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
         xmlns:local="using:MyUWPApp">
     </Toolkit:XamlApplication>
     ```

   - **App.idl**:

     IDL

     ```
     namespace MyUWPApp
     {
          [default_interface]
          runtimeclass App :
     Microsoft.Toolkit.Win32.UI.XamlHost.XamlApplication
          {
             App();
          }
     }
     ```

   - **App.h**:

```C++
#pragma once
#include "App.g.h"
#include "App.base.h"
namespace winrt::MyUWPApp::implementation
{
    class App : public AppT2<App>
    {
    public:
        App();
        ~App();
    };
}
namespace winrt::MyUWPApp::factory_implementation
{
    class App : public AppT<App, implementation::App>
    {
    };
}
```

- **App.cpp**:

```C++
#include "pch.h"
#include "App.h"
#include "App.g.cpp"
using namespace winrt;
using namespace Windows::UI::Xaml;
namespace winrt::MyUWPApp::implementation
{
    App::App()
    {
        Initialize();
        AddRef();
        m_inner.as<::IUnknown>()->Release();
    }
    App::~App()
    {
        Close();
    }
}
```

> ⓘ **Note**
>
> The `#include "App.g.cpp"` statement is necessary when the **Optimized** property on the **Common Properties** -> **C++/WinRT** page of the project properties is set to **Yes**. This is the default for new C++/WinRT projects.

> For more details about the effects of the **Optimized** property, see this section.

4. Add a new header file to the **MyUWPApp** project named **app.base.h**.

5. Add the following code to the **app.base.h** file, save the file, and close it.

```
C++
```

```
#pragma once
namespace winrt::MyUWPApp::implementation
{
    template <typename D, typename... I>
    struct App_baseWithProvider : public App_base<D,
::winrt::Windows::UI::Xaml::Markup::IXamlMetadataProvider>
    {
        using IXamlType =
::winrt::Windows::UI::Xaml::Markup::IXamlType;
        IXamlType
GetXamlType(::winrt::Windows::UI::Xaml::Interop::TypeName const& type)
        {
            return _appProvider.GetXamlType(type);
        }
        IXamlType GetXamlType(::winrt::hstring const& fullName)
        {
            return _appProvider.GetXamlType(fullName);
        }

::winrt::com_array<::winrt::Windows::UI::Xaml::Markup::XmlnsDefinition>
GetXmlnsDefinitions()
        {
            return _appProvider.GetXmlnsDefinitions();
        }
    private:
        bool _contentLoaded{ false };
        winrt::MyUWPApp::XamlMetaDataProvider _appProvider;
    };
    template <typename D, typename... I>
    using AppT2 = App_baseWithProvider<D, I...>;
}
```

6. Build the solution and confirm that it builds successfully.

Configure the desktop project to consume custom control types

Before the **MyDesktopWin32App** app can host a custom WinRT XAML control in a XAML Island, it must be configured to consume custom control types from the

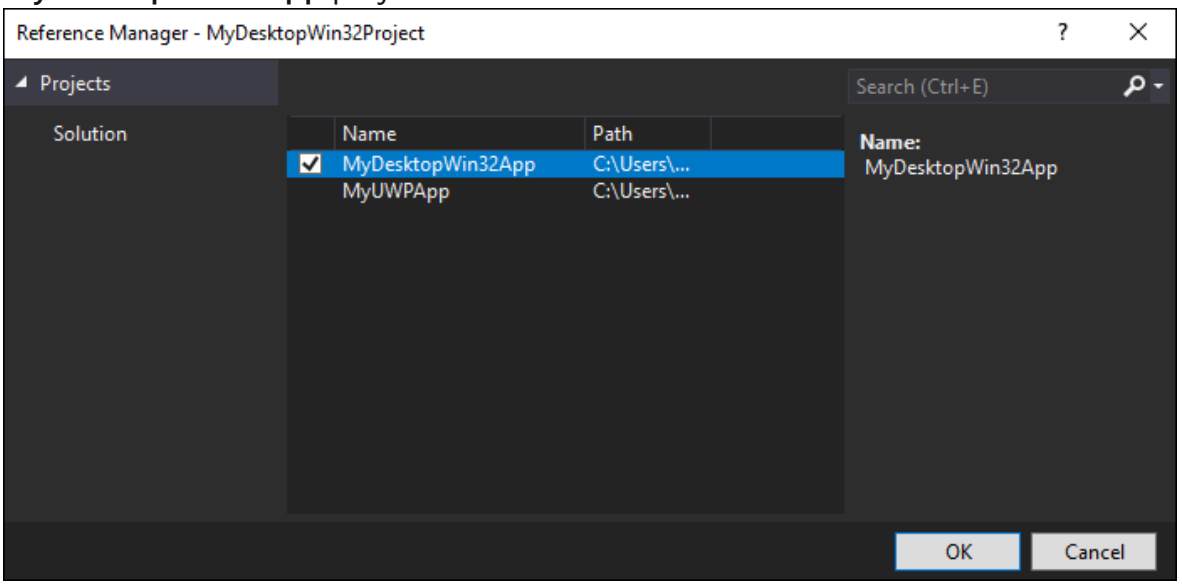
MyUWPApp project. There are two ways to do this, and you can choose either option as you complete this walkthrough.

Option 1: Package the app using MSIX

You can package the app in an MSIX package for deployment. MSIX is the modern app packaging technology for Windows, and it is based on a combination of MSI, .appx, App-V and ClickOnce installation technologies.

1. Add a new Windows Application Packaging Project to your solution. As you create the project, name it **MyDesktopWin32Project** and select **Windows 10, version 1903 (10.0; Build 18362)** for both the **Target version** and **Minimum version**.

2. In the packaging project, right-click the **Applications** node and choose **Add reference**. In the list of projects, select the check box next to the **MyDesktopWin32App** project and click **OK**.



3. For info about distributing/deploying the package, see Manage your MSIX deployment.

> ⓘ **Note**
>
> If you choose to not package your application in an MSIX package for deployment, then computers that run your app must have the Visual C++ Runtime installed.

Option 2: Create an application manifest

You can add an application manifest to your app.

1. Right-click the **MyDesktopWin32App** project and select **Add** -> **New Item**.

2. In the **Add New Item** dialog, click **Web** in the left pane and select **XML File (.xml)**.

3. Name the new file **app.manifest** and click **Add**.

4. Replace the contents of the new file with the following XML. This XML registers custom control types in the **MyUWPApp** project.

XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<assembly
 xmlns="urn:schemas-microsoft-com:asm.v1"
 xmlns:asmv3="urn:schemas-microsoft-com:asm.v3"
 manifestVersion="1.0">
  <asmv3:file name="MyUWPApp.dll">
    <activatableClass
        name="MyUWPApp.App"
        threadingModel="both"
        xmlns="urn:schemas-microsoft-com:winrt.v1" />
    <activatableClass
        name="MyUWPApp.XamlMetadataProvider"
        threadingModel="both"
        xmlns="urn:schemas-microsoft-com:winrt.v1" />
    <activatableClass
        name="MyUWPApp.MyUserControl"
        threadingModel="both"
        xmlns="urn:schemas-microsoft-com:winrt.v1" />
  </asmv3:file>
</assembly>
```

Configure additional desktop project properties

Next, update the **MyDesktopWin32App** project to define a macro for additional include directories and configure additional properties.

1. In **Solution Explorer**, right-click the **MyDesktopWin32App** project and select **Unload Project**.

2. Right-click **MyDesktopWin32App (Unloaded)** and select **Edit MyDesktopWin32App.vcxproj**.

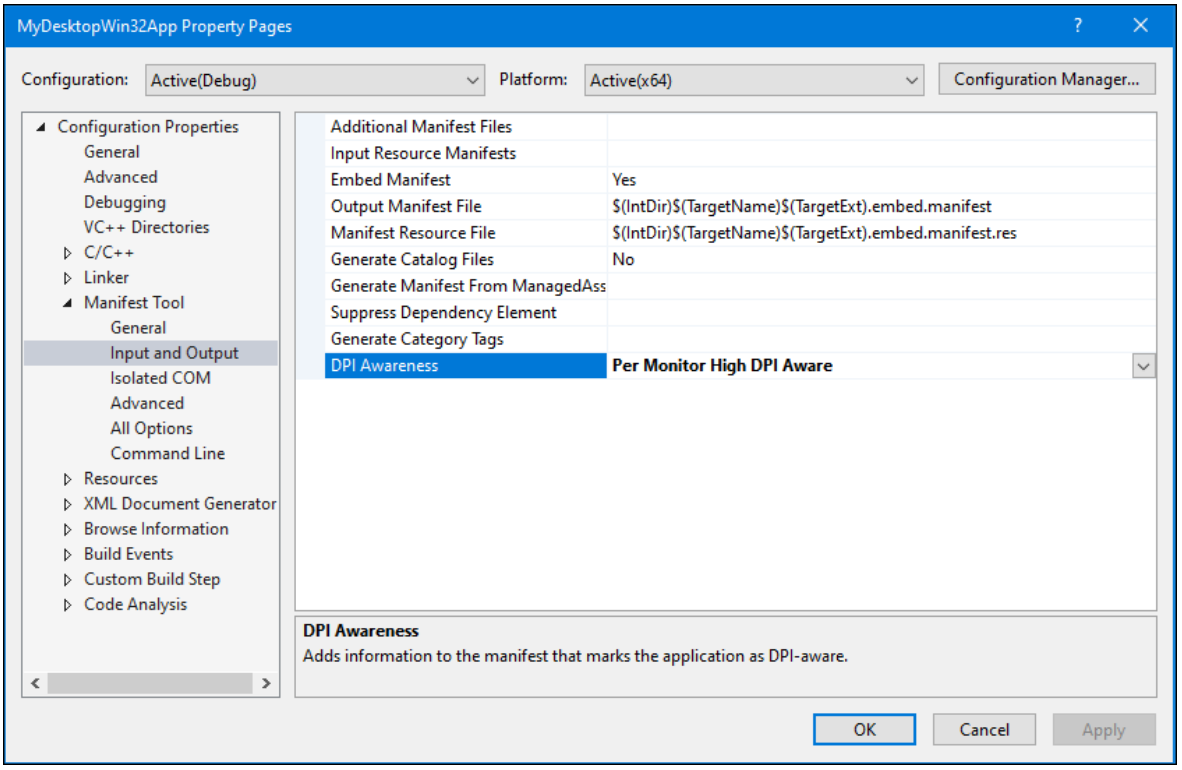
3. Add the following XML just before the closing `</Project>` tag at the end of the file. Then, save and close the file.

XML

```
  <!-- Configure these for your UWP project -->
  <PropertyGroup>
    <AppProjectName>MyUWPApp</AppProjectName>
  </PropertyGroup>
  <PropertyGroup>

<AppIncludeDirectories>$(SolutionDir)\obj\$(Platform)\$(Configuration)\
$(AppProjectName)\;$(SolutionDir)\obj\$(Platform)\$(Configuration)\$(Ap
pProjectName)\Generated Files\;</AppIncludeDirectories>
  </PropertyGroup>
  <ItemGroup>
    <ProjectReference
Include="..\$(AppProjectName)\$(AppProjectName).vcxproj" />
  </ItemGroup>
  <!-- End Section-->
```

4. In **Solution Explorer**, right-click **MyDesktopWin32App (unloaded)** and select **Reload Project**.

5. Right-click the **MyDesktopWin32App** project, select **Properties**, and expand **Manifest Tool** -> **Input and Output** in the left pane. Set the **DPI Awareness** property to **Per Monitor High DPI Aware**. If you do not set this property, you may encounter a manifest configuration error in certain high DPI scenarios.



6. Click **OK** to close the **Property Pages** dialog.

Host the custom WinRT XAML control in the desktop project

Finally, you're ready to add code to the **MyDesktopWin32App** project to host the custom WinRT XAML control you defined earlier in the **MyUWPApp** project.

1. In the **MyDesktopWin32App** project, open the **framework.h** file and comment out the following line of code. Save the file when you're done.

```
C++

#define WIN32_LEAN_AND_MEAN
```

2. Open the **MyDesktopWin32App.h** file and replace the contents of this file with the following code to reference necessary C++/WinRT header files. Save the file when you're done.

```
C++

#pragma once

#include "resource.h"
#include <winrt/Windows.Foundation.Collections.h>
#include <winrt/Windows.system.h>
#include <winrt/windows.ui.xaml.hosting.h>
#include <windows.ui.xaml.hosting.desktopwindowxamlsource.h>
#include <winrt/windows.ui.xaml.controls.h>
#include <winrt/Windows.ui.xaml.media.h>
#include <winrt/Windows.UI.Core.h>
#include <winrt/MyUWPApp.h>

using namespace winrt;
using namespace Windows::UI;
using namespace Windows::UI::Composition;
using namespace Windows::UI::Xaml::Hosting;
using namespace Windows::Foundation::Numerics;
using namespace Windows::UI::Xaml::Controls;
```

3. Open the **MyDesktopWin32App.cpp** file and add the following code to the `Global Variables:` section.

```
C++

winrt::MyUWPApp::App hostApp{ nullptr };
winrt::Windows::UI::Xaml::Hosting::DesktopWindowXamlSource
```

```
_desktopWindowXamlSource{ nullptr };
winrt::MyUWPApp::MyUserControl _myUserControl{ nullptr };
```

4. In the same file, add the following code to the `Forward declarations of functions included in this code module:` section.

```
C++

void AdjustLayout(HWND);
```

5. In the same file, add the following code immediately after the `TODO: Place code here.` comment in the `wWinMain` function.

```
C++

// TODO: Place code here.
winrt::init_apartment(winrt::apartment_type::single_threaded);
hostApp = winrt::MyUWPApp::App{};
_desktopWindowXamlSource =
winrt::Windows::UI::Xaml::Hosting::DesktopWindowXamlSource{};
```

6. In the same file, replace the default `InitInstance` function with the following code.

```
C++

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hInst = hInstance; // Store instance handle in our global variable

    HWND hWnd = CreateWindowW(szWindowClass, szTitle,
WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr,
hInstance, nullptr);

    if (!hWnd)
    {
        return FALSE;
    }

    // Begin XAML Islands walkthrough code.
    if (_desktopWindowXamlSource != nullptr)
    {
        auto interop =
_desktopWindowXamlSource.as<IDesktopWindowXamlSourceNative>();
        check_hresult(interop->AttachToWindow(hWnd));
        HWND hWndXamlIsland = nullptr;
        interop->get_WindowHandle(&hWndXamlIsland);
        RECT windowRect;
        ::GetWindowRect(hWnd, &windowRect);
```

```
        ::SetWindowPos(hWndXamlIsland, NULL, 0, 0, windowRect.right -
windowRect.left, windowRect.bottom - windowRect.top, SWP_SHOWWINDOW);
        _myUserControl = winrt::MyUWPApp::MyUserControl();
        _desktopWindowXamlSource.Content(_myUserControl);
    }
    // End XAML Islands walkthrough code.

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    return TRUE;
}
```

7. In the same file, add the following new function to the end of the file.

C++

```
void AdjustLayout(HWND hWnd)
{
    if (_desktopWindowXamlSource != nullptr)
    {
        auto interop =
_desktopWindowXamlSource.as<IDesktopWindowXamlSourceNative>();
        HWND xamlHostHwnd = NULL;
        check_hresult(interop->get_WindowHandle(&xamlHostHwnd));
        RECT windowRect;
        ::GetWindowRect(hWnd, &windowRect);
        ::SetWindowPos(xamlHostHwnd, NULL, 0, 0, windowRect.right -
windowRect.left, windowRect.bottom - windowRect.top, SWP_SHOWWINDOW);
    }
}
```

8. In the same file, locate the `WndProc` function. Replace the default `WM_DESTROY` handler in the switch statement with the following code.

C++

```
case WM_DESTROY:
    PostQuitMessage(0);
    if (_desktopWindowXamlSource != nullptr)
    {
        _desktopWindowXamlSource.Close();
        _desktopWindowXamlSource = nullptr;
    }
    break;
case WM_SIZE:
    AdjustLayout(hWnd);
    break;
```

9. Save the file.

10. Build the solution and confirm that it builds successfully.

Add a control from the WinUI 2 library to the custom control

Traditionally, WinRT XAML controls have been released as part of the Windows OS and made available to developers through the Windows SDK. The WinUI library is an alternative approach, where updated versions of WinRT XAML controls from the Windows SDK are distributed in a NuGet package that is not tied to Windows SDK releases. This library also includes new controls that aren't part of the Windows SDK and the default UWP platform.

This section demonstrates how to add a WinRT XAML control from the WinUI 2 library to your user control.

> **Note**
>
> Currently, XAML Islands only supports hosting controls from the WinUI 2 library. Support for hosting controls from the WinUI 3 library is coming in a later release.

1. In the **MyUWPApp** project, install the latest prerelease or release version of the Microsoft.UI.Xaml NuGet package.

    - If you chose to package the MyDesktopWin32App project using MSIX earlier in this walkthrough, you can install either the prerelease or release version of the Microsoft.UI.Xaml NugGet package. Packaged desktop apps can use either the prerelease or release version of this package.
    - If you chose not to package the **MyDesktopWin32App** project, you must install the prerelease version of the Microsoft.UI.Xaml NuGet package. Unpackaged desktop apps must use the prerelease version of this package.

2. In the pch.h file in this project, add the following `#include` statements and save your changes. These statements bring a required set of projection headers from the WinUI library into your project. This step is required for any C++/WinRT project that uses the WinUI library. For more information, see this article.

    C++

    ```
    #include "winrt/Microsoft.UI.Xaml.Automation.Peers.h"
    #include "winrt/Microsoft.UI.Xaml.Controls.Primitives.h"
    ```

```
#include "winrt/Microsoft.UI.Xaml.Media.h"
#include "winrt/Microsoft.UI.Xaml.XamlTypeInfo.h"
```

3. In the App.xaml file in the same project, add the following child element to the `<xaml:XamlApplication>` element and save your changes.

```
XML

<Application.Resources>
    <XamlControlsResources xmlns="using:Microsoft.UI.Xaml.Controls" />
</Application.Resources>
```

After adding this element, the contents of this file should now look similar to this.

```
XML

<Toolkit:XamlApplication
    x:Class="MyUWPApp.App"
    xmlns:Toolkit="using:Microsoft.Toolkit.Win32.UI.XamlHost"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MyUWPApp">
    <Application.Resources>
        <XamlControlsResources
xmlns="using:Microsoft.UI.Xaml.Controls"/>
    </Application.Resources>
</Toolkit:XamlApplication>
```

4. In the same project, open the MyUserControl.xaml file and add the following namespace declaration to the `<UserControl>` element.

```
XML

xmlns:winui="using:Microsoft.UI.Xaml.Controls"
```

5. In the same file, add a `<winui:RatingControl />` element as a child of the `<StackPanel>` and save your changes. This element adds an instance of the RatingControl class from the WinUI library. After adding this element, the `<StackPanel>` should now look similar to this.

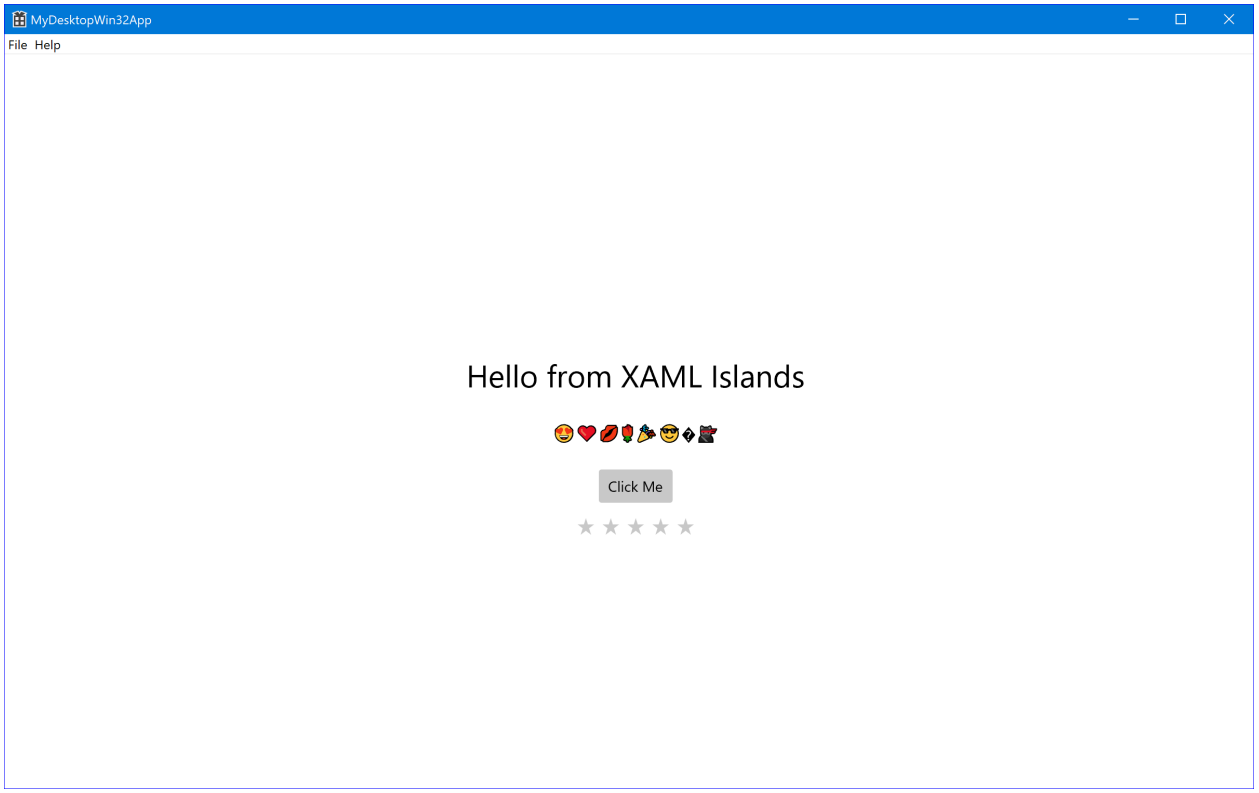
```
XML

<StackPanel HorizontalAlignment="Center" Spacing="10"
            Padding="20" VerticalAlignment="Center">
    <TextBlock HorizontalAlignment="Center" TextWrapping="Wrap"
                   Text="Hello from XAML Islands" FontSize="30" />
```

```
    <TextBlock HorizontalAlignment="Center" Margin="15"
TextWrapping="Wrap"
                    Text="😍❤️🌶️🌹🎉😎🔷🏴" FontSize="16" />
    <Button HorizontalAlignment="Center"
            x:Name="Button" Click="ClickHandler">Click Me</Button>
    <winui:RatingControl />
</StackPanel>
```

6. Build the solution and confirm that it builds successfully.

Test the app

Run the solution and confirm that **MyDesktopWin32App** opens with the following window.



Next steps

Many desktop applications that host XAML Islands will need to handle additional scenarios in order to provide a smooth user experience. For example, desktop applications may need to handle keyboard input in XAML Islands, focus navigation between XAML Islands and other UI elements, and layout changes.

For more information about handling these scenarios and pointers to related code samples, see Advanced scenarios for XAML Islands in C++ desktop apps.

Related topics

- Host WinRT XAML controls in desktop apps (XAML Islands)
- Using the WinRT XAML hosting API in a desktop Win32 app
- Host a standard WinRT XAML control in a desktop Win32 app
- Advanced scenarios for XAML Islands in desktop Win32 apps
- XAML Islands code samples

Advanced scenarios for XAML Islands in C++ desktop (Win32) apps

Article • 08/21/2024

> **Important**
>
> This topic uses or mentions types from the CommunityToolkit/Microsoft.Toolkit.Win32 GitHub repo. For important info about XAML Islands support, please see the XAML Islands Notice in that repo.

The host a standard UWP control and host a custom UWP control articles provide instructions and examples for hosting XAML Islands in a C++ desktop (Win32) app. However, the code examples in these articles do not handle many advanced scenarios that desktop applications may need to handle to provide a smooth user experience. This article provides guidance for some of these scenarios and pointers to related code samples.

Keyboard input

To properly handle keyboard input for each XAML Island, your application must pass all Windows messages to the UWP XAML framework so that certain messages can be processed correctly. To do this, in some place in your application that can access the message loop, cast the **DesktopWindowXamlSource** object for each XAML Island to an **IDesktopWindowXamlSourceNative2** COM interface. Then, call the **PreTranslateMessage** method of this interface and pass in the current message.

- **C++ desktop (Win32)**:: The app can call **PreTranslateMessage** directly in its main message loop. For an example, see the XamlBridge.cpp file.
- **WPF:** The app can call **PreTranslateMessage** from the event handler for the ComponentDispatcher.ThreadFilterMessage event. For an example, see the WindowsXamlHostBase.Focus.cs file in the Windows Community Toolkit.
- **Windows Forms:** The app can call **PreTranslateMessage** from an override for the Control.PreprocessMessage method. For an example, see the WindowsXamlHostBase.KeyboardFocus.cs file in the Windows Community Toolkit.

Keyboard focus navigation

When the user navigates through the UI elements in your application using the keyboard (for example, by pressing **Tab** or direction/arrow key), you'll need to programmatically move focus into and out of the **DesktopWindowXamlSource** object. When the user's keyboard navigation reaches the **DesktopWindowXamlSource**, move focus into the first Windows.UI.Xaml.UIElement object in the navigation order for your UI, continue to move focus to the following **Windows.UI.Xaml.UIElement** objects as the user cycles through the elements, and then move focus back out of the **DesktopWindowXamlSource** and into the parent UI element.

The UWP XAML hosting API provides several types and members to help you accomplish these tasks.

- When the keyboard navigation enters your **DesktopWindowXamlSource**, the GotFocus event is raised. Handle this event and programmatically move focus to the first hosted **Windows.UI.Xaml.UIElement** by using the NavigateFocus method.
- When the user is on the last focusable element in your **DesktopWindowXamlSource** and presses the **Tab** key or an arrow key, the TakeFocusRequested event is raised. Handle this event and programmatically move focus to the next focusable element in the host application. For example, in a WPF application where the **DesktopWindowXamlSource** is hosted in a System.Windows.Interop.HwndHost, you can use the MoveFocus method to transfer focus to the next focusable element in the host application.

For examples that demonstrate how to do this in the context of a working sample application, see the following code files:

- **C++ desktop (Win32)**: See the XamlBridge.cpp file.
- **WPF:** See the WindowsXamlHostBase.Focus.cs file in the Windows Community Toolkit.
- **Windows Forms:** See the WindowsXamlHostBase.KeyboardFocus.cs file in the Windows Community Toolkit.

Handle layout changes

When the user changes the size of the parent UI element, you'll need to handle any necessary layout changes to make sure your UWP controls display as you expect. Here are some important scenarios to consider.

- In a C++ desktop application, when your application handles the WM_SIZE message it can reposition the hosted XAML Island by using the SetWindowPos function. For an example, see the SampleApp.cpp code file.

- When the parent UI element needs to get the size of the rectangular area needed to fit the **Windows.UI.Xaml.UIElement** that you are hosting on the **DesktopWindowXamlSource**, call the Measure method of the **Windows.UI.Xaml.UIElement**. For example:

  - In a WPF application you might do this from the MeasureOverride method of the HwndHost that hosts the **DesktopWindowXamlSource**. For an example, see the WindowsXamlHostBase.Layout.cs file in the Windows Community Toolkit.

  - In a Windows Forms application you might do this from the GetPreferredSize method of the Control that hosts the **DesktopWindowXamlSource**. For an example, see the WindowsXamlHostBase.Layout.cs file in the Windows Community Toolkit.

- When the size of the parent UI element changes, call the Arrange method of the root **Windows.UI.Xaml.UIElement** that you are hosting on the **DesktopWindowXamlSource**. For example:

  - In a WPF application you might do this from the ArrangeOverride method of the HwndHost object that hosts the **DesktopWindowXamlSource**. For an example, see the WindowsXamlHost.Layout.cs file in the Windows Community Toolkit.

  - In a Windows Forms application you might do this from the handler for the SizeChanged event of the Control that hosts the **DesktopWindowXamlSource**. For an example, see the WindowsXamlHost.Layout.cs file in the Windows Community Toolkit.

Handle DPI changes

The UWP XAML framework handles DPI changes for hosted UWP controls automatically (for example, when the user drags the window between monitors with different screen DPI). For the best experience, we recommend that your Windows Forms, WPF, or C++ desktop application is configured to be per-monitor DPI aware.

To configure your application to be per-monitor DPI aware, add a side-by-side assembly manifest to your project and set the **<dpiAwareness>** element to **PerMonitorV2**. For more information about this value, see the description for DPI_AWARENESS_CONTEXT_PER_MONITOR_AWARE_V2.

XML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
    <application xmlns="urn:schemas-microsoft-com:asm.v3">
        <windowsSettings>
            <dpiAwareness
xmlns="http://schemas.microsoft.com/SMI/2016/WindowsSettings">PerMonitorV2</
dpiAwareness>
        </windowsSettings>
    </application>
</assembly>
```

Related topics

- Host UWP XAML controls in desktop apps (XAML Islands)
- Using the UWP XAML hosting API in a C++ desktop app
- Host a standard UWP control in a C++ desktop app
- Host a custom UWP control in a C++ desktop app
- XAML Islands code samples
- C++ desktop XAML Islands sample

Tutorial: Modernize a WPF app

Article • 05/13/2022

There are many ways to modernize existing desktop apps by integrating the latest Windows features into the existing source code instead of rewriting the apps from scratch. In this tutorial we'll explore several ways to modernize an existing WPF line-of-business app by using these features:

- .NET Core 3
- UWP XAML controls with XAML Islands
- Adaptive Cards and Windows notifications
- MSIX deployment

This tutorial requires the following development skills:

- Experience in developing Windows desktop apps with WPF.
- Basic knowledge of C# and XAML.
- Basic knowledge of UWP.

Overview

This tutorial provides the code for a simple WPF line-of-business app named Contoso Expenses. In the fictional scenario of the tutorial, Contoso Expenses is an internal app used by managers of Contoso Corporation to keep track of the expenses submitted by their reports. The managers are now equipped with touch-enabled devices, and they would like to use the Contoso Expenses app without a mouse or keyboard. Unfortunately, the current version of the app isn't touch friendly.

Contoso wants to modernize this app with new Windows features to enable employees to create expenses reports more efficiently. Many of the features could be easily implemented by building a new UWP app. However, the existing app is complex and is the result of many years of development by different teams. As such, rewriting it from scratch with a new technology isn't an option. The team is looking for the best approach to add new features into the existing codebase.

At the beginning of the tutorial, Contoso Expenses targets the .NET Framework 4.7.2 and uses the following 3rd party libraries:

- MVVM Light, a basic implementation for the MVVM pattern.
- Unity, a dependency injection container.
- LiteDb, an embedded NoSQL solution to store the data.

- Bogus, a tool to generate fake data.

In the tutorial, you'll enhance Contoso Expenses with new Windows features:

- Migrate an existing WPF app to .NET Core 3.0. This will open up new and important scenarios in the future.
- Use XAML Islands to host the **InkCanvas** and **MapControl** wrapped controls provided by the Windows Community Toolkit.
- Use XAML Islands to host any standard UWP XAML control (in this case, a **CalendarView**).
- Integrate Adaptive Cards and Windows notifications into the app.
- Package the app with MSIX and set up a CI/CD pipeline on Azure DevOps so that you can automatically deliver new versions of the app to testers and users as soon as it is available.

Prerequisites

To perform this tutorial, your development computer must have these prerequisites installed:

- Windows 10, version 1903 (build 18362) or a later version.
- Visual Studio 2019 .
- .NET Core 3 SDK (install the latest version).

Make sure you install the following workloads and optional features with Visual Studio 2019:

- .NET Desktop development
- Universal Windows Platform development
- Windows SDK (10.0.18362.0 or later)

Get the Contoso Expenses sample app

Before you begin the tutorial, download the source code for the Contoso Expenses app and make sure you can build the code in Visual Studio.

1. Download the app source code from the **Releases** tab of the AppConsult WinAppsModernization workshop repository . The direct link is https://github.com/microsoft/AppConsult-WinAppsModernizationWorkshop/releases.
2. Open the zip file and extract all the content to the root of your **C:** drive. It will create a folder named **C:\WinAppsModernizationWorkshop**.

3. Open Visual Studio 2019 and double click on the **C:\WinAppsModernizationWorkshop\Lab\Exercise1\01-Start\ContosoExpenses\ContosoExpenses.sln** file to open the solution.
4. Verify that you can build, run, and debug the Contoso Expenses WPF project by pressing the **Start** button or CTRL + F5.

Get started

After you have the source code for the Contoso Expenses sample app and you can confirm that you can build it in Visual Studio, you're ready to start the tutorial:

- Part 1: Migrate the Contoso Expenses app to .NET Core 3
- Part 2: Add a UWP InkCanvas control using XAML Islands
- Part 3: Add a UWP CalendarView control using XAML Islands
- Part 4: Add Windows 10 user activities and notifications
- Part 5: Package and deploy with MSIX

Important concepts

The following sections provide background for some of the key concepts discussed in this tutorial. If you're already familiar with these concepts, you can skip this section.

Universal Windows Platform (UWP)

In Windows 8, Microsoft introduced a new API set as part of the Windows Runtime (WinRT). Unlike the .NET Framework, WinRT is a native layer of APIs which are exposed directly to apps. WinRT also introduced language projections, which are layers added on top of the runtime to allow developers to interact with it using languages such as C# and JavaScript in addition to C++. Projections enable developers to build apps on top of WinRT that leverage the same C# and XAML knowledge they acquired in building apps with the .NET Framework.

In Windows 10, Microsoft introduced the Universal Windows Platform (UWP), which is built on top of WinRT. The most important feature of UWP is that it offers a common set of APIs across every device platform: no matter if the app is running on a desktop, on an Xbox One or on a HoloLens, you're able to use the same APIs.

Going forward, most new Windows features are exposed via WinRT APIs, including features such as Timeline, Project Rome, and Windows Hello.

MSIX packaging

MSIX is the modern packaging model for Windows apps. MSIX supports UWP apps as well as desktop apps building using technologies such as Win32, WPF, Windows Forms, Java, Electron, and more. When you package a desktop app in an MSIX package, you can publish your app to the Microsoft Store. Your desktop app also get package identity when it is installed, which enables your desktop app to use a broader set of WinRT APIs.

For more information, see these articles:

- Package desktop applications
- Behind the scenes of your packaged desktop application

XAML Islands

Starting in Windows 10, version 1903, you can host UWP controls in non-UWP desktop apps using a feature called *XAML Islands*. This feature enables you to enhance the look, feel, and functionality of your existing desktop apps with the latest Windows UI features that are only available via UWP controls. This means that you can use UWP features such as Windows Ink and controls that support the Fluent Design System in your existing WPF, Windows Forms, and C++ Win32 apps.

For more information, see UWP controls in desktop applications (XAML Islands). This tutorial guides you through the process of using two different types of XAML Island controls:

- The InkCanvas and MapControl in the Windows Community Toolkit. These WPF controls wrap the interface and functionality of the corresponding UWP controls and can be used like any other WPF control in the Visual Studio designer.
- The UWP Calendar view control. This is a standard UWP control that you will host by using the WindowsXamlHost control in the Windows Community Toolkit.

.NET Core 3

.NET Core is an open-source framework that implements a cross-platform, lightweight and easily extensible version of the full .NET Framework. Compared to the full .NET Framework, .NET Core startup time is much faster and many of the APIs have been optimized.

Through its first several releases, the focus of .NET Core was for supporting web or back-end apps. With .NET Core, you can easily build scalable web apps or APIs that can be hosted on Windows, Linux, or in micro-service architectures like Docker containers.

.NET Core 3 is latest release of .NET Core. The highlight of this release is support for Windows desktop apps, including Windows Forms and WPF apps. You can run new and existing Windows desktop apps on .NET Core 3 and enjoy all the benefits that .NET Core has to offer. UWP controls that are hosted in XAML Islands can also be used in Windows Forms and WPF apps that target .NET Core 3.

> **Note**
>
> WPF and Windows Forms are not becoming cross-platform, and you cannot run a WPF or Windows Forms on Linux and MacOS. The UI components of WPF and Windows Forms still have a dependency on the Windows rendering system.

For more information, see What's new in .NET Core 3.0.

Part 1: Migrate the Contoso Expenses app to .NET Core 3

Article • 10/16/2021

This is the first part of a tutorial that demonstrates how to modernize a sample WPF desktop app named Contoso Expenses. For an overview of the tutorial, prerequisites, and instructions for downloading the sample app, see Tutorial: Modernize a WPF app.

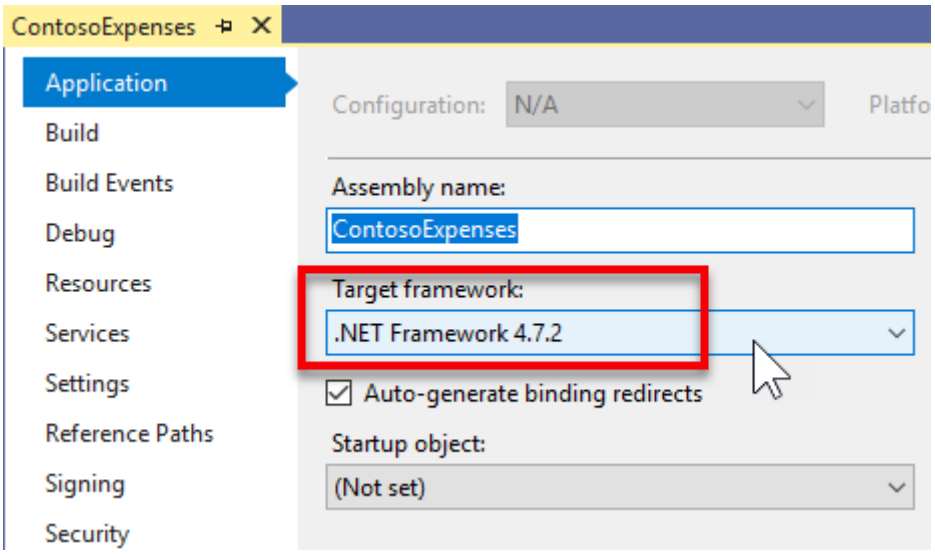
In this part of the tutorial, you will migrate the entire Contoso Expenses app from the .NET Framework 4.7.2 to .NET Core 3. Before you start this part of the tutorial, make sure you open and build the ContosoExpenses sample in Visual Studio 2019.

> **Note**
>
> For more information about migrating a WPF application from the .NET Framework to .NET Core 3, see this blog series.

Migrate the ContosoExpenses project to .NET Core 3

In this section, you'll migrate the ContosoExpenses project in the Contoso Expenses app to .NET Core 3. You'll do this by creating a new project file that contains the same files as the existing ContosoExpenses project but targets .NET Core 3 instead of the .NET Framework 4.7.2. This enables you to maintain a single solution with both .NET Framework and .NET Core versions of the app.

1. Verify that the ContosoExpenses project currently targets the .NET Framework 4.7.2. In Solution Explorer, right-click the **ContosoExpenses** project, choose **Properties**, and confirm that the **Target framework** property on the **Application** tab is set to the .NET Framework 4.7.2.



2. In Windows Explorer, navigate to the **C:\WinAppsModernizationWorkshop\Lab\Exercise1\01-Start\ContosoExpenses** folder and create a new text file named **ContosoExpenses.Core.csproj**.

3. Right click on the file, choose **Open with**, and then open it in a text editor of your choice, such as Notepad, Visual Studio Code or Visual Studio.

4. Copy the following text to the file and save it.

```xml
<Project Sdk="Microsoft.NET.Sdk.WindowsDesktop">

  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>netcoreapp3.0</TargetFramework>
    <UseWPF>true</UseWPF>
 </PropertyGroup>

</Project>
```

5. Close the file and return to the **ContosoExpenses** solution in Visual Studio.

6. Right-click the **ContosoExpenses** solution and choose **Add -> Existing Project**. Select the **ContosoExpenses.Core.csproj** file you just created in the `C:\WinAppsModernizationWorkshop\Lab\Exercise1\01-Start\ContosoExpenses` folder to add it to the solution.

The **ContosoExpenses.Core.csproj** includes the following elements:

- The **Project** element specifies an SDK version of **Microsoft.NET.Sdk.WindowsDesktop**. This refers to .NET applications for Windows Desktop, and it includes components for WPF and Windows Forms apps.

- The **PropertyGroup** element contains child elements that indicate the project output is an executable (not a DLL), targets .NET Core 3, and uses WPF. For a Windows Forms app, you would use a **UseWinForms** element instead of the **UseWPF** element.

> **Note**
>
> When working with the .csproj format introduced with .NET Core 3.0, all the files in the same folder as the .csproj are considered part of the project. Therefore, you don't have to specify every file included in the project. You must specify only those files for which you want to define a custom build action or that you want to exclude.

Migrate the ContosoExpenses.Data project to .NET Standard

The **ContosoExpenses** solution includes a **ContosoExpenses.Data** class library that contains models and interfaces for services and targets .NET 4.7.2. .NET Core 3.0 apps can use .NET Framework libraries, as long as they don't use APIs which aren't available in .NET Core. However, the best modernization path is to move your libraries to .NET Standard. This will make sure that your library is fully supported by your .NET Core 3.0 app. Additionally, you can reuse the library also with other platforms, such as web (through ASP.NET Core).

To migrate the **ContosoExpenses.Data** project to .NET Standard:

1. In Visual Studio, right-click the **ContosoExpenses.Data** project and choose **Unload Project**. Right-click the project again and then choose **Edit ContosoExpenses.Data.csproj**.

2. Delete the entire contents of the project file.

3. Copy and paste the following XML and save the file.

   XML

   ```xml
   <Project Sdk="Microsoft.NET.Sdk">

     <PropertyGroup>
       <TargetFramework>netstandard2.0</TargetFramework>
     </PropertyGroup>
   ```

```
</Project>
```

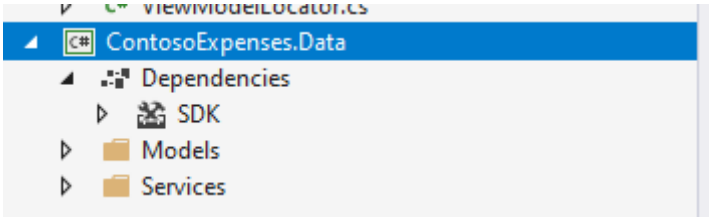
4. Right-click the **ContosoExpenses.Data** project and choose **Reload Project**.

Configure NuGet packages and dependencies

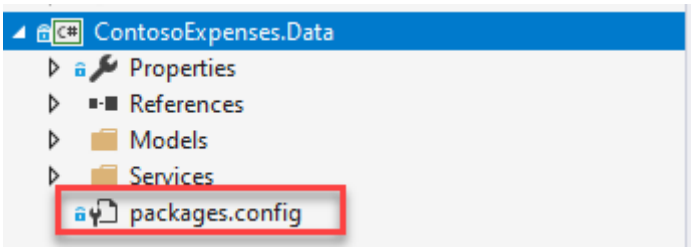
When you migrated the **ContosoExpenses.Core** and **ContosoExpenses.Data** projects in the previous sections, you removed the NuGet package references from the projects. In this section, you'll add these references back.

To configure NuGet packages for the **ContosoExpenses.Data** project:

1. In the **ContosoExpenses.Data** project, expand the **Dependencies** node. Note that the **NuGet** section is missing.



   If you open the **Packages.config** in the **Solution Explorer** you will find the 'old' references of the NuGet packages used the project when it was using the full .NET Framework.



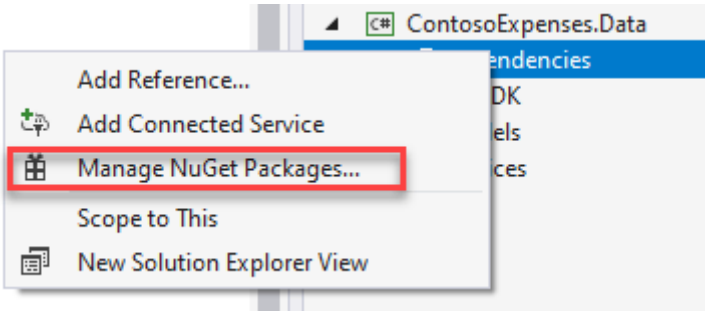
   Here is the content of the **Packages.config** file. You will notice that all the NuGet Packages target the Full .NET Framework 4.7.2:

   XML

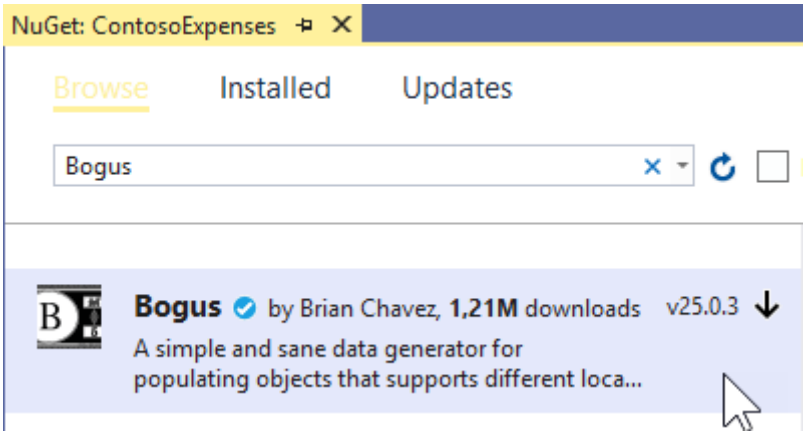
```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Bogus" version="26.0.2" targetFramework="net472" />
  <package id="LiteDB" version="4.1.4" targetFramework="net472" />
</packages>
```

2. In the **ContosoExpenses.Data** project, delete the **Packages.config** file.

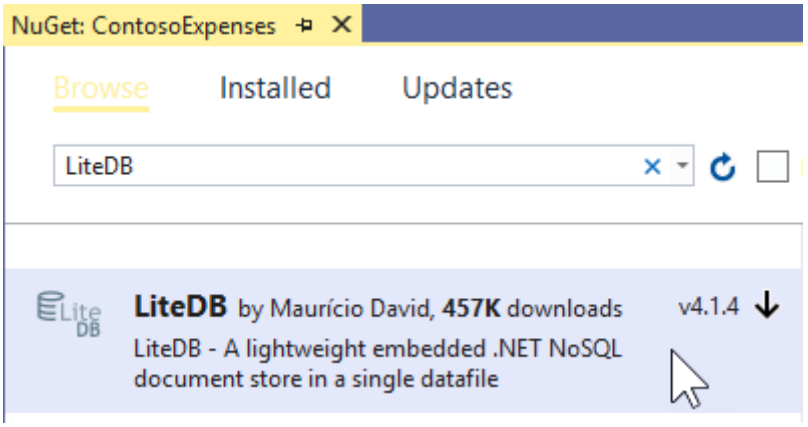
3. In the **ContosoExpenses.Data** project, right-click the **Dependencies** node and choose **Manage NuGet Packages**.



5. In the **NuGet Package Manager** window, click **Browse**. Search for the `Bogus` package and install the latest stable version.



6. Search for the `LiteDB` package and install the latest stable version.



You may be wondering where these list of NuGet packages is stored, since the project no longer has a packages.config file. The referenced NuGet packages are stored directly in the .csproj file. You can check this by viewing the contents of the **ContosoExpenses.Data.csproj** project file in a text editor. You will find the following lines added at the end of the file:

```
XML
```

```
<ItemGroup>
   <PackageReference Include="Bogus" Version="26.0.2" />
   <PackageReference Include="LiteDB" Version="4.1.4" />
</ItemGroup>
```

> **Note**
>
> You may also notice that you're installing the same packages for this .NET Core 3 project as the ones used by .NET Framework 4.7.2 projects. NuGet packages supports multi-targeting. Library authors can include different versions of a library in the same package, compiled for different architectures and platforms. These packages support the full .NET Framework as well as .NET Standard 2.0, which is compatible with .NET Core 3 projects. For more info about the differences .NET Framework, .NET Core and .NET Standard, see .NET Standard.

To configure NuGet packages for the **ContosoExpenses.Core** project:

1. In the **ContosoExpenses.Core** project, open the **packages.config** file. Notice that it currently contains the following references that target the .NET Framework 4.7.2.

   XML

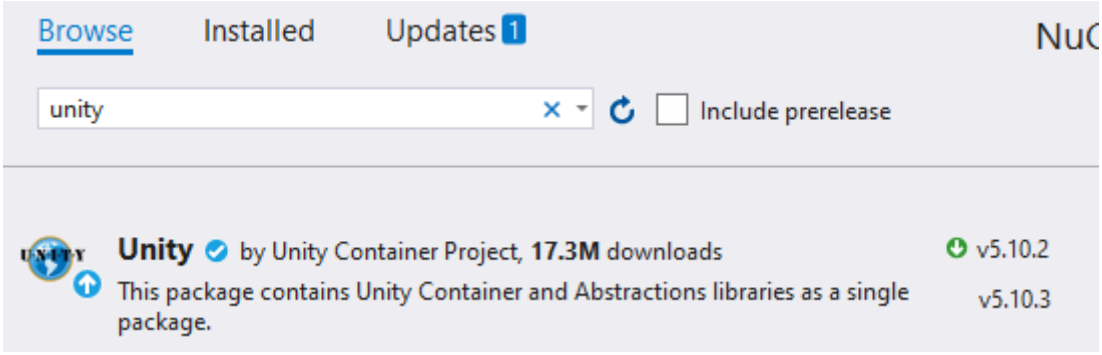
   ```
   <?xml version="1.0" encoding="utf-8"?>
   <packages>
     <package id="CommonServiceLocator" version="2.0.2"
   targetFramework="net472" />
     <package id="MvvmLightLibs" version="5.4.1.1"
   targetFramework="net472" />
     <package id="System.Runtime.CompilerServices.Unsafe" version="4.5.2"
   targetFramework="net472" />
     <package id="Unity" version="5.10.2" targetFramework="net472" />
   </packages>
   ```

   In the following steps you'll .NET Standard versions of the `MvvmLightLibs` and `Unity` packages. The other two are dependencies automatically downloaded by NuGet when you install these two libraries.

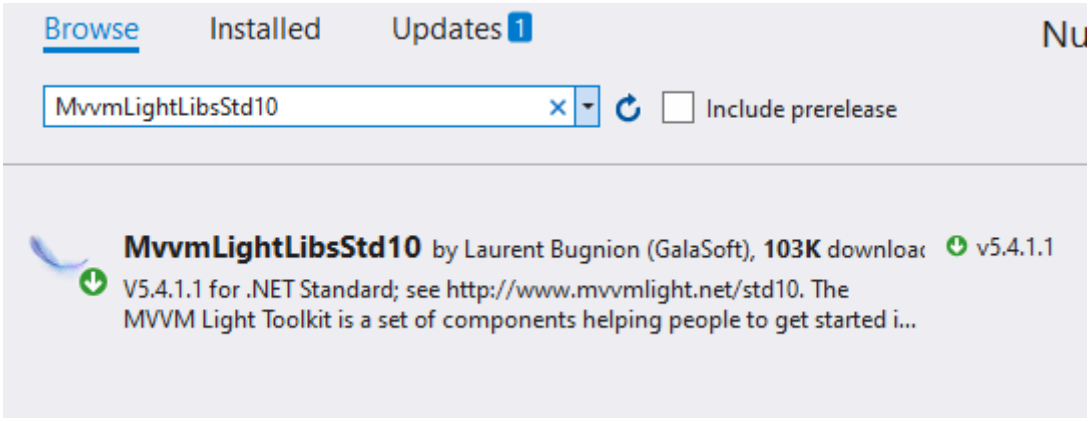
2. In the **ContosoExpenses.Core** project, delete the **Packages.config** file.

3. Right-click the **ContosoExpenses.Core** project and choose **Manage NuGet Packages**.

4. In the **NuGet Package Manager** window, click **Browse**. Search for the `Unity` package and install the latest stable version.

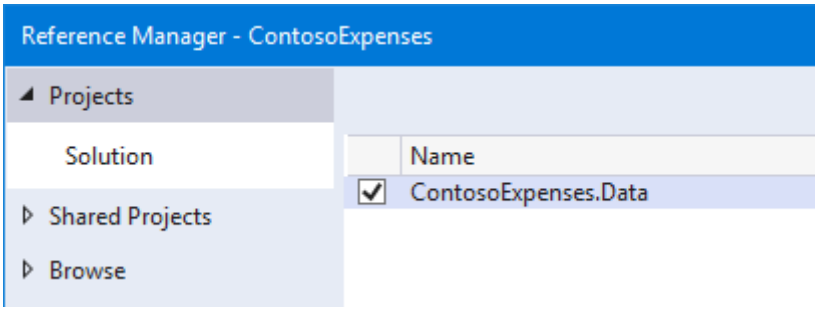


5. Search for the `MvvmLightLibsStd10` package and install the latest stable version. This is the .NET Standard version of the `MvvmLightLibs` package. For this package, the author chose to package the .NET Standard version of the library in a separate package than the .NET Framework version.



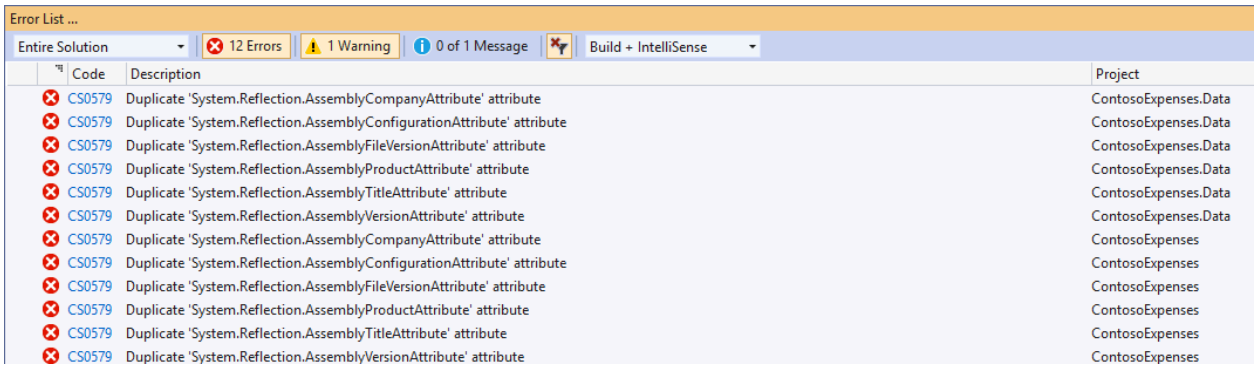
6. In the **ContosoExpenses.Core** project, right-click the **Dependencies** node and choose **Add Reference**.

7. In the **Projects** > **Solution** category, select **ContosoExpenses.Data** and click **OK**.



Disable auto-generated assembly attributes

At this point in the migration process, if you try to build the **ContosoExpenses.Core** project you'll see some errors.



This problem is happening because the new .csproj format introduced with .NET Core 3.0 stores the assembly info in the project file rather than the **AssemblyInfo.cs** file. To fix these errors, disable this behavior and let the project continue to use the **AssemblyInfo.cs** file.

1. In Visual Studio, right-click the **ContosoExpenses.Core** project and choose **Unload Project**. Right-click the project again and then choose **Edit ContosoExpenses.Core.csproj**.

2. Add the following element in the **PropertyGroup** section and save the file.

   XML

   ```
   <GenerateAssemblyInfo>false</GenerateAssemblyInfo>
   ```

   After adding this element, the **PropertyGroup** section should now look like this:

   XML

   ```
   <PropertyGroup>
     <OutputType>WinExe</OutputType>
     <TargetFramework>netcoreapp3.0</TargetFramework>
     <UseWPF>true</UseWPF>
     <GenerateAssemblyInfo>false</GenerateAssemblyInfo>
   </PropertyGroup>
   ```

3. Right-click the **ContosoExpenses.Core** project and choose **Reload Project**.

4. Right-click the **ContosoExpenses.Data** project and choose **Unload Project**. Right-click the project again and then choose **Edit ContosoExpenses.Data.csproj**.

5. Add the same entry in the **PropertyGroup** section and save the file.

   XML

   ```
   <GenerateAssemblyInfo>false</GenerateAssemblyInfo>
   ```

After adding this element, the **PropertyGroup** section should now look like this:

```
XML

<PropertyGroup>
  <TargetFramework>netstandard2.0</TargetFramework>
  <GenerateAssemblyInfo>false</GenerateAssemblyInfo>
</PropertyGroup>
```

6. Right-click the **ContosoExpenses.Data** project and choose **Reload Project**.

Add the Windows Compatibility Pack

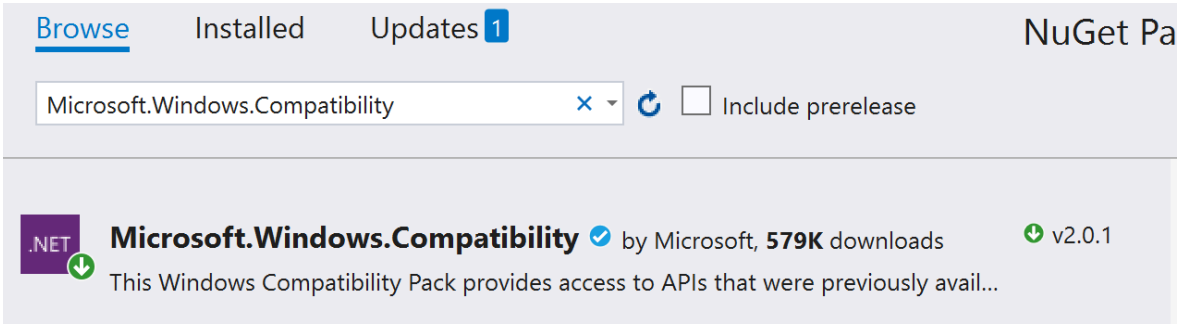
If you now try to compile the **ContosoExpenses.Core** and **ContosoExpenses.Data** projects, you'll see that the previous errors are now fixed but there are still some errors in the **ContosoExpenses.Data** library similar to these.

`Services\RegistryService.cs(9,26,9,34): error CS0103: The name 'Registry' does not exist in the current context` `Services\RegistryService.cs(12,26,12,34): error CS0103: The name 'Registry' does not exist in the current context` `Services\RegistryService.cs(12,97,12,123): error CS0103: The name 'RegistryKeyPermissionCheck' does not exist in the current context`

These errors are a result of converting the **ContosoExpenses.Data** project from a .NET Framework library (which is specific for Windows) to a .NET Standard library, which can run on multiple platforms including Linux, Android, iOS, and more. The **ContosoExpenses.Data** project contains a class called **RegistryService**, which interacts with the registry, a Windows-only concept.

To resolve these errors, install the Windows Compatibility NuGet package. This package provides support for many Windows-specific APIs to be used in a .NET Standard library. The library will no longer be cross-platform after using this package, but it will still target .NET Standard.

1. Right-click on the **ContosoExpenses.Data** project.
2. Choose **Manage NuGet Packages**.
3. In the **NuGet Package Manager** window, click **Browse**. Search for the `Microsoft.Windows.Compatibility` package and install the latest stable version.



4. Now try again to compile the project, by right clicking on the **ContosoExpenses.Data** project and choosing **Build**.

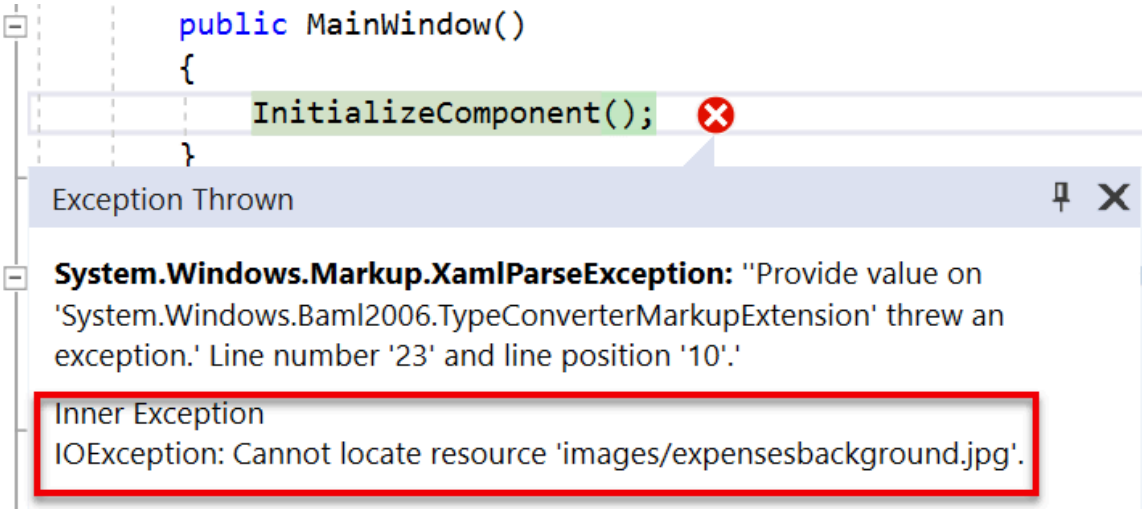
This time the build process will complete without errors.

Test and debug the migration

Now that the projects are building successfully, you are ready to run and test the app to see if there are any runtime errors.

1. Right-click the **ContosoExpenses.Core** project and choose **Set as Startup Project**.

2. Press F5 to start the **ContosoExpenses.Core** project in the debugger. You'll see an exception similar to the following.



   This exception is being raised because when you deleted the content from the .csproj file at the beginning of the migration, you removed the information about the **Build action** for the image files. The following steps fix this issue.

3. Stop the debugger.

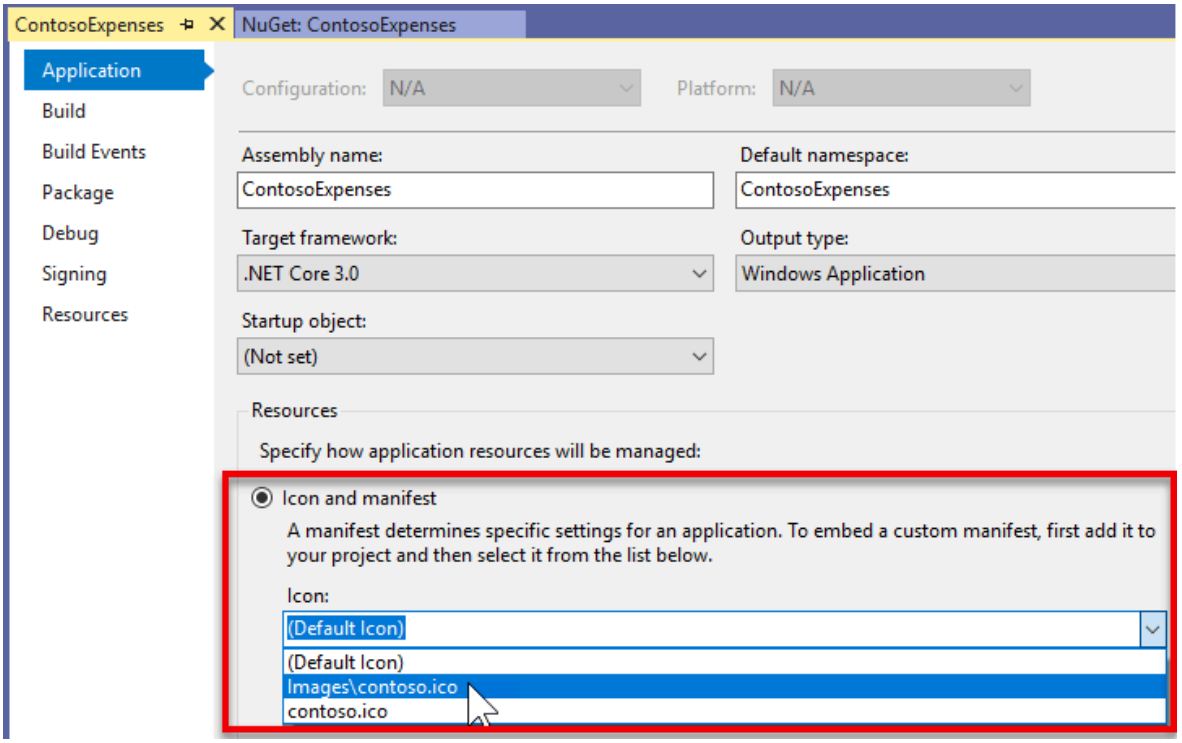
4. Right-click the **ContosoExpenses.Core** project and choose **Unload Project**. Right-click the project again and then choose **Edit ContosoExpenses.Core.csproj**.

5. Before the closing **Project** element, add the following entry:

```xml
<ItemGroup>
  <Content Include="Images/*">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </Content>
</ItemGroup>
```

6. Right-click the **ContosoExpenses.Core** project and choose **Reload Project**.

7. To assign the Contoso.ico to the app, right-click the **ContosoExpenses.Core** project and choose **Properties**. In the opened page, click drop-down under **Icon** and select `Images\contoso.ico`.



8. Click **Save**.

9. Press F5 to start the **ContosoExpenses.Core** project in the debugger. Confirm that the app now runs.

Next steps

At this point in the tutorial, you have successfully migrated the Contoso Expenses app to .NET Core 3. You are now ready for Part 2: Add a UWP InkCanvas control using XAML Islands.

> ⓘ Note

If you have a high resolution screen, you may notice that the app looks very small. You'll address this problem in the next step of the tutorial.

Part 2: Add a UWP InkCanvas control using XAML Islands

Article • 10/16/2021

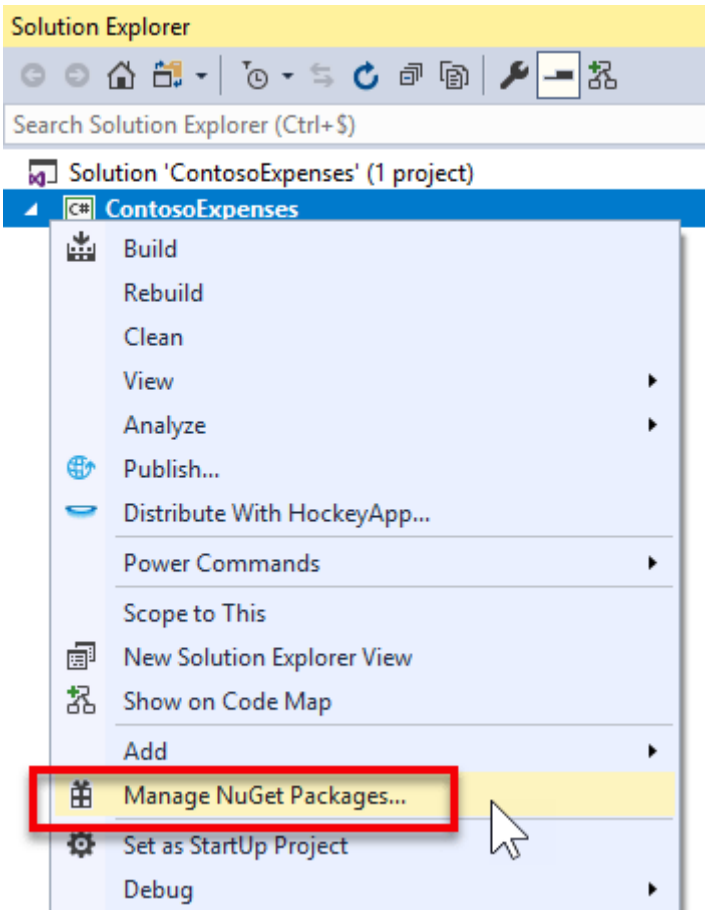
This is the second part of a tutorial that demonstrates how to modernize a sample WPF desktop app named Contoso Expenses. For an overview of the tutorial, prerequisites, and instructions for downloading the sample app, see Tutorial: Modernize a WPF app. This article assumes you have already completed part 1.

In the fictional scenario of this tutorial, the Contoso development team wants to add support for digital signatures to the Contoso Expenses app. The UWP **InkCanvas** control is a great option for this scenario, because it supports digital ink and AI-powered features like the capability to recognize text and shapes. To do this, you will use the InkCanvas wrapped UWP control available in the Windows Community Toolkit. This control wraps the interface and functionality of the UWP **InkCanvas** control for use in a WPF app. For more details about wrapped UWP controls, see Host UWP XAML controls in desktop apps (XAML Islands).

Configure the project to use XAML Islands

Before you can add an **InkCanvas** control to the Contoso Expenses app, you first need to configure the project to support UWP XAML Islands.

1. In Visual Studio 2019, right-click on the **ContosoExpenses.Core** project in **Solution Explorer** and choose **Manage NuGet Packages**.



2. In the **NuGet Package Manager** window, click **Browse**. Search for the `Microsoft.Toolkit.Wpf.UI.Controls` package and install version 6.0.0 or a later version.

> ⓘ Note
>
> This package contains all the necessary infrastructure for hosting UWP XAML Islands in a WPF app, including the **InkCanvas** wrapped UWP control. A similar package named `Microsoft.Toolkit.Forms.UI.Controls` is available for Windows Forms apps.

3. Right-click **ContosoExpenses.Core** project in **Solution Explorer** and choose **Add -> New item**.

4. Select **Application Manifest File**, name it **app.manifest**, and click **Add**. For more information about application manifests, see this article.

5. In the manifest file, uncomment the following `<supportedOS>` element for Windows 10.

```
XML
```

```
<!-- Windows 10 -->
<supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a15a9a}" />
```

6. In the manifest file, locate the following commented `<application>` element.

```
XML

<!--
<application xmlns="urn:schemas-microsoft-com:asm.v3">
  <windowsSettings>
    <dpiAware
xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">true</dpi
Aware>
  </windowsSettings>
</application>
-->
```

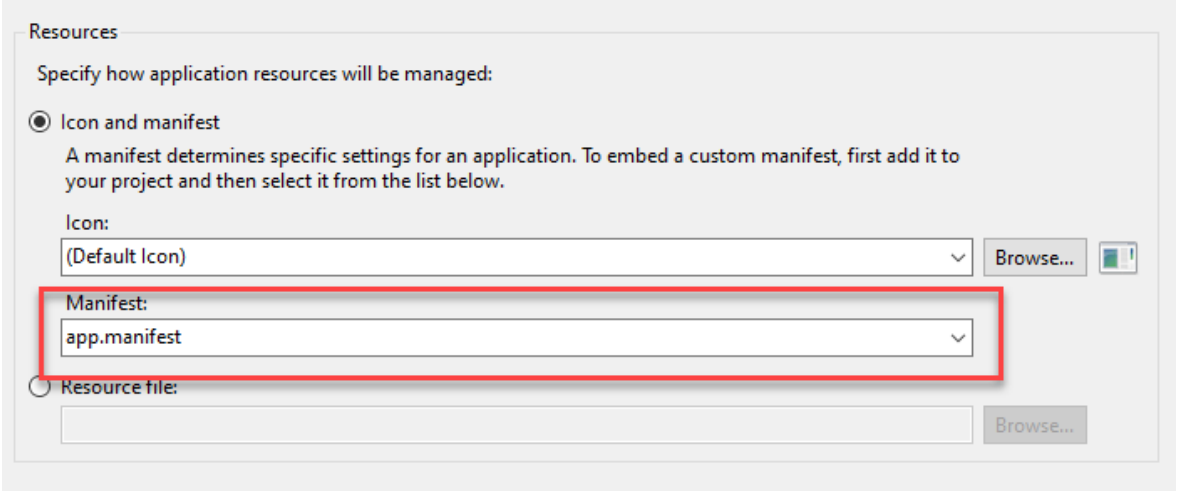
7. Delete this section and replace it with the following XML. This configures the app to be DPI aware and better handle different scaling factors supported by Windows 10.

```
XML

<application xmlns="urn:schemas-microsoft-com:asm.v3">
  <windowsSettings>
      <dpiAware
xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">true/PM</
dpiAware>
      <dpiAwareness
xmlns="http://schemas.microsoft.com/SMI/2016/WindowsSettings">PerMonito
rV2, PerMonitor</dpiAwareness>
  </windowsSettings>
</application>
```

8. Save and close the `app.manifest` file.

9. In **Solution Explorer**, right-click the **ContosoExpenses.Core** project and choose **Properties**.

10. In the **Resources** section of the **Application** tab, make sure the **Manifest** dropdown is set to **app.manifest**.



11. Save the changes to the project properties.

Add an InkCanvas control to the app

Now that you have configured your project to use UWP XAML Islands, you are now ready to add an InkCanvas wrapped UWP control to the app.

1. In **Solution Explorer**, expand the **Views** folder of the **ContosoExpenses.Core** project and double-click the **ExpenseDetail.xaml** file.

2. In the **Window** element near the top of the XAML file, add the following attribute. This references the XAML namespace for the InkCanvas wrapped UWP control.

```
XML

xmlns:toolkit="clr-
namespace:Microsoft.Toolkit.Wpf.UI.Controls;assembly=Microsoft.Toolkit.
Wpf.UI.Controls"
```

After adding this attribute, the **Window** element should now look like this.

```
XML

<Window x:Class="ContosoExpenses.Views.ExpenseDetail"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:toolkit="clr-
namespace:Microsoft.Toolkit.Wpf.UI.Controls;assembly=Microsoft.Toolkit.
Wpf.UI.Controls"
        xmlns:converters="clr-namespace:ContosoExpenses.Converters"
        DataContext="{Binding Source={StaticResource ViewModelLocator},
```

```
Path=ExpensesDetailViewModel}"
        xmlns:local="clr-namespace:ContosoExpenses"
        mc:Ignorable="d"
        Title="Expense Detail" Height="500" Width="800"
        Background="{StaticResource HorizontalBackground}">
```

3. In the **ExpenseDetail.xaml** file, locate the closing `</Grid>` tag that immediately precedes the `<!-- Chart -->` comment. Add the following XAML just before the closing `</Grid>` tag. This XAML adds an **InkCanvas** control (prefixed by the **toolkit** keyword you defined earlier as a namespace) and a simple **TextBlock** that acts as a header for the control.

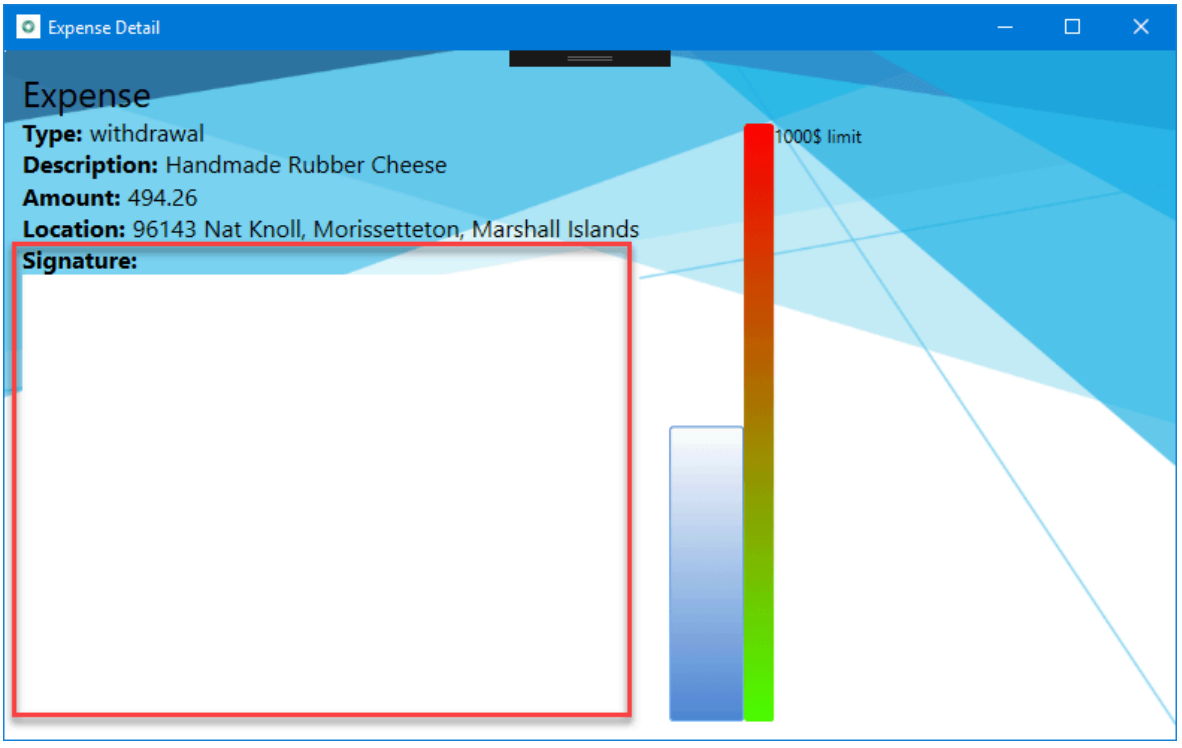
XML

```
<TextBlock Text="Signature:" FontSize="16" FontWeight="Bold"
Grid.Row="5" />

<toolkit:InkCanvas x:Name="Signature" Grid.Row="6" />
```

4. Save the **ExpenseDetail.xaml** file.

5. Press F5 to run the app in the debugger.

6. Choose an employee from the list and then choose one of the available expenses. Notice that the expense detail page contains space for the **InkCanvas** control.



If you have a device which supports a digital pen, like a Surface, and you're running this lab on a physical machine, go on and try to use it. You will see the

digital ink appearing on the screen. However, if you don't have a pen capable device and you try to sign with your mouse, nothing will happen. This is happening because the **InkCanvas** control is enabled only for digital pens by default. However, we can change this behavior.

7. Close the app and double-click the **ExpenseDetail.xaml.cs** file under the **Views** folder of the **ContosoExpenses.Core** project.

8. Add the following namespace declaration at the top of the class:

```
C#

using Microsoft.Toolkit.Win32.UI.Controls.Interop.WinRT;
```

9. Locate the `ExpenseDetail()` constructor.

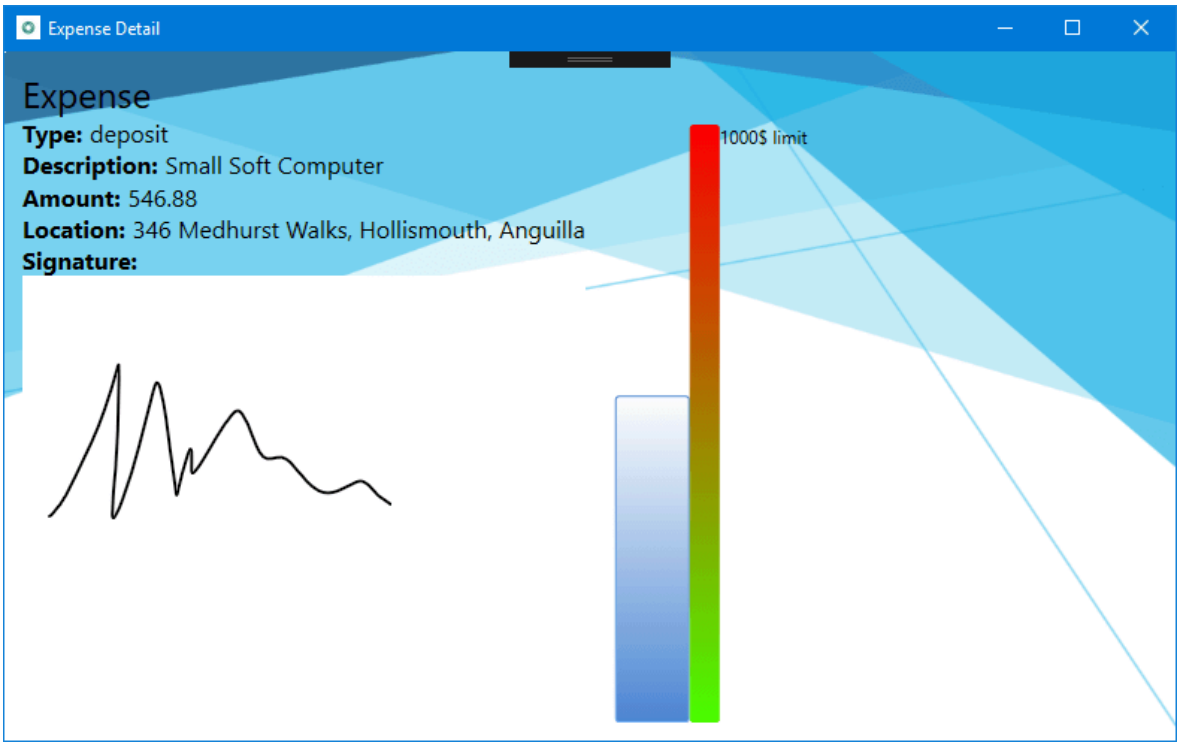
10. Add the following line of code after the `InitializeComponent()` method and save the code file.

```
C#

Signature.InkPresenter.InputDeviceTypes = CoreInputDeviceTypes.Mouse |
CoreInputDeviceTypes.Pen;
```

    You can use the **InkPresenter** object to customize the default inking experience. This code uses the **InputDeviceTypes** property to enable mouse as well as pen input.

11. Press F5 again to rebuild and run the app in the debugger. Choose an employee from the list and then choose one of the available expenses.

12. Try now to draw something in the signature space with the mouse. This time, you'll see the ink appearing on the screen.



Next steps

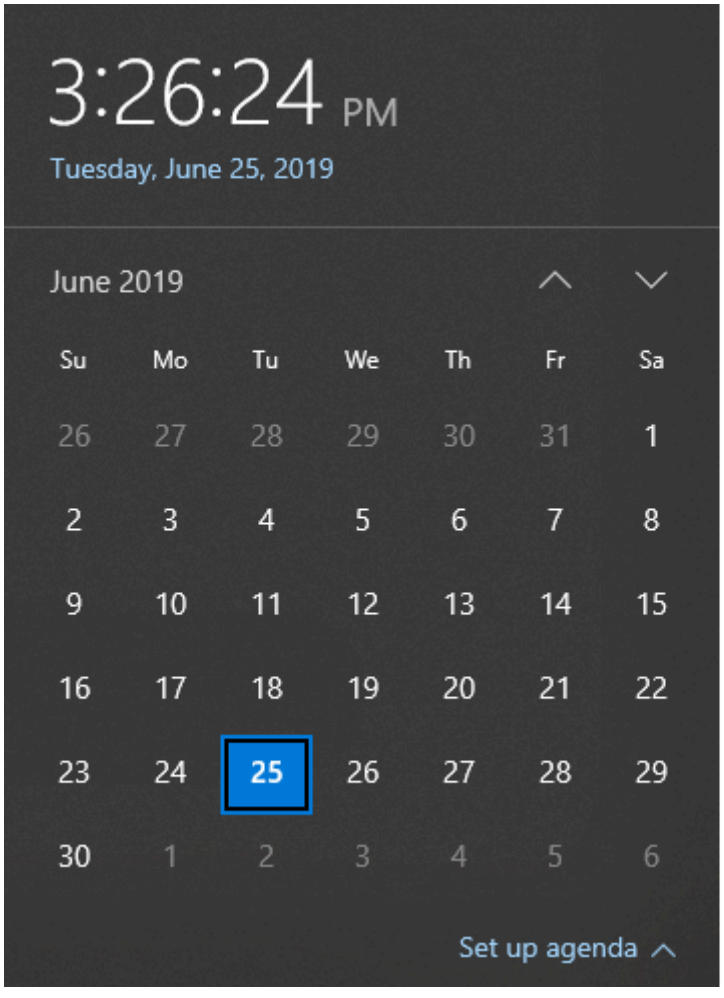
At this point in the tutorial, you have successfully added a UWP **InkCanvas** control to the Contoso Expenses app. You are now ready for Part 3: Add a UWP CalendarView control using XAML Islands.

Part 3: Add a UWP CalendarView control using XAML Islands

Article • 05/13/2022

This is the third part of a tutorial that demonstrates how to modernize a sample WPF desktop app named Contoso Expenses. For an overview of the tutorial, prerequisites, and instructions for downloading the sample app, see Tutorial: Modernize a WPF app. This article assumes you have already completed part 2.

In the fictional scenario of this tutorial, the Contoso development team wants to make it easier to choose the date for an expense report on a touch-enabled device. In this part of the tutorial, you will add a UWP CalendarView control to the app. This is the same control that is used in the Windows date and time functionality on the taskbar.



Unlike the **InkCanvas** control you added in part 2, the Windows Community Toolkit does not provide a wrapped version of the UWP **CalendarView** that can be used in WPF apps. As an alternative, you'll host an **InkCanvas** in the generic WindowsXamlHost control. You can use this control to host any first-party UWP control provided by the Windows SDK or WinUI library or any custom UWP control created by a third party. The

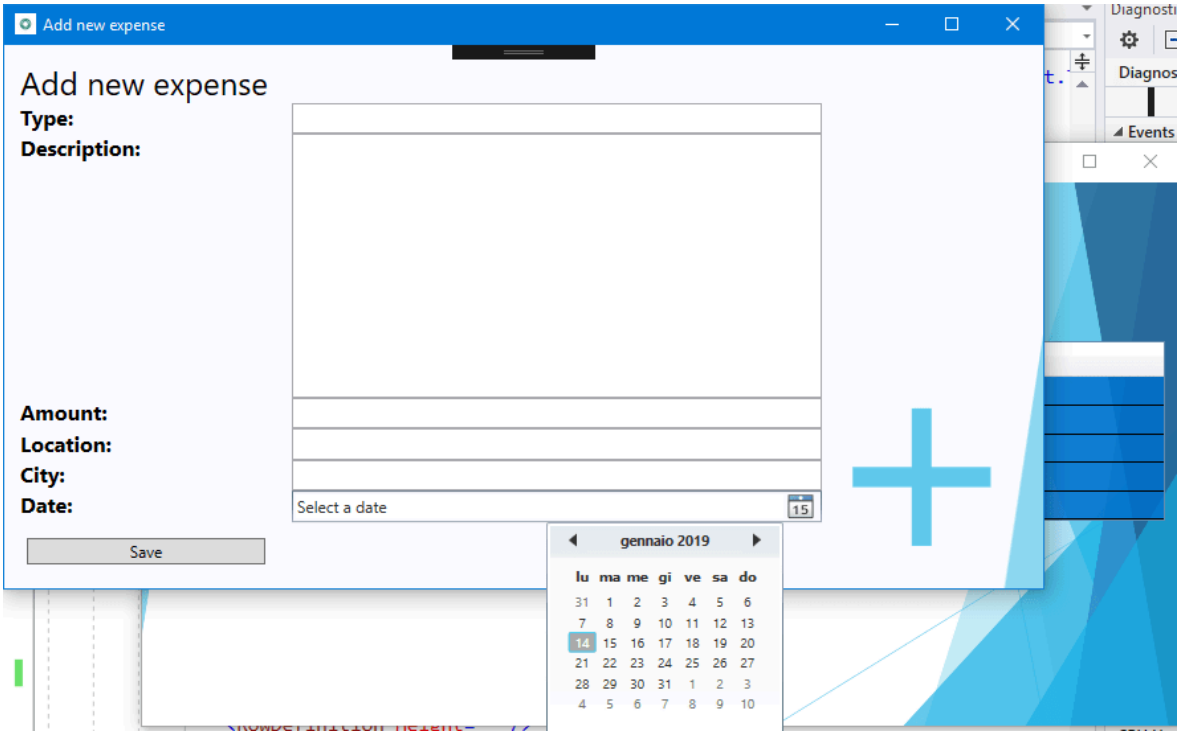
WindowsXamlHost control is provided by the `Microsoft.Toolkit.Wpf.UI.XamlHost` package NuGet package. This package is included with the `Microsoft.Toolkit.Wpf.UI.Controls` NuGet package that you installed in part 2.

> **Note**
>
> This tutorial only demonstrates how to use WindowsXamlHost to host the first-party **CalendarView** control provided by the Windows SDK. For a walkthrough that demonstrates how to host a custom control, see Host a custom UWP control in a WPF app using XAML Islands.

In order to use the **WindowsXamlHost** control, you'll need to directly call WinRT APIs from code in the WPF app. The `Microsoft.Windows.SDK.Contracts` NuGet package contains the references necessary to enable you to call WinRT APIs from the app. This package is also included in the `Microsoft.Toolkit.Wpf.UI.Controls` NuGet package that you installed in part 2.

Add the WindowsXamlHost control

1. In **Solution Explorer**, expand the **Views** folder in the **ContosoExpenses.Core** project and double-click the **AddNewExpense.xaml** file. This is the form used to add a new expense to the list. Here is how it appears in the current version of the app.



The date picker control included in WPF is meant for traditional computers with mouse and keyboard. Choosing a date with a touch screen isn't really feasible, due to the small size of the control and the limited space between each day in the calendar.

2. At the top of the **AddNewExpense.xaml** file, add the following attribute to the **Window** element.

```
XML

xmlns:xamlhost="clr-
namespace:Microsoft.Toolkit.Wpf.UI.XamlHost;assembly=Microsoft.Toolkit.
Wpf.UI.XamlHost"
```

After adding this attribute, the **Window** element should now look like this.

```
XML

<Window x:Class="ContosoExpenses.Views.AddNewExpense"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:xamlhost="clr-
namespace:Microsoft.Toolkit.Wpf.UI.XamlHost;assembly=Microsoft.Toolkit.
Wpf.UI.XamlHost"
        DataContext="{Binding Source={StaticResource
ViewModelLocator},Path=AddNewExpenseViewModel}"
        xmlns:local="clr-namespace:ContosoExpenses"
        mc:Ignorable="d"
        Title="Add new expense" Height="450" Width="800"
        Background="{StaticResource AddNewExpenseBackground}">
```

3. Change the **Height** attribute of the **Window** element from 450 to 800. This is needed because the UWP **CalendarView** control takes more space than the WPF date picker.

```
XML

<Window x:Class="ContosoExpenses.Views.AddNewExpense"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:xamlhost="clr-
```

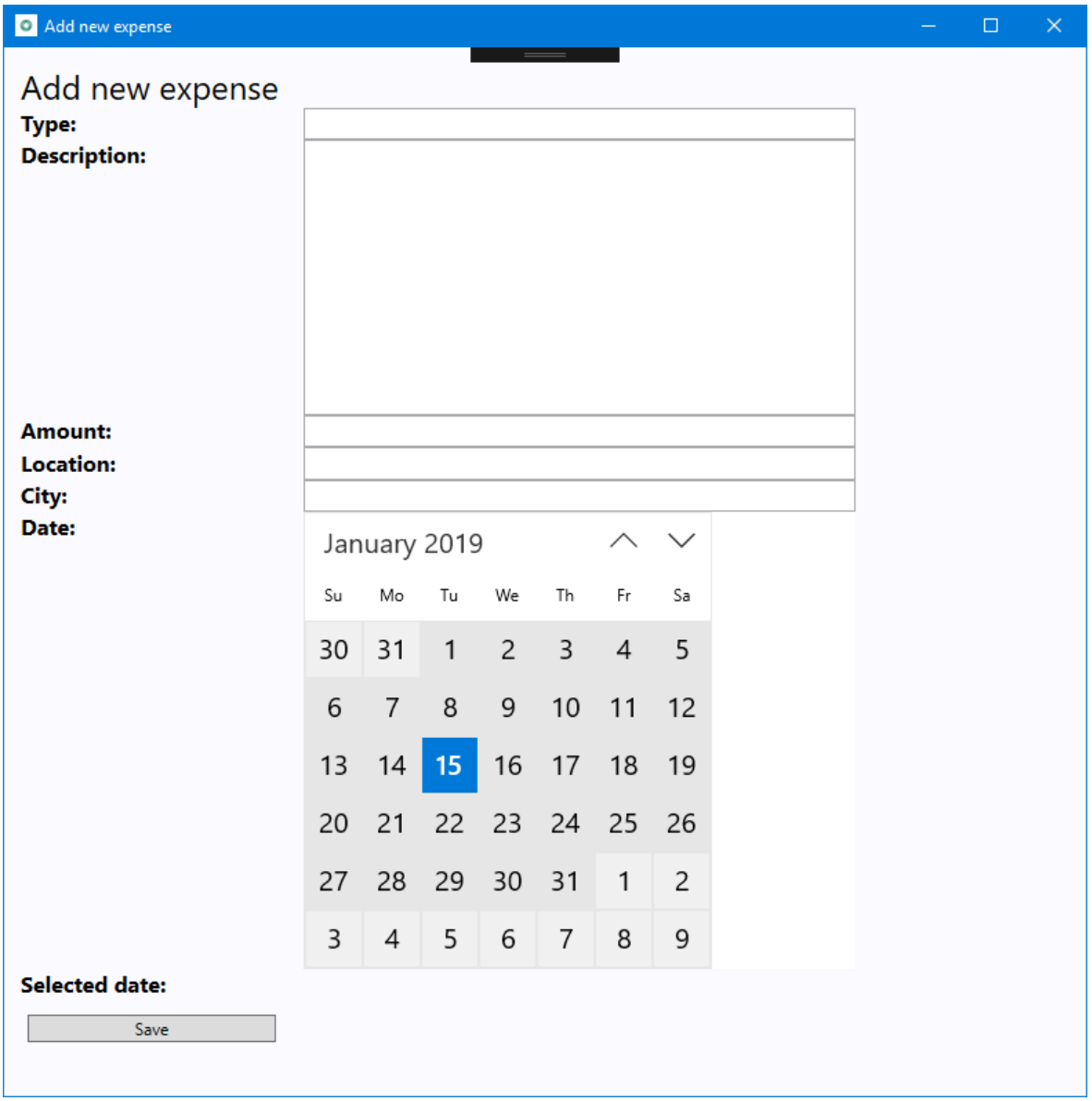
```
namespace:Microsoft.Toolkit.Wpf.UI.XamlHost;assembly=Microsoft.Toolkit.
Wpf.UI.XamlHost"
        DataContext="{Binding Source={StaticResource
ViewModelLocator},Path=AddNewExpenseViewModel}"
        xmlns:local="clr-namespace:ContosoExpenses"
        mc:Ignorable="d"
        Title="Add new expense" Height="800" Width="800"
        Background="{StaticResource AddNewExpenseBackground}">
```

4. Locate the `DatePicker` element near the bottom of the file, and replace this element with the following XAML.

```
XML

<xamlhost:WindowsXamlHost
InitialTypeName="Windows.UI.Xaml.Controls.CalendarView" Grid.Column="1"
Grid.Row="6" Margin="5, 0, 0, 0" x:Name="CalendarUwp"  />
```

   This XAML adds the **WindowsXamlHost** control. The **InitialTypeName** property indicates the full name of the UWP control you want to host (in this case, **Windows.UI.Xaml.Controls.CalendarView**).

5. Press F5 to build and run the app in the debugger. Choose an employee from the list and then press the **Add new expense** button. Confirm that the following page hosts the new UWP **CalendarView** control.



6. Close the app.

Interact with the WindowsXamlHost control

Next, you'll update the app to process the selected date, display it on the screen, and populate the **Expense** object to save in the database.

The UWP CalendarView contains two members that are relevant for this scenario:

- The **SelectedDates** property contains the date selected by the user.
- The **SelectedDatesChanged** event is raised when the user selects a date.

However, the **WindowsXamlHost** control is a generic host control for *any* kind of UWP control. As such, it doesn't expose a property called **SelectedDates** or an event called **SelectedDatesChanged**, because they are specific of the **CalendarView** control. To access these members, you must write code that casts the **WindowsXamlHost** to the **CalendarView** type. The best place to do this is in response to the **ChildChanged** event

of the **WindowsXamlHost** control, which is raised when the hosted control has been rendered.

1. In the **AddNewExpense.xaml** file, add an event handler for the **ChildChanged** event of the **WindowsXamlHost** control you added earlier. When you are done, the **WindowsXamlHost** element should look like this.

```
XML
```

```
<xamlhost:WindowsXamlHost
InitialTypeName="Windows.UI.Xaml.Controls.CalendarView" Grid.Column="1"
Grid.Row="6" Margin="5, 0, 0, 0" x:Name="CalendarUwp"
ChildChanged="CalendarUwp_ChildChanged" />
```

2. In the same file, locate the **Grid.RowDefinitions** element of the main **Grid**. Add one more **RowDefinition** element with the **Height** equal to **Auto** at the end of the list of child elements. When you are done, the **Grid.RowDefinitions** element should look like this (there should now be 9 **RowDefinition** elements).

```
XML
```

```
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
```

3. Add the following XAML after the **WindowsXamlHost** element and before the **Button** element near the end of the file.

```
XML
```

```
<TextBlock Text="Selected date:" FontSize="16" FontWeight="Bold"
Grid.Row="7" Grid.Column="0" />
<TextBlock Text="{Binding Path=Date}" FontSize="16" Grid.Row="7"
Grid.Column="1" />
```

4. Locate the **Button** element near the end of the file and change the **Grid.Row** property from **7** to **8**. This shifts the button down one row in the grid because you added a new row.

XML

```
<Button Content="Save" Grid.Row="8" Grid.Column="0" Command="{Binding
Path=SaveExpenseCommand}" Margin="5, 12, 0, 0"
HorizontalAlignment="Left" Width="180" />
```

5. Open the **AddNewExpense.xaml.cs** code file.

6. Add the following statement to the top of the file.

C#

```
using Microsoft.Toolkit.Wpf.UI.XamlHost;
```

7. Add the following event handler to the `AddNewExpense` class. This code implements the **ChildChanged** event of the **WindowsXamlHost** control you declared earlier in the XAML file.

C#

```
private void CalendarUwp_ChildChanged(object sender, System.EventArgs
e)
{
    WindowsXamlHost windowsXamlHost = (WindowsXamlHost)sender;

    Windows.UI.Xaml.Controls.CalendarView calendarView =
        (Windows.UI.Xaml.Controls.CalendarView)windowsXamlHost.Child;

    if (calendarView != null)
    {
        calendarView.SelectedDatesChanged += (obj, args) =>
        {
            if (args.AddedDates.Count > 0)
            {
                Messenger.Default.Send<SelectedDateMessage>(new
SelectedDateMessage(args.AddedDates[0].DateTime));
            }
        };
    }
}
```

This code uses the **Child** property of the **WindowsXamlHost** control to access the UWP **CalendarView** control. The code then subscribes to the **SelectedDatesChanged** event, which is triggered when the user selects a date from the calendar. This event handler passes the selected date to the ViewModel where the new **Expense** object is created and saved to the database. To do this, the code uses the messaging infrastructure provided by the MVVM Light NuGet package.

The code sends a message called **SelectedDateMessage** to the ViewModel, which will receive it and set the **Date** property with the selected value. In this scenario the **CalendarView** control is configured for single selection mode, so the collection will contain only one element.

At this point, the project still won't compile because it is missing the implementation for the **SelectedDateMessage** class. The following steps implement this class.

8. In **Solution Explorer**, right-click the **Messages** folder and choose **Add -> Class**. Name the new class **SelectedDateMessage** and click **Add**.

9. Replace the contents of the **SelectedDateMessage.cs** code file with the following code. This code adds a using statement for the `GalaSoft.MvvmLight.Messaging` namespace (from the MVVM Light NuGet package), inherits from the **MessageBase** class, and adds **DateTime** property that is initialized through the public constructor. When you are done, the file should look like this.

```
C#
```

```
using GalaSoft.MvvmLight.Messaging;
using System;

namespace ContosoExpenses.Messages
{
    public class SelectedDateMessage: MessageBase
    {
        public DateTime SelectedDate { get; set; }

        public SelectedDateMessage(DateTime selectedDate)
        {
            this.SelectedDate = selectedDate;
        }
    }
}
```

10. Next, you'll update the ViewModel to receive this message and populate the **Date** property of the ViewModel. In **Solution Explorer**, expand the **ViewModels** folder and open the **AddNewExpenseViewModel.cs** file.

11. Locate the public constructor for the `AddNewExpenseViewModel` class and add the following code to the end of the constructor. This code registers to receive the **SelectedDateMessage**, extract the selected date from it through the **SelectedDate** property, and we use it to set the **Date** property exposed by the ViewModel. Because this property is bound with the **TextBlock** control you added earlier, you can see the date selected by the user.

```
C#

Messenger.Default.Register<SelectedDateMessage>(this, message =>
{
    Date = message.SelectedDate;
});
```

When you are done, the `AddNewExpenseViewModel` constructor should look like this.

```
C#

public AddNewExpenseViewModel(IDatabaseService databaseService,
IStorageService storageService)
{
    this.databaseService = databaseService;
    this.storageService = storageService;

    Date = DateTime.Today;

    Messenger.Default.Register<SelectedDateMessage>(this, message =>
    {
        Date = message.SelectedDate;
    });
}
```

> **Note**
>
> The `SaveExpenseCommand` method in the same code file does the work of saving the expenses to the database. This method uses the **Date** property to create the new **Expense** object. This property is now being populated by the **CalendarView** control through the `SelectedDateMessage` message you just created.

12. Press F5 to build and run the app in the debugger. Choose one of the available employees and then click the **Add new expense** button. Complete all fields in the form and choose a date from the new **CalendarView** control. Confirm the selected date is displayed as a string below the control.

13. Press the **Save** button. The form will be closed and the new expense is added to the end of the list of expenses. The first column with the expense date should be the date you selected in the **CalendarView** control.

Next steps

At this point in the tutorial, you have successfully replaced a WPF date time control with the UWP **CalendarView** control, which supports touch and digital pens in addition to mouse and keyboard input. Although the Windows Community Toolkit doesn't provide a wrapped version of the UWP **CalendarView** control that can be used directly in a WPF app, you were able to host the control by using the generic WindowsXamlHost control.

You are now ready for Part 4: Add Windows user activities and notifications.

Part 4: Add Windows user activities and notifications

Article • 05/13/2022

This is the fourth part of a tutorial that demonstrates how to modernize a sample WPF desktop app named Contoso Expenses. For an overview of the tutorial, prerequisites, and instructions for downloading the sample app, see Tutorial: Modernize a WPF app. This article assumes you have already completed part 3.

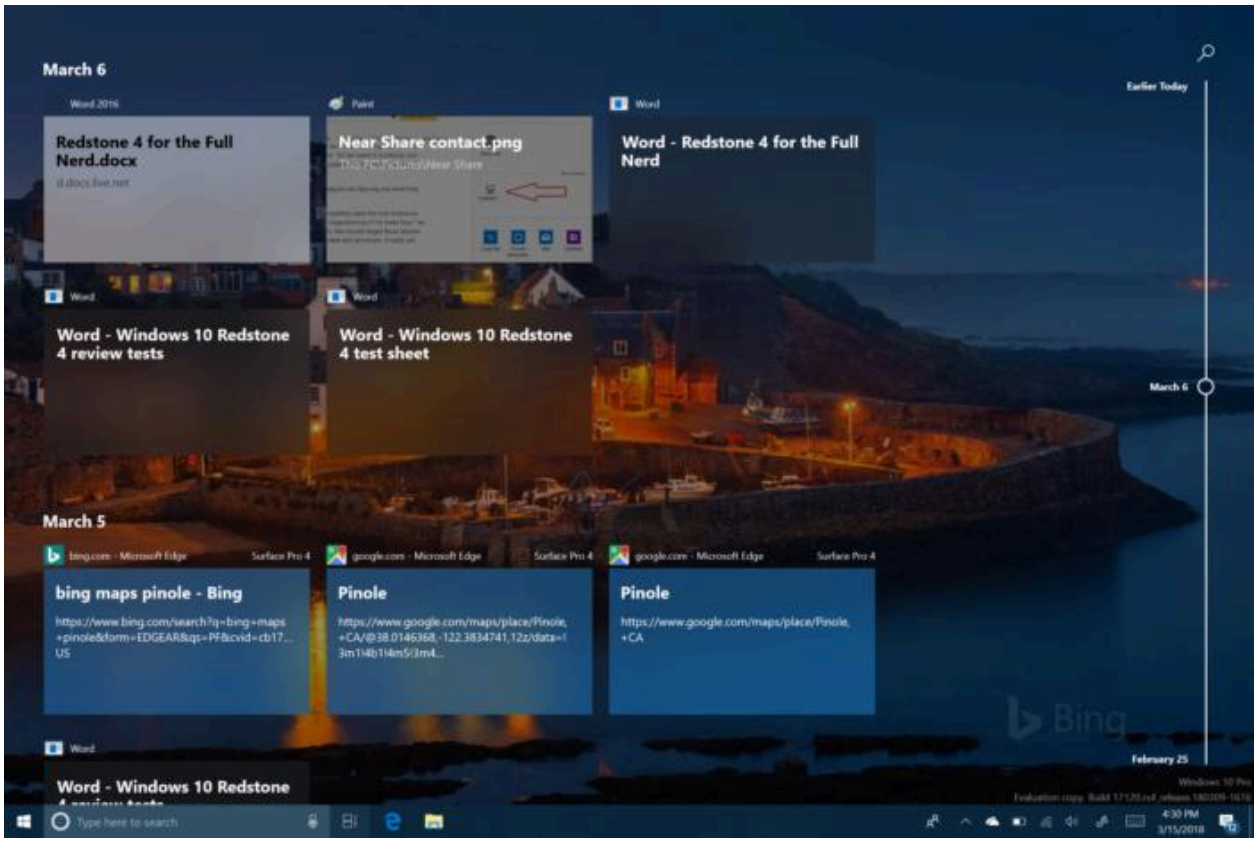
In the previous parts of this tutorial you added UWP XAML controls to the app using XAML Islands. As a byproduct of this, you also enabled the app to call any WinRT API. This opens up the opportunity for the app to use many other features offered by Windows, not just UWP XAML controls.

In the fictional scenario of this tutorial, the Contoso development team has decided to add two new features to the app: activities and notifications. This part of the tutorial shows how to implement these features.

Add a user activity

> **Note**
>
> The timeline feature is discontinued since Windows 11

In Windows 10, apps can track activities performed by the user such as opening a file or displaying a specific page. These activities are then made available through Timeline, a feature introduced in Windows 10 version 1803, which allows the user to quickly go back to the past and resume an activity they started previously.



User activities are tracked using Microsoft Graph. However, when you're building a Windows 10 app, you don't need to interact directly with the REST endpoints provided by Microsoft Graph. Instead, you can use a convenient set of WinRT APIs. We're going to use these WinRT APIs in the Contoso Expenses app to track every time the user opens an expense within the app, and use Adaptive Cards to enable users to create the activity.

Introduction to Adaptive Cards

This section provides a brief overview of Adaptive Cards. If you don't need this information, you can skip this and go right to the add an Adaptive Card instructions.

Adaptive Cards enable developers to exchange card content in a common and consistent way. An Adaptive Card is described by a JSON payload that defines its content, which can include text, images, actions, and more.

An Adaptive Card defines just the content and not the visual appearance of the content. The platform where the Adaptive Card is received can render the content using the most appropriate styling. The way Adaptive Cards are designed is through a renderer, which is able to take the JSON payload and to convert it into native UI. For example, the UI could be XAML for a WPF or UWP app, AXML for an Android app, or HTML for a website or a bot chat.

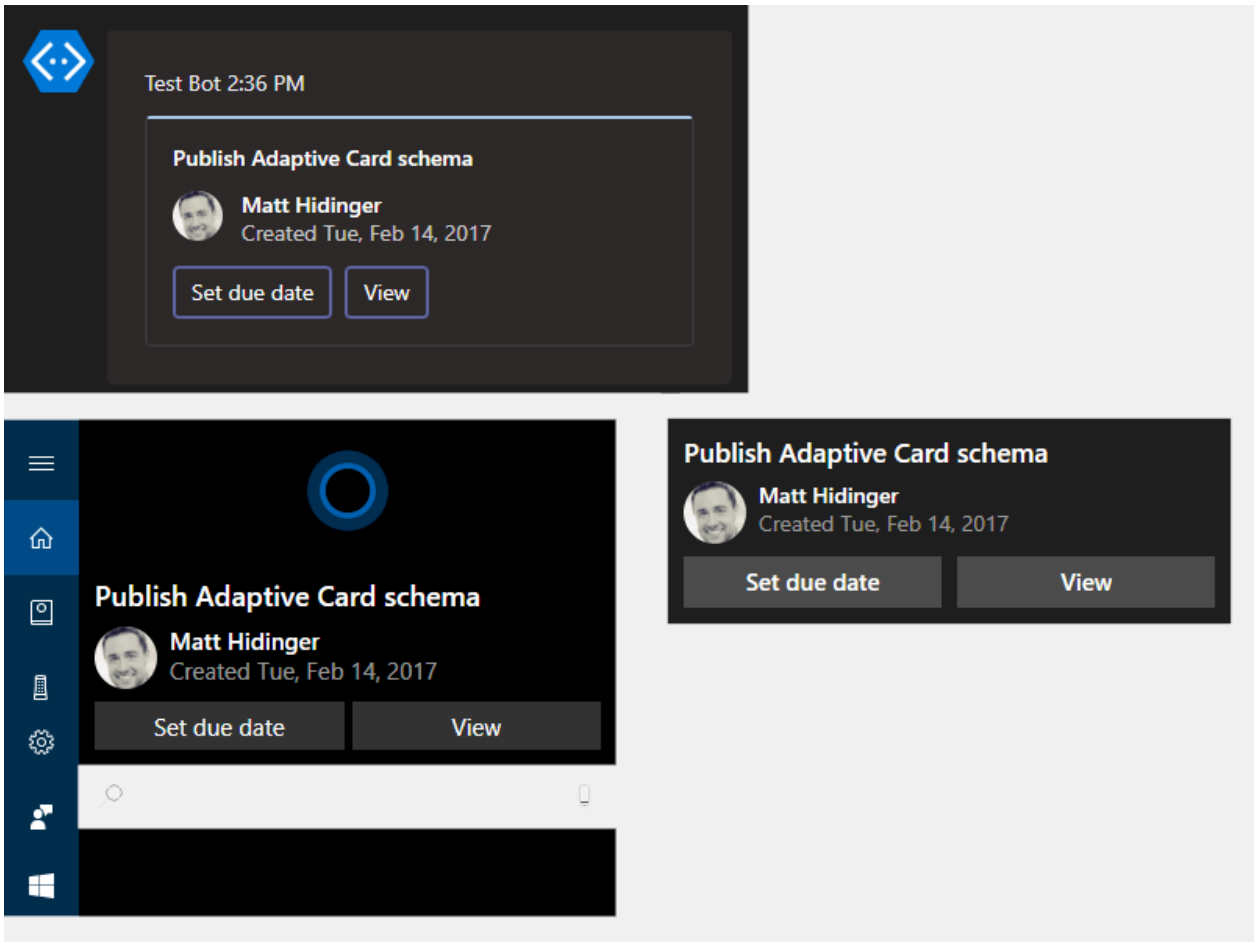
Here is an example of a simple Adaptive Card payload.

```
JSON
```

```
{
    "type": "AdaptiveCard",
    "body": [
        {
            "type": "Container",
            "items": [
                {
                    "type": "TextBlock",
                    "size": "Medium",
                    "weight": "Bolder",
                    "text": "Publish Adaptive Card schema"
                },
                {
                    "type": "ColumnSet",
                    "columns": [
                        {
                            "type": "Column",
                            "items": [
                                {
                                    "type": "Image",
                                    "style": "Person",
                                    "url":
"https://pbs.twimg.com/profile_images/3647943215/d7f12830b3c17a5a9e4afcc370e
3a37e_400x400.jpeg",
                                    "size": "Small"
                                }
                            ],
                            "width": "auto"
                        },
                        {
                            "type": "Column",
                            "items": [
                                {
                                    "type": "TextBlock",
                                    "weight": "Bolder",
                                    "text": "Matt Hidinger",
                                    "wrap": true
                                },
                                {
                                    "type": "TextBlock",
                                    "spacing": "None",
                                    "text": "Created {{DATE(2017-02-
14T06:08:39Z,SHORT)}}",
                                    "isSubtle": true,
                                    "wrap": true
                                }
                            ],
                            "width": "stretch"
                        }
                    ]
                }
            ]
        }
    ],
```

```
    "actions": [
        {
            "type": "Action.ShowCard",
            "title": "Set due date",
            "card": {
                "type": "AdaptiveCard",
                "style": "emphasis",
                "body": [
                    {
                        "type": "Input.Date",
                        "id": "dueDate"
                    },
                    {
                        "type": "Input.Text",
                        "id": "comment",
                        "placeholder": "Add a comment",
                        "isMultiline": true
                    }
                ],
                "actions": [
                    {
                        "type": "Action.OpenUrl",
                        "title": "OK",
                        "url": "http://adaptivecards.io"
                    }
                ],
                "$schema": "http://adaptivecards.io/schemas/adaptive-
card.json"
            }
        },
        {
            "type": "Action.OpenUrl",
            "title": "View",
            "url": "http://adaptivecards.io"
        }
    ],
    "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
    "version": "1.0"
}
```

The image below shows how this JSON is rendered in different ways by ta Teams channel, Cortana and a Windows notification.



Adaptive cards play an important role in Timeline because it's the way Windows renders activities. Each thumbnail displayed inside Timeline is actually an Adaptive Card. As such, when you're going to create a user activity inside your app, you will be asked to provide an Adaptive Card to render it.

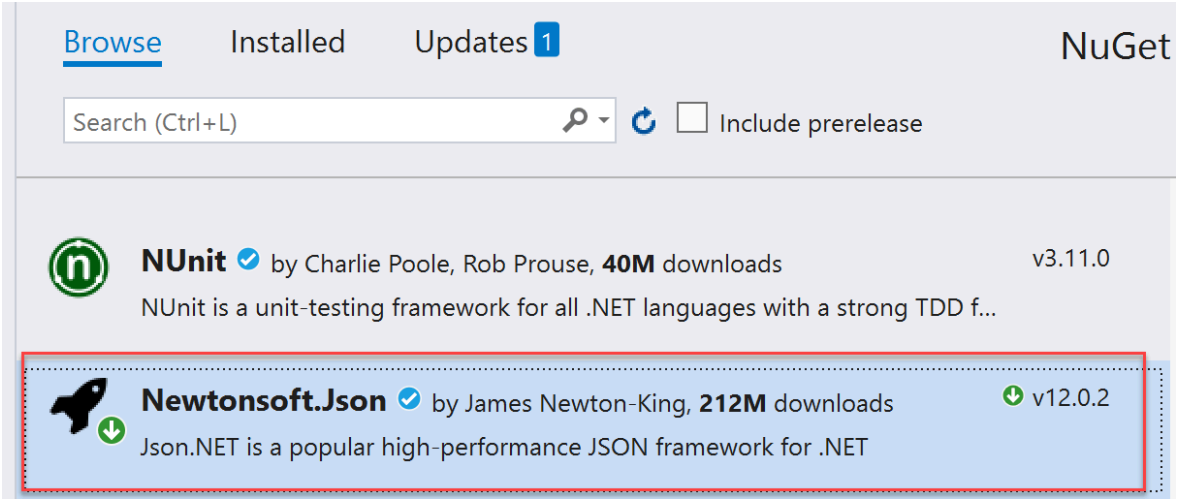
> ⓘ Note
>
> A great way to brainstorm the design of an Adaptive Card is using the online designer. You will have the chance to design the card with building blocks (images, texts, columns, etc) and get the corresponding JSON. After you have an idea of the final design, you can use a library called Adaptive Cards to make it easier to build your Adaptive Card using C# classes instead of plain JSON, which might be hard to debug and build.

Add an Adaptive Card

1. Right click on the **ContosoExpenses.Core** project in Solution Explorer and choose **Manage NuGet packages**.

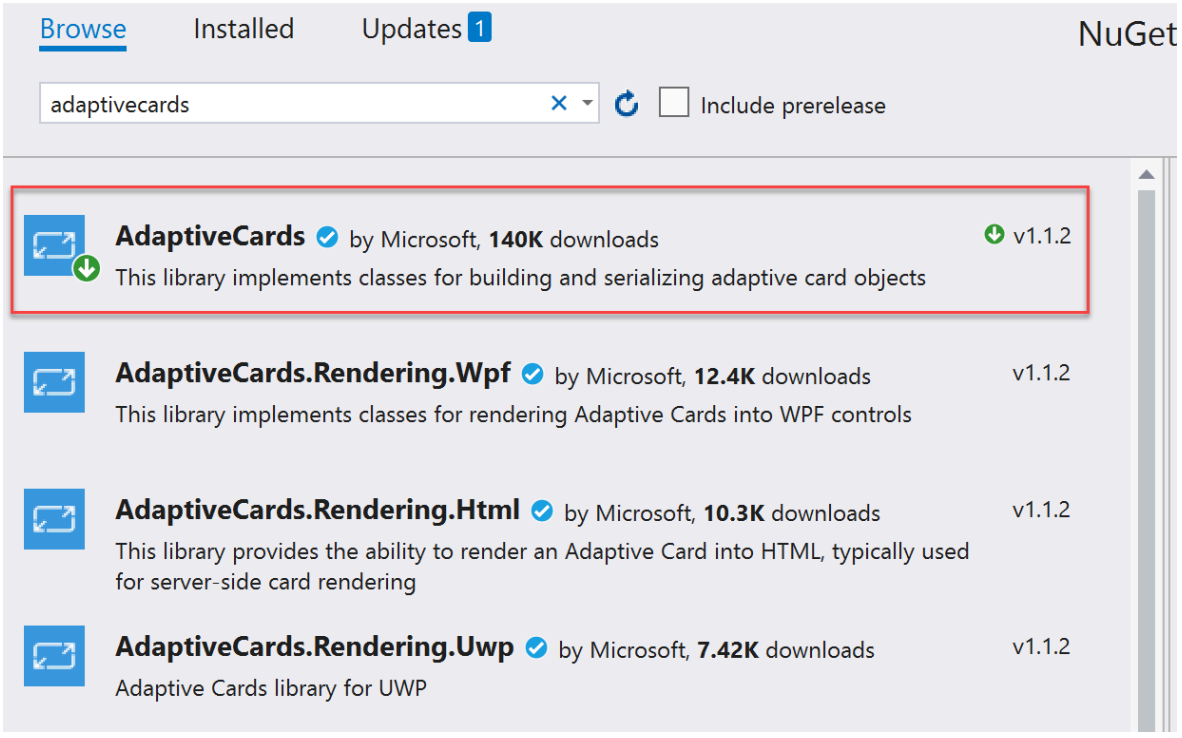
2. In the **NuGet Package Manager** window, click **Browse**. Search for the `Newtonsoft.Json` package and install the latest available version. This is a popular

JSON manipulation library that you will use to help manipulate the JSON strings required by Adaptive Cards.



> **Note**
>
> If you don't install the `Newtonsoft.Json` package separately, The Adaptive Cards library will reference an older version of the `Newtonsoft.Json` package that doesn't support .NET Core 3.0.

3. In the **NuGet Package Manager** window, click **Browse**. Search for the `AdaptiveCards` package and install the latest available version.



4. In **Solution Explorer**, right-click the **ContosoExpenses.Core** project, choose **Add -> Class**. Name the class **TimelineService.cs** and click **OK**.

5. In the **TimelineService.cs** file, add the following statements to the top of the file.

```
C#

using AdaptiveCards;
using ContosoExpenses.Data.Models;
```

6. Change the namespace declared in the file from `ContosoExpenses.Core` to `ContosoExpenses`.

7. Add the following method to the `TimelineService` class.

```
C#

 private string BuildAdaptiveCard(Expense expense)
 {
     AdaptiveCard card = new AdaptiveCard("1.0");

     AdaptiveTextBlock title = new AdaptiveTextBlock
     {
         Text = expense.Description,
         Size = AdaptiveTextSize.Medium,
         Wrap = true
     };

     AdaptiveColumnSet columnSet = new AdaptiveColumnSet();
     AdaptiveColumn photoColumn = new AdaptiveColumn
     {
         Width = "auto"
     };

     AdaptiveImage image = new AdaptiveImage
     {
         Url = new
Uri("https://appmodernizationworkshop.blob.core.windows.net/contosoexpe
nses/Contoso192x192.png"),
         Size = AdaptiveImageSize.Small,
         Style = AdaptiveImageStyle.Default
     };
     photoColumn.Items.Add(image);

     AdaptiveTextBlock amount = new AdaptiveTextBlock
     {
         Text = expense.Cost.ToString(),
         Weight = AdaptiveTextWeight.Bolder,
         Wrap = true
     };

     AdaptiveTextBlock date = new AdaptiveTextBlock
     {
         Text = expense.Date.Date.ToShortDateString(),
         IsSubtle = true,
```

```
            Spacing = AdaptiveSpacing.None,
            Wrap = true
        };

        AdaptiveColumn expenseColumn = new AdaptiveColumn
        {
            Width = "stretch"
        };
        expenseColumn.Items.Add(amount);
        expenseColumn.Items.Add(date);

        columnSet.Columns.Add(photoColumn);
        columnSet.Columns.Add(expenseColumn);

        card.Body.Add(title);
        card.Body.Add(columnSet);

        string json = card.ToJson();
        return json;
    }
```

About the code

This method receives an **Expense** object with all the information about the expense to render and it builds a new **AdaptiveCard** object. The method adds the following to the card:

- A title, which uses the description of the expense.
- An image, which is the Contoso logo.
- The amount of the expense.
- The date of the expense.

The last 3 elements are split into two different columns, so that the Contoso logo and the details about the expense can be placed side by side. After the object is built, the method returns the corresponding JSON string with the help of the **ToJson** method.

Define the user activity

Now that you have defined the Adaptive Card, you can create a user activity based on it.

1. Add the following statements to the top of **TimelineService.cs** file:

```
C#
```

```
using Windows.ApplicationModel.UserActivities;
using System.Threading.Tasks;
```

```
using Windows.UI.Shell;
```

> **Note**
>
> These are UWP namespaces. These resolve because the `Microsoft.Toolkit.Wpf.UI.Controls` NuGet package that you installed in step 2 includes a reference to the `Microsoft.Windows.SDK.Contracts` package, which enables the **ContosoExpenses.Core** project to reference WinRT APIs even though it is a .NET Core 3 project.

2. Add the following field declarations to the `TimelineService` class.

C#

```
private UserActivityChannel _userActivityChannel;
private UserActivity _userActivity;
private UserActivitySession _userActivitySession;
```

3. Add the following method to the `TimelineService` class.

C#

```
public async Task AddToTimeline(Expense expense)
{
    _userActivityChannel = UserActivityChannel.GetDefault();
    _userActivity = await
_userActivityChannel.GetOrCreateUserActivityAsync($"Expense-
{expense.ExpenseId}");

    _userActivity.ActivationUri = new
Uri($"contosoexpenses://expense/{expense.ExpenseId}");
    _userActivity.VisualElements.DisplayText = "Contoso Expenses";

    string json = BuildAdaptiveCard(expense);

    _userActivity.VisualElements.Content =
AdaptiveCardBuilder.CreateAdaptiveCardFromJson(json);

    await _userActivity.SaveAsync();
    _userActivitySession?.Dispose();
    _userActivitySession = _userActivity.CreateSession();
}
```

4. Save the changes to **TimelineService.cs**.

About the code

The `AddToTimeline` method first gets a **UserActivityChannel** object that is required to store user activities. Then it creates a new user activity using the **GetOrCreateUserActivityAsync** method, which requires a unique identifier. This way, if an activity already exists, the app can update it; otherwise it will create a new one. The identifier to pass depends by the kind of application you're building:

- If you want to always update the same activity so that Timeline will only show the most recent one, you can use a fixed identifier (such as **Expenses**).
- If you want to track every activity as a different one, so that Timeline will display all of them, you can use a dynamic identifier.

In this scenario, the app will track each opened expense as a different user activity, so the code creates each identifier by using the keyword **Expense-** followed by the unique expense ID.

After the method creates a **UserActivity** object, it populates the object with the following info:

- An **ActivationUri** that is invoked when the user clicks on the activity in Timeline. The code uses a custom protocol called **contosoexpenses** that the app will handle later.
- The **VisualElements** object, which contains a set of properties that define the visual appearance of the activity. This code sets the **DisplayText** (which is the title displayed on top of the entry in Timeline) and the **Content**.

This is where the Adaptive Card you defined earlier plays a role. The app passes the Adaptive Card you designed earlier as content to the method. However, Windows 10 uses a different object to represent a card compared to the one used by the `AdaptiveCards` NuGet package. Therefore, the method recreates the card by using the **CreateAdaptiveCardFromJson** method exposed by the **AdaptiveCardBuilder** class. After the method creates the user activity, it saves the activity and creates a new session.

When a user clicks on an activity in Timeline, the **contosoexpenses://** protocol will be activated and the URL will include the information the app needs to retrieve the selected expense. As an optional task, you could implement the protocol activation so that the application reacts properly when the user uses Timeline.

Integrate the application with Timeline

Now that you have created a class that interacts with Timeline, we can start using it to enhance the application's experience. The best place to use the **AddToTimeline** method

exposed by the **TimelineService** class is when the user opens the detail page of an expense.

1. In the **ContosoExpenses.Core** project, expand the **ViewModels** folder and open the **ExpenseDetailViewModel.cs** file. This is the ViewModel that supports the expense detail's window.

2. Locate the public constructor of the **ExpenseDetailViewModel** class and add the following code at the end of the constructor. Whenever the expense window is opened, the method calls the **AddToTimeline** method and passes the current expense. The **TimelineService** class uses this info to create a user activity using the expense information.

   C#

   ```
   TimelineService timeline = new TimelineService();
   timeline.AddToTimeline(expense);
   ```

   When you are done, the constructor should look like this.

   C#

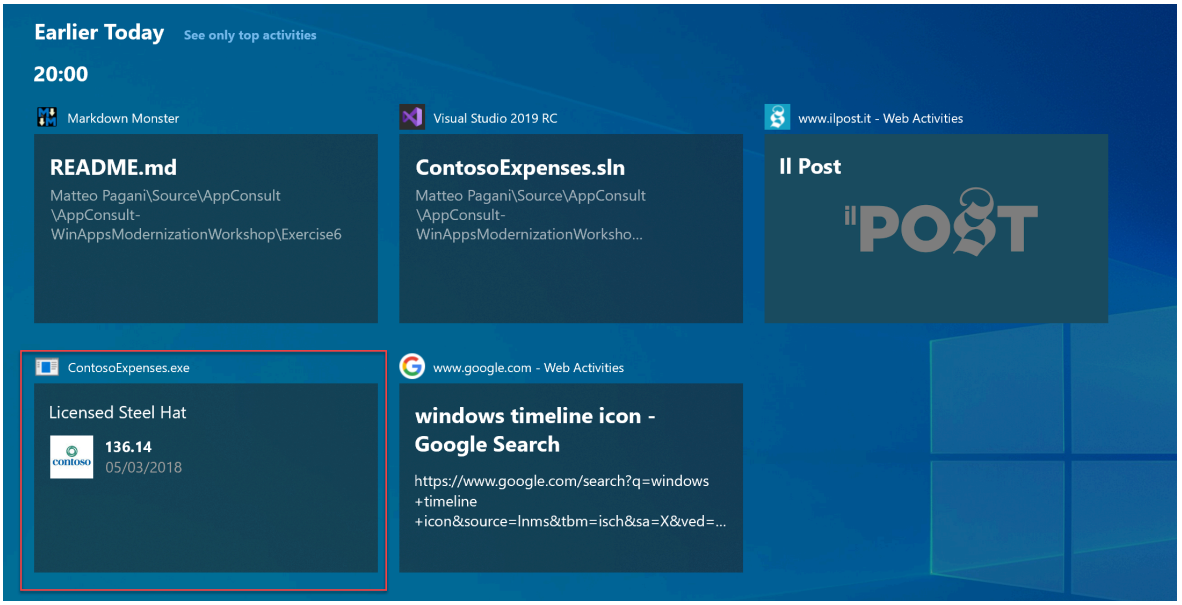
   ```
   public ExpensesDetailViewModel(IDatabaseService databaseService,
   IStorageService storageService)
   {
       var expense =
   databaseService.GetExpense(storageService.SelectedExpense);

       ExpenseType = expense.Type;
       Description = expense.Description;
       Location = expense.Address;
       Amount = expense.Cost;

       TimelineService timeline = new TimelineService();
       timeline.AddToTimeline(expense);
   }
   ```

3. Press F5 to build and run the app in the debugger. Choose an employee from the list and then choose an expense. In the detail page, note the description of the expense, the date and the amount.

4. Press **Start + TAB** to open Timeline.

5. Scroll down the list of currently opened applications until you see the section titled **Earlier today**. This section shows some of your most recent user activities. Click the **See all activities** link next to the **Earlier today** heading.

6. Confirm that you see a new card with the information about the expense you have just selected in the application.



7. If you now open other expenses, you will see new cards being added as user activities. Remember that the code uses a different identifier for each activity, so it creates a card for each expense you open in the app.

8. Close the app.

Add a notification

The second feature the Contoso development team wants to add is a notification that is shown to the user whenever a new expense is saved to the database. To do this, you can leverage the built-in notifications system in Windows 10, which is exposed to developers via WinRT APIs. This notification system has many advantages:

- Notifications are consistent with the rest of the OS.
- They are actionable.
- They get stored in Action Center so they can be reviewed later.

To add a notification to the app:

1. In **Solution Explorer**, right-click the **ContosoExpenses.Core** project, choose **Add -> Class**. Name the class **NotificationService.cs** and click **OK**.

2. In the **NotificationService.cs** file, add the following statements to the top of the file.

```
C#
```

```
using Windows.Data.Xml.Dom;
using Windows.UI.Notifications;
```

3. Change the namespace declared in the file from `ContosoExpenses.Core` to `ContosoExpenses`.

4. Add the following method to the `NotificationService` class.

```
C#
```

```
public void ShowNotification(string description, double amount)
{
    string xml = $@"<toast>
                      <visual>
                        <binding template='ToastGeneric'>
                          <text>Expense added</text>
                          <text>Description: {description} - Amount:
{amount} </text>
                        </binding>
                      </visual>
                    </toast>";

    XmlDocument doc = new XmlDocument();
    doc.LoadXml(xml);

    ToastNotification toast = new ToastNotification(doc);
    ToastNotificationManager.CreateToastNotifier().Show(toast);
}
```

Toast notifications are represented by an XML payload, which can include text, images, actions, and more. You can find all the supported elements here. This code uses a very simple schema with two lines of text: the title and the body. After the code defines the XML payload and loads it in an **XmlDocument** object, it wraps the XML in a **ToastNotification** object and shows it by using the **ToastNotificationManager** class.

5. In the **ContosoExpenses.Core** project, expand the **ViewModels** folder and open the **AddNewExpenseViewModel.cs** file.

6. Locate the `SaveExpenseCommand` method, which is triggered when the user presses on the button to save a new expense. Add the following code to this method, just after the call to the `SaveExpense` method.

```
C#
```

```
NotificationService notificationService = new NotificationService();
notificationService.ShowNotification(expense.Description,
```

```
expense.Cost);
```

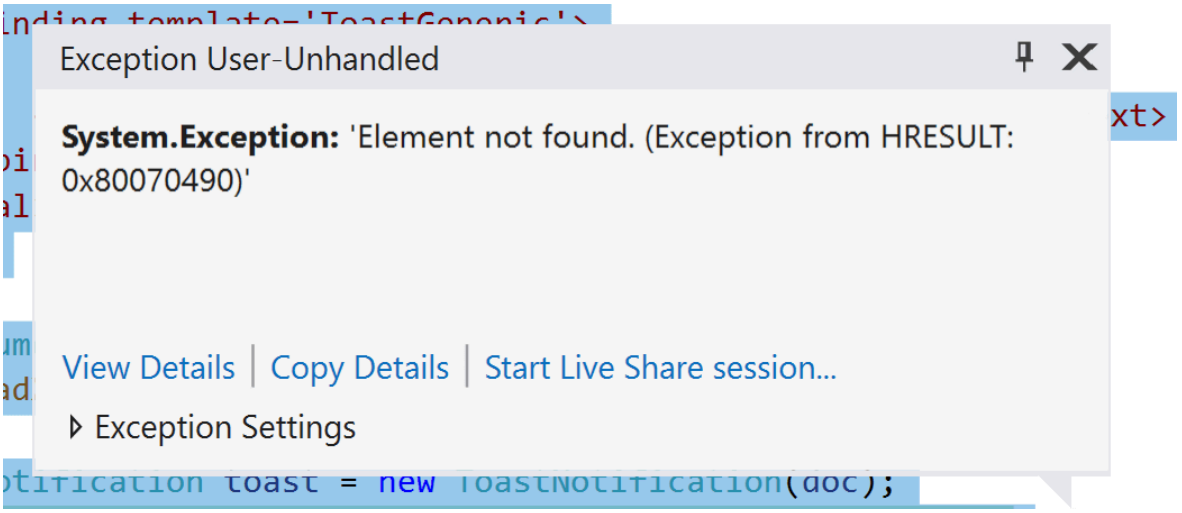
When you are done, the `SaveExpenseCommand` method should look like this.

C#

```
private RelayCommand _saveExpenseCommand;
public RelayCommand SaveExpenseCommand
{
    get
    {
        if (_saveExpenseCommand == null)
        {
            _saveExpenseCommand = new RelayCommand(() =>
            {
                Expense expense = new Expense
                {
                    Address = Address,
                    City = City,
                    Cost = Cost,
                    Date = Date,
                    Description = Description,
                    EmployeeId = storageService.SelectedEmployeeId,
                    Type = ExpenseType
                };

                databaseService.SaveExpense(expense);

                NotificationService notificationService = new
NotificationService();

notificationService.ShowNotification(expense.Description,
expense.Cost);

                Messenger.Default.Send<UpdateExpensesListMessage>(new
UpdateExpensesListMessage());
                Messenger.Default.Send<CloseWindowMessage>(new
CloseWindowMessage());
            }, () => IsFormFilled
            );
        }

        return _saveExpenseCommand;
    }
}
```

7. Press F5 to build and run the app in the debugger. Choose an employee from the list and then click the **Add new expense** button. Complete all fields in the form and press **Save**.

8. You will receive the following exception.



This exception is caused by the fact that the Contoso Expenses app doesn't yet have package identity. Some WinRT APIs, including the notifications API, require package identity before they can be used in an app. UWP apps receive package identity by default because they can only be distributed via MSIX packages. Other types of Windows apps, including WPF apps, can also be deployed via MSIX packages to obtain package identity. The next part of this tutorial will explore how to do this.

Next steps

At this point in the tutorial, you have successfully added a user activity to the app that integrates with Windows Timeline, and you have also added a notification to the app that is triggered when users create a new expense. However, the notification doesn't yet work because the app requires package identity to use the notifications API. To learn how to build an MSIX package for the app to obtain package identity and gain other deployment benefits, see Part 5: Package and deploy with MSIX.

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

Open a documentation issue

Provide product feedback

Part 5: Package and deploy with MSIX

Article • 02/09/2022

This is the final part of a tutorial that demonstrates how to modernize a sample WPF desktop app named Contoso Expenses. For an overview of the tutorial, prerequisites, and instructions for downloading the sample app, see Tutorial: Modernize a WPF app. This article assumes you have already completed part 4.

In part 4 you learned that some WinRT APIs, including the notifications API, require package identity before they can be used in an app. You can obtain package identity by packaging Contoso Expenses using MSIX, the packaging format introduced in Windows 10 to package and deploy Windows applications. MSIX provides advantages for developers and IT Pros, including:

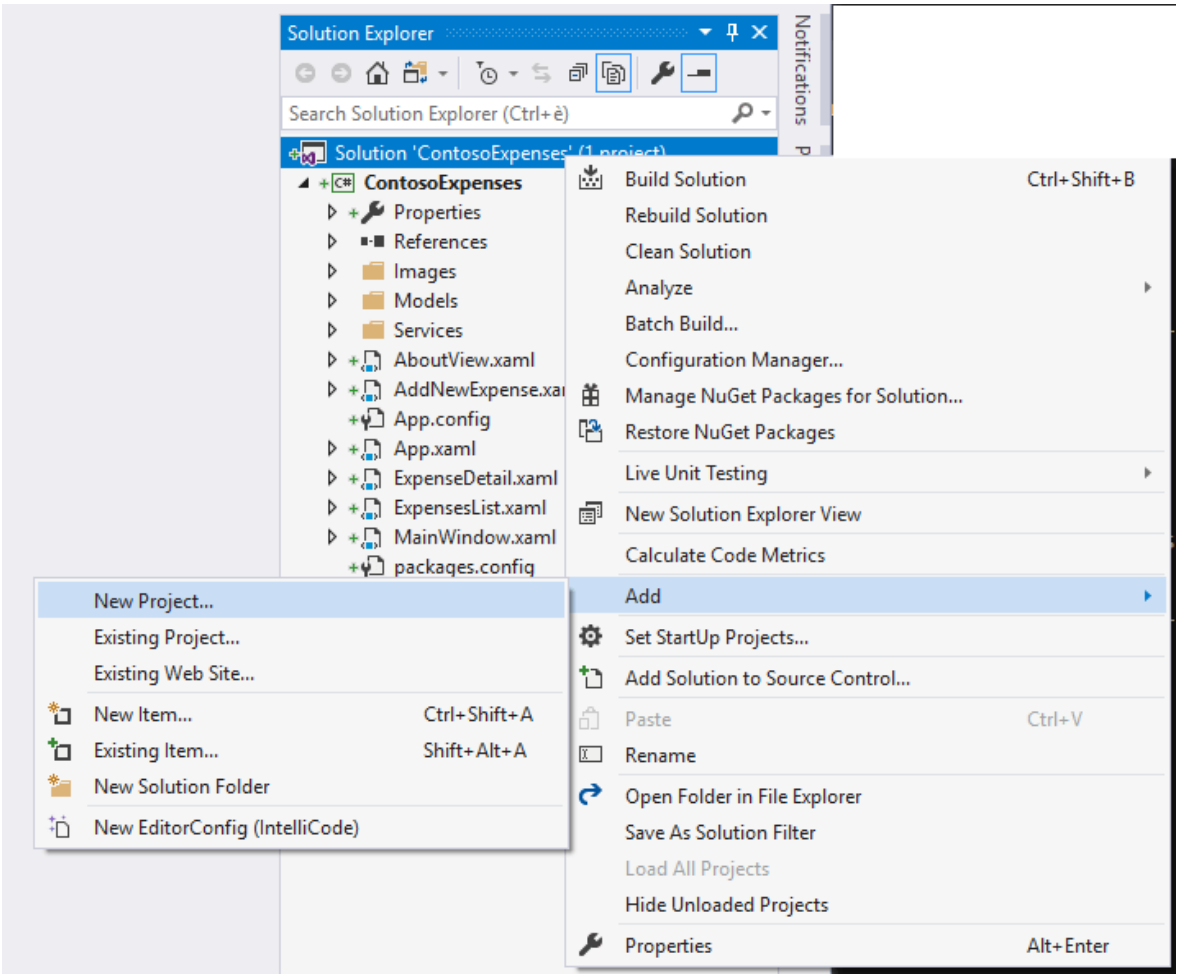
- Optimized network usage and storage space.
- Complete clean uninstall, thanks to a lightweight container where the app is executed. No registry keys and temporary files are left on the system.
- Decouples OS updates from application updates and customizations.
- Simplifies the install, update, and uninstall process.

In this part of the tutorial you'll learn how to package the Contoso Expenses app in an MSIX package.

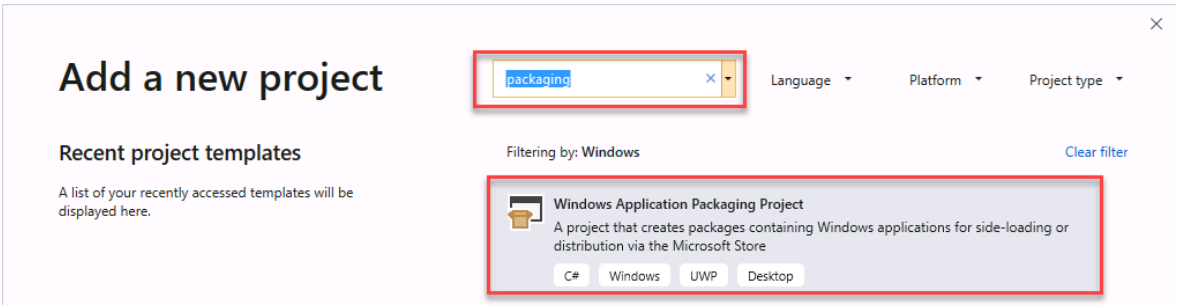
Package the application

Visual Studio 2019 provides an easy way to package a desktop application by using the Windows Application Packaging Project.

1. In **Solution Explorer**, right-click the **ContosoExpenses** solution and choose **Add -> New project**.



2. In the **Add a new project** dialog box, search for `packaging`, choose the **Windows Application Packaging Project** project template in the C# category, and click **Next**.



3. Name the new project `ContosoExpenses.Package` and click **Create**.

4. Select **Windows 10, version 1903 (10.0; Build 18362)** for both the **Target version** and **Minimum version** and click **OK**.

   The **ContosoExpenses.Package** project is added to the **ContosoExpenses** solution. This project includes a package manifest, which describes the application, and some default assets that are used for items such as the icon in the Programs menu and the tile in the Start screen. However, unlike a UWP project, the packaging project doesn't contain code. Its purpose is to package an existing desktop app.

5. In the **ContosoExpenses.Package** project, right-click the **Applications** node and choose **Add reference**. This node specifies which applications in your solution will be included in the package.

6. In the list of projects, select **ContosoExpenses.Core** and click **OK**.

7. Expand the **Applications** node and confirm that the **ContosoExpense.Core** project is referenced and highlighted in bold. This means that it will be used as a starting point for the package.

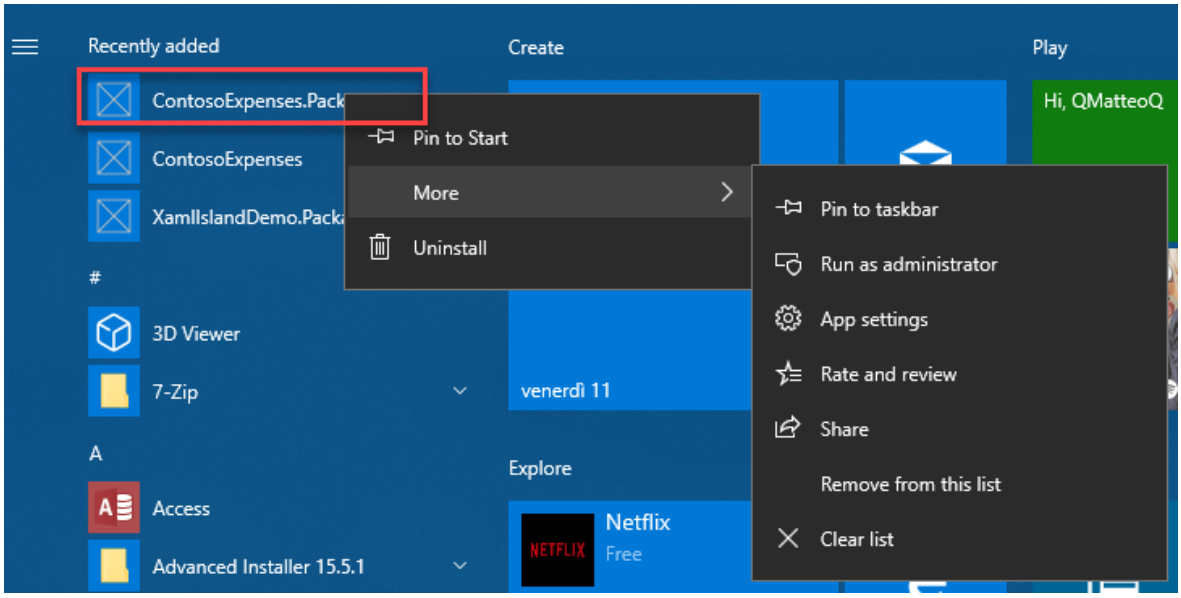
8. Right-click the **ContosoExpenses.Package** project and choose **Set As Startup Project**.

9. Press **F5** to start the packaged app in the debugger.

At this point, you can notice some changes that indicate the app is now running as packaged:

- The icon in the taskbar or in the Start menu is now the default asset that is included in every **Windows Application Packaging Project**.

- If you right-click the **ContosoExpense.Package** application listed in the Start menu, you'll notice options that are typically reserved for apps downloaded from the Microsoft Store, such as **App settings**, **Rate and review** and **Share**.

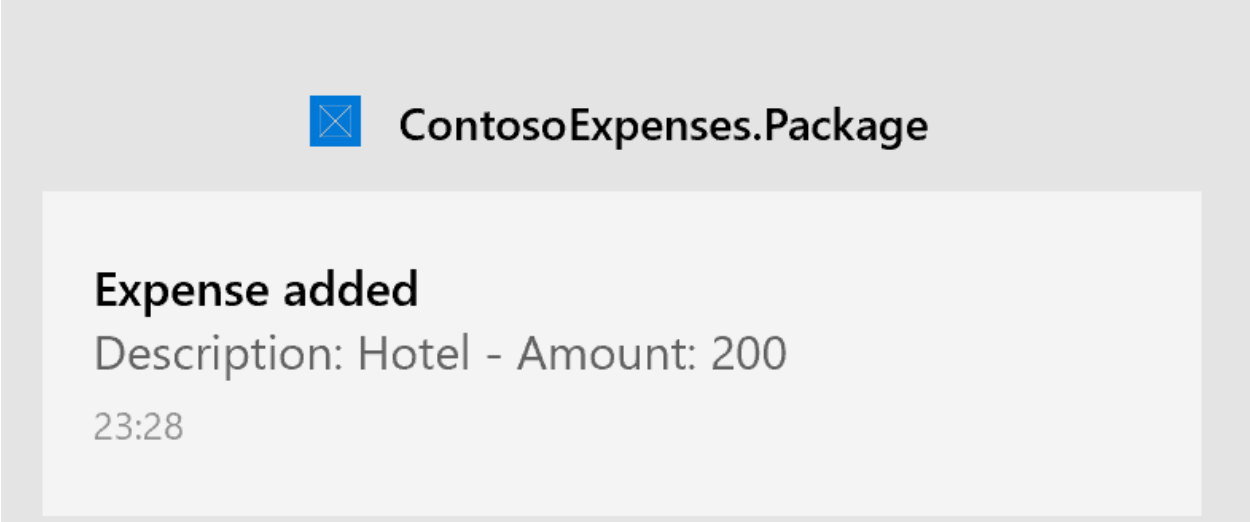


- If you want to uninstall the app, you can right-click **ContosoExpense.Package** in the Start menu and choose **Uninstall**. The app will be immediately removed, without leaving any leftover on the system.

Test the notification

Now that you have packaged the Contoso Expenses app with MSIX, you can test the notification scenario which wasn't working at the end of part 4.

1. In the Contoso Expenses app, choose an employee from the list and then click the **Add new expense** button.
2. Complete all fields in the form and press **Save**.
3. Confirm that you see an OS notification.



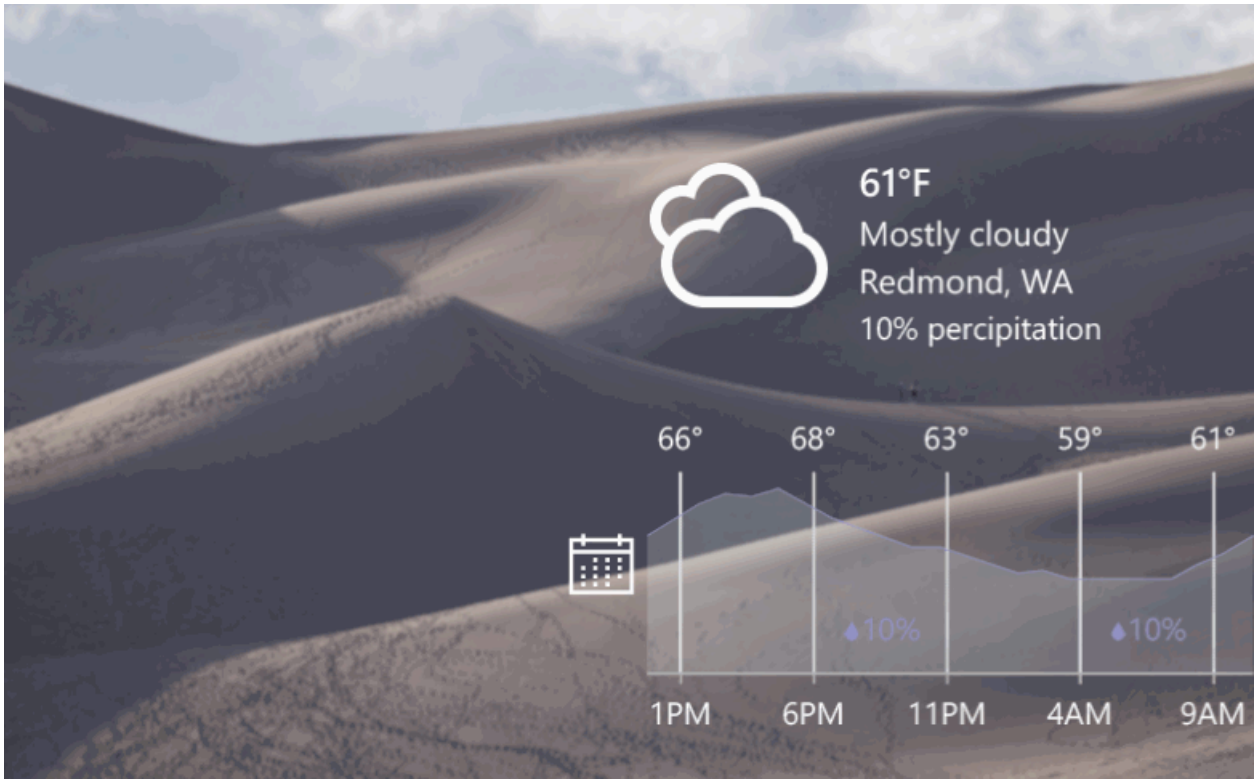
Using the Visual layer in desktop apps

Article • 08/21/2024

You can now use Windows Runtime APIs in non-UWP desktop applications to enhance the look, feel, and functionality of your WPF, Windows Forms, and C++ Win32 applications, and take advantage of the latest Windows UI features that are only available via UWP.

For many scenarios, you can use XAML islands to add modern XAML controls to your app. However, when you need to create custom experiences that go beyond the built-in controls, you can access the Visual layer APIs.

The Visual layer provides a high performance, retained-mode API for graphics, effects, and animations. It's the foundation for UI across Windows devices. UWP XAML controls are built on the Visual layer, and it enables many aspects of the Fluent Design System, such as Light, Depth, Motion, Material, and Scale.



> *User interface created with the visual layer*

Create a visually engaging user interface in any Windows app

The Visual layer lets you create engaging experiences by using lightweight compositing of custom drawn content (visuals) and applying powerful animations, effects, and

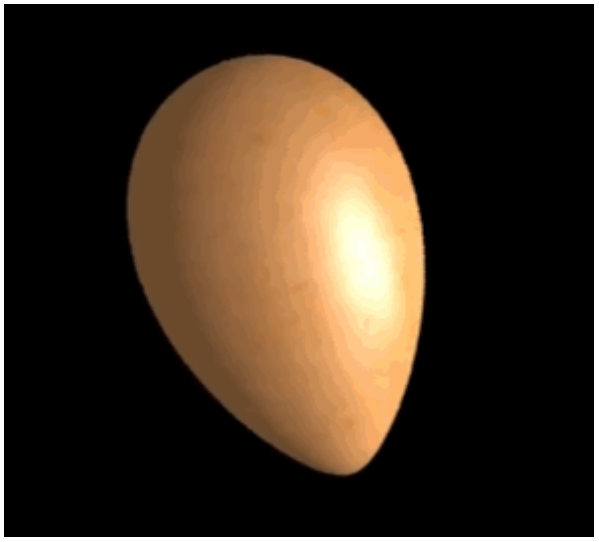
manipulations on those objects in your application. The Visual layer doesn't replace any existing UI framework; instead, it's a valuable supplement to those frameworks.

You can use the Visual layer to give your application a unique look and feel, and establish an identity that sets it apart from other applications. It also enables Fluent Design principles, which are designed to make your applications easier to use, drawing more engagement from users. For example, you can use it to create visual cues and animated screen transitions that show relationships among items on the screen.

Visual layer features

Brushes

Composition brushes let you paint UI objects with solid colors, gradients, images, videos, complex effects, and more.



> *An egg created with the Material Creator demo app.*

Effects

Composition effects include light, shadow, and a list of filter effects. They can be animated, customized, and chained, then applied directly to visuals. The SceneLightingEffect can be combined with composition lighting to create atmosphere, depth and materials.

> *Lights and material demonstrated in the Windows UI Composition sample gallery.*

Animations

Composition animations run directly in the compositor process, independent of the UI thread. This ensures smoothness and scale, so you can run large numbers of concurrent, explicit animations. In addition to familiar KeyFrame animations to drive property changes over time, you can use expressions to set up mathematical relationships between different properties, including user input. Input driven animations let you create UI that dynamically and fluidly responds to user input, which can result in higher user engagement.

> *Motion demonstrated in the Windows UI Composition sample gallery.*

Keep your existing codebase and adopt incrementally

The code in your existing applications represents a significant investment that you don't want to lose. You can migrate *islands* of content to use the Visual layer and keep the rest of the UI in its existing framework. This means you can make significant updates and enhancements to your application UI without needing to make extensive changes to your existing code base.

Samples and tutorials

Learn how to use the Visual layer in your applications by experimenting with our samples. These samples and tutorials help you get started using the Visual layer and show you how features work.

Win32

- Using the Visual layer with Win32 tutorial
  - Hello Composition sample
- Hello Vectors sample
- Virtual Surfaces sample
- Screen Capture sample

Windows Forms

- Using the Visual layer with Windows Forms tutorial
  - Hello Composition sample
- Visual Layer Integration sample

WPF

- Using the Visual layer with WPF tutorial
  - Hello Composition sample
- Visual Layer Integration sample
- Screen Capture sample

Limitations

While many Visual Layer features work the same when hosted in a desktop application as they do in a UWP app, some features do have limitations. Here are some of the limitations to be aware of:

- Effect chains rely on Win2D for the effect descriptions. The Win2D NuGet package is not supported in desktop applications, so you would need to recompile it from the source code.
- To do hit testing, you need to do bounds calculations by walking the visual tree yourself. This is the same as the Visual Layer in UWP, except in this case there's no XAML element you can easily bind to for hit testing.
- The Visual Layer does not have a primitive for rendering text.

- When two different UI technologies are used together, such as WPF and the Visual Layer, they are each responsible for drawing their own pixels on the screen, and they can't share pixels. As a result, Visual Layer content is always rendered on top of other UI content. (This is known as the *airspace* issue.) You might need to do extra coding and testing to ensure your Visual layer content resizes with the host UI and doesn't occlude other content.
- Content hosted in a desktop application doesn't automatically resize or scale for DPI. Extra steps might required to ensure your content handles DPI changes. (See the platform specific tutorials for more info.)

Additional Resources

- Visual layer
- Composition visual
- Composition brushes
- Composition effects
- Composition animations

API reference

- Windows.UI.Composition

Using the Visual Layer with WPF

Article • 08/21/2024

You can use Windows Runtime Composition APIs (also called the Visual layer) in your Windows Presentation Foundation (WPF) apps to create modern experiences that light up for Windows users.

The complete code for this tutorial is available on GitHub: WPF HelloComposition sample.

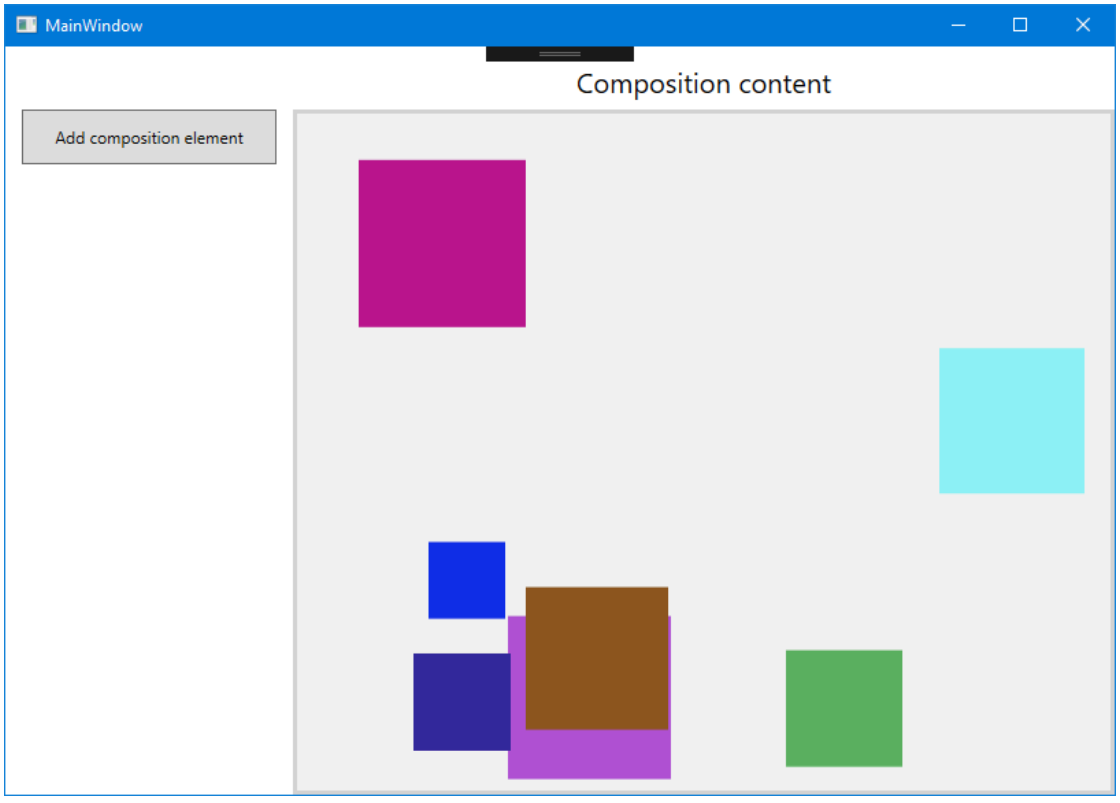
Prerequisites

The UWP XAML hosting API has these prerequisites.

- We assume that you have some familiarity with app development using WPF and UWP. For more info, see:
  - Getting Started (WPF)
  - Get started with Windows apps
  - Enhance your desktop application for Windows
- .NET Framework 4.7.2 or later
- Windows 10 version 1803 or later
- Windows 10 SDK 17134 or later

How to use Composition APIs in WPF

In this tutorial, you create a simple WPF app UI and add animated Composition elements to it. Both the WPF and Composition components are kept simple, but the interop code shown is the same regardless of the complexity of the components. The finished app looks like this.



Create a WPF project

The first step is to create the WPF app project, which includes an application definition and the XAML page for the UI.

To create a new WPF Application project in Visual C# named *HelloComposition*:

1. Open Visual Studio and select **File** > **New** > **Project**.

   The **New Project** dialog opens.

2. Under the **Installed** category, expand the **Visual C#** node, and then select **Windows Desktop**.
3. Select the **WPF App (.NET Framework)** template.
4. Enter the name *HelloComposition*, select Framework **.NET Framework 4.7.2**, then click **OK**.

   Visual Studio creates the project and opens the designer for the default application window named MainWindow.xaml.

Configure the project to use Windows Runtime APIs

To use Windows Runtime (WinRT) APIs in your WPF app, you need to configure your Visual Studio project to access the Windows Runtime. In addition, vectors are used extensively by the Composition APIs, so you need to add the references required to use vectors.

NuGet packages are available to address both of these needs. Install the latest versions of these packages to add the necessary references to your project.

- Microsoft.Windows.SDK.Contracts (Requires default package management format set to PackageReference.)
- System.Numerics.Vectors

> **Note**
>
> While we recommend using the NuGet packages to configure your project, you can add the required references manually. For more info, see Enhance your desktop application for Windows. The following table shows the files that you need to add references to.

Expand table

File	Location
System.Runtime.WindowsRuntime	C:\Windows\Microsoft.NET\Framework\v4.0.30319
Windows.Foundation.UniversalApiContract.winmd	C:\Program Files (x86)\Windows Kits\10\References\<*sdk version*>\Windows.Foundation.UniversalApiContract\<*version*>
Windows.Foundation.FoundationContract.winmd	C:\Program Files (x86)\Windows Kits\10\References\<*sdk version*>\Windows.Foundation.FoundationContract\<*version*>
System.Numerics.Vectors.dll	C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Numerics.Vectors\v4.0_4.0.0.0__b03f5f7f11d50a3a
System.Numerics.dll	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework.NETFramework\v4.7.2

Configure the project to be per-monitor DPI aware

The visual layer content you add to your app does not automatically scale to match the DPI settings of the screen it's shown on. You need to enable per-monitor DPI awareness for your app, and then make sure that the code you use to create your visual layer content takes into account the current DPI scale when the app runs. Here, we configure the project to be DPI aware. In later sections, we show how to use the DPI information to scale the visual layer content.

WPF apps are System DPI aware by default, but need to declare themselves to be per-monitor DPI aware in an app.manifest file. To turn on Windows-level per-monitor DPI awareness in the app manifest file:

1. In **Solution Explorer**, right click the *HelloComposition* project.
2. In the context menu, select **Add** > **New Item...**.
3. In the **Add New Item** dialog, select 'Application Manifest File', then click **Add**. (You can leave the default name.)
4. In the app.manifest file, find this xml and un-comment it:

```xaml
<application xmlns="urn:schemas-microsoft-com:asm.v3">
    <windowsSettings>
      <dpiAware xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">true</dpiAware>
```

```
    </windowsSettings>
  </application>
```

5. Add this setting after the opening `<windowsSettings>` tag:

```
XAML

      <dpiAwareness xmlns="http://schemas.microsoft.com/SMI/2016/WindowsSettings">PerMonitor</dpiAwareness>
```

6. You also need to set the **DoNotScaleForDpiChanges** setting in the App.config file.

   Open App.Config and add this xml inside the `<configuration>` element:

```
XML

<runtime>
  <AppContextSwitchOverrides value="Switch.System.Windows.DoNotScaleForDpiChanges=false"/>
</runtime>
```

> **Note**
>
> **AppContextSwitchOverrides** can only be set once. If your application already has one set, you must semicolon delimit this switch inside the value attribute.

(For more info, see the Per Monitor DPI Developer Guide and samples on GitHub.)

Create an HwndHost derived class to host composition elements

To host content you create with the visual layer, you need to create a class that derives from HwndHost. This is where you do most of the configuration for hosting Composition APIs. In this class, you use Platform Invocation Services (PInvoke) and COM Interop to bring Composition APIs into your WPF app. For more info about PInvoke and COM Interop, see Interoperating with unmanaged code.

> **Tip**
>
> If you need to, check the complete code at the end of the tutorial to make sure all the code is in the right places as you work through the tutorial.

1. Add a new class file to your project that derives from HwndHost.

   - In **Solution Explorer**, right click the *HelloComposition* project.
   - In the context menu, select **Add** > **Class...**.
   - In the **Add New Item** dialog, name the class *CompositionHost.cs*, then click **Add**.

2. In CompositionHost.cs, edit the class definition to derive from **HwndHost**.

```
C#

// Add
// using System.Windows.Interop;

namespace HelloComposition
{
    class CompositionHost : HwndHost
    {
    }
}
```

3. Add the following code and constructor to the class.

```
C#
```

```
// Add
// using Windows.UI.Composition;

IntPtr hwndHost;
int hostHeight, hostWidth;
object dispatcherQueue;
ICompositionTarget compositionTarget;

public Compositor Compositor { get; private set; }

public Visual Child
{
    set
    {
        if (Compositor == null)
        {
            InitComposition(hwndHost);
        }
        compositionTarget.Root = value;
    }
}

internal const int
  WS_CHILD = 0x40000000,
  WS_VISIBLE = 0x10000000,
  LBS_NOTIFY = 0x00000001,
  HOST_ID = 0x00000002,
  LISTBOX_ID = 0x00000001,
  WS_VSCROLL = 0x00200000,
  WS_BORDER = 0x00800000;

public CompositionHost(double height, double width)
{
    hostHeight = (int)height;
    hostWidth = (int)width;
}
```

4. Override the **BuildWindowCore** and **DestroyWindowCore** methods.

> ⓘ **Note**
>
> In BuildWindowCore, you call the *InitializeCoreDispatcher* and *InitComposition* methods. You create these methods in the next steps.

C#

```
// Add
// using System.Runtime.InteropServices;

protected override HandleRef BuildWindowCore(HandleRef hwndParent)
{
    // Create Window
    hwndHost = IntPtr.Zero;
    hwndHost = CreateWindowEx(0, "static", "",
                              WS_CHILD | WS_VISIBLE,
                              0, 0,
                              hostWidth, hostHeight,
                              hwndParent.Handle,
                              (IntPtr)HOST_ID,
                              IntPtr.Zero,
                              0);

    // Create Dispatcher Queue
    dispatcherQueue = InitializeCoreDispatcher();

    // Build Composition tree of content
    InitComposition(hwndHost);

    return new HandleRef(this, hwndHost);
}

protected override void DestroyWindowCore(HandleRef hwnd)
{
    if (compositionTarget.Root != null)
    {
```

```
        compositionTarget.Root.Dispose();
    }
    DestroyWindow(hwnd.Handle);
}
```

- CreateWindowEx and DestroyWindow require a PInvoke declaration. Place this declaration at the end of the code for the class.

C#

```
#region PInvoke declarations

[DllImport("user32.dll", EntryPoint = "CreateWindowEx", CharSet = CharSet.Unicode)]
internal static extern IntPtr CreateWindowEx(int dwExStyle,
                                             string lpszClassName,
                                             string lpszWindowName,
                                             int style,
                                             int x, int y,
                                             int width, int height,
                                             IntPtr hwndParent,
                                             IntPtr hMenu,
                                             IntPtr hInst,
                                             [MarshalAs(UnmanagedType.AsAny)] object pvParam);

[DllImport("user32.dll", EntryPoint = "DestroyWindow", CharSet = CharSet.Unicode)]
internal static extern bool DestroyWindow(IntPtr hwnd);

#endregion PInvoke declarations
```

5. Initialize a thread with a CoreDispatcher. The core dispatcher is responsible for processing window messages and dispatching events for WinRT APIs. New instances of **CoreDispatcher** must be created on a thread that has a CoreDispatcher.

   - Create a method named *InitializeCoreDispatcher* and add code to set up the dispatcher queue.

C#

```
private object InitializeCoreDispatcher()
{
    DispatcherQueueOptions options = new DispatcherQueueOptions();
    options.apartmentType = DISPATCHERQUEUE_THREAD_APARTMENTTYPE.DQTAT_COM_STA;
    options.threadType = DISPATCHERQUEUE_THREAD_TYPE.DQTYPE_THREAD_CURRENT;
    options.dwSize = Marshal.SizeOf(typeof(DispatcherQueueOptions));

    object queue = null;
    CreateDispatcherQueueController(options, out queue);
    return queue;
}
```

   - The dispatcher queue also requires a PInvoke declaration. Place this declaration inside the *PInvoke declarations* region you created in the previous step.

C#

```
//typedef enum DISPATCHERQUEUE_THREAD_APARTMENTTYPE
//{
//    DQTAT_COM_NONE,
//    DQTAT_COM_ASTA,
//    DQTAT_COM_STA
//};
internal enum DISPATCHERQUEUE_THREAD_APARTMENTTYPE
{
    DQTAT_COM_NONE = 0,
    DQTAT_COM_ASTA = 1,
    DQTAT_COM_STA = 2
};

//typedef enum DISPATCHERQUEUE_THREAD_TYPE
//{
//    DQTYPE_THREAD_DEDICATED,
//    DQTYPE_THREAD_CURRENT
//};
internal enum DISPATCHERQUEUE_THREAD_TYPE
{
    DQTYPE_THREAD_DEDICATED = 1,
```

```
    DQTYPE_THREAD_CURRENT = 2,
};

//struct DispatcherQueueOptions
//{
//    DWORD dwSize;
//    DISPATCHERQUEUE_THREAD_TYPE threadType;
//    DISPATCHERQUEUE_THREAD_APARTMENTTYPE apartmentType;
//};
[StructLayout(LayoutKind.Sequential)]
internal struct DispatcherQueueOptions
{
    public int dwSize;

    [MarshalAs(UnmanagedType.I4)]
    public DISPATCHERQUEUE_THREAD_TYPE threadType;

    [MarshalAs(UnmanagedType.I4)]
    public DISPATCHERQUEUE_THREAD_APARTMENTTYPE apartmentType;
};

//HRESULT CreateDispatcherQueueController(
//  DispatcherQueueOptions options,
//  ABI::Windows::System::IDispatcherQueueController** dispatcherQueueController
//);
[DllImport("coremessaging.dll", EntryPoint = "CreateDispatcherQueueController", CharSet = CharSet.Unicode)]
internal static extern IntPtr CreateDispatcherQueueController(DispatcherQueueOptions options,
                                        [MarshalAs(UnmanagedType.IUnknown)]
                                        out object dispatcherQueueController);
```

You now have the dispatcher queue ready and can begin to initialize and create Composition content.

6. Initialize the Compositor. The Compositor is a factory that creates a variety of types in the Windows.UI.Composition namespace spanning visuals, the effects system, and the animation system. The Compositor class also manages the lifetime of objects created from the factory.

C#

```
private void InitComposition(IntPtr hwndHost)
{
    ICompositorDesktopInterop interop;

    compositor = new Compositor();
    object iunknown = compositor as object;
    interop = (ICompositorDesktopInterop)iunknown;
    IntPtr raw;
    interop.CreateDesktopWindowTarget(hwndHost, true, out raw);

    object rawObject = Marshal.GetObjectForIUnknown(raw);
    ICompositionTarget target = (ICompositionTarget)rawObject;

    if (raw == null) { throw new Exception("QI Failed"); }
}
```

- **ICompositorDesktopInterop** and **ICompositionTarget** require COM imports. Place this code after the *CompositionHost* class, but inside the namespace declaration.

C#

```
#region COM Interop

/*
#undef INTERFACE
#define INTERFACE ICompositorDesktopInterop
    DECLARE_INTERFACE_IID_(ICompositorDesktopInterop, IUnknown, "29E691FA-4567-4DCA-B319-D0F207EB6807")
    {
        IFACEMETHOD(CreateDesktopWindowTarget)(
            _In_ HWND hwndTarget,
            _In_ BOOL isTopmost,
            _COM_Outptr_ IDesktopWindowTarget * *result
            ) PURE;
    };
*/
[ComImport]
[Guid("29E691FA-4567-4DCA-B319-D0F207EB6807")]
[InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
```

```
public interface ICompositorDesktopInterop
{
    void CreateDesktopWindowTarget(IntPtr hwndTarget, bool isTopmost, out IntPtr test);
}

//[contract(Windows.Foundation.UniversalApiContract, 2.0)]
//[exclusiveto(Windows.UI.Composition.CompositionTarget)]
//[uuid(A1BEA8BA - D726 - 4663 - 8129 - 6B5E7927FFA6)]
//interface ICompositionTarget : IInspectable
//{
//    [propget] HRESULT Root([out] [retval] Windows.UI.Composition.Visual** value);
//    [propput] HRESULT Root([in] Windows.UI.Composition.Visual* value);
//}

[ComImport]
[Guid("A1BEA8BA-D726-4663-8129-6B5E7927FFA6")]
[InterfaceType(ComInterfaceType.InterfaceIsIInspectable)]
public interface ICompositionTarget
{
    Windows.UI.Composition.Visual Root
    {
        get;
        set;
    }
}

#endregion COM Interop
```

Create a UserControl to add your content to the WPF visual tree

The last step to set up the infrastructure required to host Composition content is to add the HwndHost to the WPF visual tree.

Create a UserControl

A UserControl is a convenient way to package your code that creates and manages Composition content, and easily add the content to your XAML.

1. Add a new user control file to your project.

   - In **Solution Explorer**, right click the *HelloComposition* project.
   - In the context menu, select **Add** > **User Control...**.
   - In the **Add New Item** dialog, name the user control *CompositionHostControl.xaml*, then click **Add**.

   Both the CompositionHostControl.xaml and CompositionHostControl.xaml.cs files are created and added to your project.

2. In CompositionHostControl.xaml, replace the `<Grid> </Grid>` tags with this **Border** element, which is the XAML container that your HwndHost will go in.

   XAML

   ```
   <Border Name="CompositionHostElement"/>
   ```

In the code for the user control, you create an instance of the CompositionHost class you created in the previous step and add it as a child element of *CompositionHostElement*, the Border you created in the XAML page.

1. In CompositionHostControl.xaml.cs, add private variables for the objects you'll use in your Composition code. Add these after the class definition.

   C#

   ```
   CompositionHost compositionHost;
   Compositor compositor;
   Windows.UI.Composition.ContainerVisual containerVisual;
   DpiScale currentDpi;
   ```

2. Add a handler for the user control's **Loaded** event. This is where you set up your CompositionHost instance.

- In the constructor, hook up the event handler as shown here (`Loaded += CompositionHostControl_Loaded;`).

```
C#

public CompositionHostControl()
{
    InitializeComponent();
    Loaded += CompositionHostControl_Loaded;
}
```

- Add the event handler method with the name *CompositionHostControl_Loaded*.

```
C#

private void CompositionHostControl_Loaded(object sender, RoutedEventArgs e)
{
    // If the user changes the DPI scale setting for the screen the app is on,
    // the CompositionHostControl is reloaded. Don't redo this set up if it's
    // already been done.
    if (compositionHost is null)
    {
        currentDpi = VisualTreeHelper.GetDpi(this);

        compositionHost =
            new CompositionHost(ControlHostElement.ActualHeight, ControlHostElement.ActualWidth);
        ControlHostElement.Child = compositionHost;
        compositor = compositionHost.Compositor;
        containerVisual = compositor.CreateContainerVisual();
        compositionHost.Child = containerVisual;
    }
}
```

In this method, you set up the objects you'll use in your Composition code. Here's a quick look at what's happening.

- First, make sure the set up is only done once by checking whether an instance of CompositionHost already exists.

```
C#

// If the user changes the DPI scale setting for the screen the app is on,
// the CompositionHostControl is reloaded. Don't redo this set up if it's
// already been done.
if (compositionHost is null)
{

}
```

- Get the current DPI. This is used to properly scale your Composition elements.

```
C#

currentDpi = VisualTreeHelper.GetDpi(this);
```

- Create an instance of CompositionHost and assign it as the Child of the Border, *CompositionHostElement*.

```
C#

compositionHost =
    new CompositionHost(ControlHostElement.ActualHeight, ControlHostElement.ActualWidth);
ControlHostElement.Child = compositionHost;
```

- Get the Compositor from the CompositionHost.

```
C#

compositor = compositionHost.Compositor;
```

- Use the Compositor to create a container visual. This is the Composition container that you add your Composition elements to.

```
C#
```

```
containerVisual = compositor.CreateContainerVisual();
compositionHost.Child = containerVisual;
```

Add composition elements

With the infrastructure in place, you can now generate the Composition content you want to show.

For this example, you add code that creates and animates a simple square SpriteVisual.

1. Add a composition element. In CompositionHostControl.xaml.cs, add these methods to the CompositionHostControl class.

C#

```
// Add
// using System.Numerics;

public void AddElement(float size, float offsetX, float offsetY)
{
    var visual = compositor.CreateSpriteVisual();
    visual.Size = new Vector2(size, size);
    visual.Scale = new Vector3((float)currentDpi.DpiScaleX, (float)currentDpi.DpiScaleY, 1);
    visual.Brush = compositor.CreateColorBrush(GetRandomColor());
    visual.Offset = new Vector3(offsetX * (float)currentDpi.DpiScaleX, offsetY * (float)currentDpi.DpiScaleY, 0);

    containerVisual.Children.InsertAtTop(visual);

    AnimateSquare(visual, 3);
}

private void AnimateSquare(SpriteVisual visual, int delay)
{
    float offsetX = (float)(visual.Offset.X); // Already adjusted for DPI.

    // Adjust values for DPI scale, then find the Y offset that aligns the bottom of the square
    // with the bottom of the host container. This is the value to animate to.
    var hostHeightAdj = CompositionHostElement.ActualHeight * currentDpi.DpiScaleY;
    var squareSizeAdj = visual.Size.Y * currentDpi.DpiScaleY;
    float bottom = (float)(hostHeightAdj - squareSizeAdj);

    // Create the animation only if it's needed.
    if (visual.Offset.Y != bottom)
    {
        Vector3KeyFrameAnimation animation = compositor.CreateVector3KeyFrameAnimation();
        animation.InsertKeyFrame(1f, new Vector3(offsetX, bottom, 0f));
        animation.Duration = TimeSpan.FromSeconds(2);
        animation.DelayTime = TimeSpan.FromSeconds(delay);
        visual.StartAnimation("Offset", animation);
    }
}

private Windows.UI.Color GetRandomColor()
{
    Random random = new Random();
    byte r = (byte)random.Next(0, 255);
    byte g = (byte)random.Next(0, 255);
    byte b = (byte)random.Next(0, 255);
    return Windows.UI.Color.FromArgb(255, r, g, b);
}
```

Handle DPI changes

The code to add and animate an element takes into account the current DPI scale when elements are created, but you also need to account for DPI changes while the app is running. You can handle the HwndHost.DpiChanged event to be notified of changes and adjust your calculations based on the new DPI.

1. In the CompositionHostControl_Loaded method, after the last line, add this to hook up the DpiChanged event handler.

C#

```
compositionHost.DpiChanged += CompositionHost_DpiChanged;
```

2. Add the event handler method with the name *CompositionHostDpiChanged*. This code adjusts the scale and offset of each element, and recalculates any animations that aren't complete.

C#

```
private void CompositionHost_DpiChanged(object sender, DpiChangedEventArgs e)
{
    currentDpi = e.NewDpi;
    Vector3 newScale = new Vector3((float)e.NewDpi.DpiScaleX, (float)e.NewDpi.DpiScaleY, 1);

    foreach (SpriteVisual child in containerVisual.Children)
    {
        child.Scale = newScale;
        var newOffsetX = child.Offset.X * ((float)e.NewDpi.DpiScaleX / (float)e.OldDpi.DpiScaleX);
        var newOffsetY = child.Offset.Y * ((float)e.NewDpi.DpiScaleY / (float)e.OldDpi.DpiScaleY);
        child.Offset = new Vector3(newOffsetX, newOffsetY, 1);

        // Adjust animations for DPI change.
        AnimateSquare(child, 0);
    }
}
```

Add the user control to your XAML page

Now, you can add the user control to your XAML UI.

1. In MainWindow.xaml, set the Window Height to 600 and the Width to 840.

2. Add the XAML for the UI. In MainWindow.xaml, add this XAML between the root `<Grid> </Grid>` tags.

XAML

```
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="210"/>
    <ColumnDefinition Width="600"/>
    <ColumnDefinition/>
</Grid.ColumnDefinitions>
<Grid.RowDefinitions>
    <RowDefinition Height="46"/>
    <RowDefinition/>
</Grid.RowDefinitions>
<Button Content="Add composition element" Click="Button_Click"
        Grid.Row="1" Margin="12,0"
        VerticalAlignment="Top" Height="40"/>
<TextBlock Text="Composition content" FontSize="20"
           Grid.Column="1" Margin="0,12,0,4"
           HorizontalAlignment="Center"/>
<local:CompositionHostControl x:Name="CompositionHostControl1"
                              Grid.Row="1" Grid.Column="1"
                              VerticalAlignment="Top"
                              Width="600" Height="500"
                              BorderBrush="LightGray"
                              BorderThickness="3"/>
```

3. Handle the button click to create new elements. (The Click event is already hooked up in the XAML.)

   In MainWindow.xaml.cs, add this *Button_Click* event handler method. This code calls *CompositionHost.AddElement* to create a new element with a randomly generated size and offset.

C#

```
// Add
// using System;

private void Button_Click(object sender, RoutedEventArgs e)
{
    Random random = new Random();
    float size = random.Next(50, 150);
    float offsetX = random.Next(0, (int)(CompositionHostControl1.ActualWidth - size));
    float offsetY = random.Next(0, (int)(CompositionHostControl1.ActualHeight/2 - size));
    CompositionHostControl1.AddElement(size, offsetX, offsetY);
}
```

You can now build and run your WPF app. If you need to, check the complete code at the end of the tutorial to make sure all the code is in the right places.

When you run the app and click the button, you should see animated squares added to the UI.

Next steps

For a more complete example that builds on the same infrastructure, see the WPF Visual layer integration sample on GitHub.

Additional resources

- Getting Started (WPF) (.NET)
- Interoperating with unmanaged code (.NET)
- Get started with Windows apps (UWP)
- Enhance your desktop application for Windows (UWP)
- Windows.UI.Composition namespace (UWP)

Complete code

Here's the complete code for this tutorial.

MainWindow.xaml

XAML

```
<Window x:Class="HelloComposition.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:HelloComposition"
        mc:Ignorable="d"
        Title="MainWindow" Height="600" Width="840">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="210"/>
            <ColumnDefinition Width="600"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="46"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <Button Content="Add composition element" Click="Button_Click"
                Grid.Row="1" Margin="12,0"
                VerticalAlignment="Top" Height="40"/>
        <TextBlock Text="Composition content" FontSize="20"
                   Grid.Column="1" Margin="0,12,0,4"
                   HorizontalAlignment="Center"/>
        <local:CompositionHostControl x:Name="CompositionHostControl1"
                                      Grid.Row="1" Grid.Column="1"
                                      VerticalAlignment="Top"
                                      Width="600" Height="500"
                                      BorderBrush="LightGray" BorderThickness="3"/>
    </Grid>
</Window>
```

MainWindow.xaml.cs

C#

```
using System;
using System.Windows;

namespace HelloComposition
{
    /// <summary>
```

```
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            Random random = new Random();
            float size = random.Next(50, 150);
            float offsetX = random.Next(0, (int)(CompositionHostControl1.ActualWidth - size));
            float offsetY = random.Next(0, (int)(CompositionHostControl1.ActualHeight/2 - size));
            CompositionHostControl1.AddElement(size, offsetX, offsetY);
        }
    }
}
```

CompositionHostControl.xaml

XAML

```
<UserControl x:Class="HelloComposition.CompositionHostControl"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:HelloComposition"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Border Name="CompositionHostElement"/>
</UserControl>
```

CompositionHostControl.xaml.cs

C#

```
using System;
using System.Numerics;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Media;
using Windows.UI.Composition;

namespace HelloComposition
{
    /// <summary>
    /// Interaction logic for CompositionHostControl.xaml
    /// </summary>
    public partial class CompositionHostControl : UserControl
    {
        CompositionHost compositionHost;
        Compositor compositor;
        Windows.UI.Composition.ContainerVisual containerVisual;
        DpiScale currentDpi;

        public CompositionHostControl()
        {
            InitializeComponent();
            Loaded += CompositionHostControl_Loaded;
        }

        private void CompositionHostControl_Loaded(object sender, RoutedEventArgs e)
        {
            // If the user changes the DPI scale setting for the screen the app is on,
            // the CompositionHostControl is reloaded. Don't redo this set up if it's
            // already been done.
            if (compositionHost is null)
            {
                currentDpi = VisualTreeHelper.GetDpi(this);

                compositionHost = new CompositionHost(CompositionHostElement.ActualHeight,
```

```
CompositionHostElement.ActualWidth);
                CompositionHostElement.Child = compositionHost;
                compositor = compositionHost.Compositor;
                containerVisual = compositor.CreateContainerVisual();
                compositionHost.Child = containerVisual;
            }
        }

        protected override void OnDpiChanged(DpiScale oldDpi, DpiScale newDpi)
        {
            base.OnDpiChanged(oldDpi, newDpi);
            currentDpi = newDpi;
            Vector3 newScale = new Vector3((float)newDpi.DpiScaleX, (float)newDpi.DpiScaleY, 1);

            foreach (SpriteVisual child in containerVisual.Children)
            {
                child.Scale = newScale;
                var newOffsetX = child.Offset.X * ((float)newDpi.DpiScaleX / (float)oldDpi.DpiScaleX);
                var newOffsetY = child.Offset.Y * ((float)newDpi.DpiScaleY / (float)oldDpi.DpiScaleY);
                child.Offset = new Vector3(newOffsetX, newOffsetY, 1);

                // Adjust animations for DPI change.
                AnimateSquare(child, 0);
            }
        }

        public void AddElement(float size, float offsetX, float offsetY)
        {
            var visual = compositor.CreateSpriteVisual();
            visual.Size = new Vector2(size, size);
            visual.Scale = new Vector3((float)currentDpi.DpiScaleX, (float)currentDpi.DpiScaleY, 1);
            visual.Brush = compositor.CreateColorBrush(GetRandomColor());
            visual.Offset = new Vector3(offsetX * (float)currentDpi.DpiScaleX, offsetY * (float)currentDpi.DpiScaleY,
0);

            containerVisual.Children.InsertAtTop(visual);

            AnimateSquare(visual, 3);
        }

        private void AnimateSquare(SpriteVisual visual, int delay)
        {
            float offsetX = (float)(visual.Offset.X); // Already adjusted for DPI.

            // Adjust values for DPI scale, then find the Y offset that aligns the bottom of the square
            // with the bottom of the host container. This is the value to animate to.
            var hostHeightAdj = CompositionHostElement.ActualHeight * currentDpi.DpiScaleY;
            var squareSizeAdj = visual.Size.Y * currentDpi.DpiScaleY;
            float bottom = (float)(hostHeightAdj - squareSizeAdj);

            // Create the animation only if it's needed.
            if (visual.Offset.Y != bottom)
            {
                Vector3KeyFrameAnimation animation = compositor.CreateVector3KeyFrameAnimation();
                animation.InsertKeyFrame(1f, new Vector3(offsetX, bottom, 0f));
                animation.Duration = TimeSpan.FromSeconds(2);
                animation.DelayTime = TimeSpan.FromSeconds(delay);
                visual.StartAnimation("Offset", animation);
            }
        }

        private Windows.UI.Color GetRandomColor()
        {
            Random random = new Random();
            byte r = (byte)random.Next(0, 255);
            byte g = (byte)random.Next(0, 255);
            byte b = (byte)random.Next(0, 255);
            return Windows.UI.Color.FromArgb(255, r, g, b);
        }
    }
}
```

CompositionHost.cs

C#

```
using System;
using System.Runtime.InteropServices;
using System.Windows.Interop;
using Windows.UI.Composition;

namespace HelloComposition
{
    class CompositionHost : HwndHost
    {
        IntPtr hwndHost;
        int hostHeight, hostWidth;
        object dispatcherQueue;
        ICompositionTarget compositionTarget;

        public Compositor Compositor { get; private set; }

        public Visual Child
        {
            set
            {
                if (Compositor == null)
                {
                    InitComposition(hwndHost);
                }
                compositionTarget.Root = value;
            }
        }

        internal const int
          WS_CHILD = 0x40000000,
          WS_VISIBLE = 0x10000000,
          LBS_NOTIFY = 0x00000001,
          HOST_ID = 0x00000002,
          LISTBOX_ID = 0x00000001,
          WS_VSCROLL = 0x00200000,
          WS_BORDER = 0x00800000;

        public CompositionHost(double height, double width)
        {
            hostHeight = (int)height;
            hostWidth = (int)width;
        }

        protected override HandleRef BuildWindowCore(HandleRef hwndParent)
        {
            // Create Window
            hwndHost = IntPtr.Zero;
            hwndHost = CreateWindowEx(0, "static", "",
                                      WS_CHILD | WS_VISIBLE,
                                      0, 0,
                                      hostWidth, hostHeight,
                                      hwndParent.Handle,
                                      (IntPtr)HOST_ID,
                                      IntPtr.Zero,
                                      0);

            // Create Dispatcher Queue
            dispatcherQueue = InitializeCoreDispatcher();

            // Build Composition Tree of content
            InitComposition(hwndHost);

            return new HandleRef(this, hwndHost);
        }

        protected override void DestroyWindowCore(HandleRef hwnd)
        {
            if (compositionTarget.Root != null)
            {
                compositionTarget.Root.Dispose();
            }
            DestroyWindow(hwnd.Handle);
        }

        private object InitializeCoreDispatcher()
        {
            DispatcherQueueOptions options = new DispatcherQueueOptions();
            options.apartmentType = DISPATCHERQUEUE_THREAD_APARTMENTTYPE.DQTAT_COM_STA;
            options.threadType = DISPATCHERQUEUE_THREAD_TYPE.DQTYPE_THREAD_CURRENT;
```

```
            options.dwSize = Marshal.SizeOf(typeof(DispatcherQueueOptions));

            object queue = null;
            CreateDispatcherQueueController(options, out queue);
            return queue;
        }

        private void InitComposition(IntPtr hwndHost)
        {
            ICompositorDesktopInterop interop;

            Compositor = new Compositor();
            object iunknown = Compositor as object;
            interop = (ICompositorDesktopInterop)iunknown;
            IntPtr raw;
            interop.CreateDesktopWindowTarget(hwndHost, true, out raw);

            object rawObject = Marshal.GetObjectForIUnknown(raw);
            compositionTarget = (ICompositionTarget)rawObject;

            if (raw == null) { throw new Exception("QI Failed"); }
        }

        #region PInvoke declarations

        //typedef enum DISPATCHERQUEUE_THREAD_APARTMENTTYPE
        //{
        //    DQTAT_COM_NONE,
        //    DQTAT_COM_ASTA,
        //    DQTAT_COM_STA
        //};
        internal enum DISPATCHERQUEUE_THREAD_APARTMENTTYPE
        {
            DQTAT_COM_NONE = 0,
            DQTAT_COM_ASTA = 1,
            DQTAT_COM_STA = 2
        };

        //typedef enum DISPATCHERQUEUE_THREAD_TYPE
        //{
        //    DQTYPE_THREAD_DEDICATED,
        //    DQTYPE_THREAD_CURRENT
        //};
        internal enum DISPATCHERQUEUE_THREAD_TYPE
        {
            DQTYPE_THREAD_DEDICATED = 1,
            DQTYPE_THREAD_CURRENT = 2,
        };

        //struct DispatcherQueueOptions
        //{
        //    DWORD dwSize;
        //    DISPATCHERQUEUE_THREAD_TYPE threadType;
        //    DISPATCHERQUEUE_THREAD_APARTMENTTYPE apartmentType;
        //};
        [StructLayout(LayoutKind.Sequential)]
        internal struct DispatcherQueueOptions
        {
            public int dwSize;

            [MarshalAs(UnmanagedType.I4)]
            public DISPATCHERQUEUE_THREAD_TYPE threadType;

            [MarshalAs(UnmanagedType.I4)]
            public DISPATCHERQUEUE_THREAD_APARTMENTTYPE apartmentType;
        };

        //HRESULT CreateDispatcherQueueController(
        //  DispatcherQueueOptions options,
        //  ABI::Windows::System::IDispatcherQueueController** dispatcherQueueController
        //);
        [DllImport("coremessaging.dll", EntryPoint = "CreateDispatcherQueueController", CharSet = CharSet.Unicode)]
        internal static extern IntPtr CreateDispatcherQueueController(DispatcherQueueOptions options,
                                            [MarshalAs(UnmanagedType.IUnknown)]
                                           out object dispatcherQueueController);


        [DllImport("user32.dll", EntryPoint = "CreateWindowEx", CharSet = CharSet.Unicode)]
        internal static extern IntPtr CreateWindowEx(int dwExStyle,
                                                     string lpszClassName,
```

```
                                                string lpszWindowName,
                                                int style,
                                                int x, int y,
                                                int width, int height,
                                                IntPtr hwndParent,
                                                IntPtr hMenu,
                                                IntPtr hInst,
                                                [MarshalAs(UnmanagedType.AsAny)] object pvParam);

        [DllImport("user32.dll", EntryPoint = "DestroyWindow", CharSet = CharSet.Unicode)]
        internal static extern bool DestroyWindow(IntPtr hwnd);


        #endregion PInvoke declarations

    }
    #region COM Interop

    /*
    #undef INTERFACE
    #define INTERFACE ICompositorDesktopInterop
        DECLARE_INTERFACE_IID_(ICompositorDesktopInterop, IUnknown, "29E691FA-4567-4DCA-B319-D0F207EB6807")
        {
            IFACEMETHOD(CreateDesktopWindowTarget)(
                _In_ HWND hwndTarget,
                _In_ BOOL isTopmost,
                _COM_Outptr_ IDesktopWindowTarget * *result
                ) PURE;
        };
    */
    [ComImport]
    [Guid("29E691FA-4567-4DCA-B319-D0F207EB6807")]
    [InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
    public interface ICompositorDesktopInterop
    {
        void CreateDesktopWindowTarget(IntPtr hwndTarget, bool isTopmost, out IntPtr test);
    }

    //[contract(Windows.Foundation.UniversalApiContract, 2.0)]
    //[exclusiveto(Windows.UI.Composition.CompositionTarget)]
    //[uuid(A1BEA8BA - D726 - 4663 - 8129 - 6B5E7927FFA6)]
    //interface ICompositionTarget : IInspectable
    //{
    //    [propget] HRESULT Root([out] [retval] Windows.UI.Composition.Visual** value);
    //    [propput] HRESULT Root([in] Windows.UI.Composition.Visual* value);
    //}

    [ComImport]
    [Guid("A1BEA8BA-D726-4663-8129-6B5E7927FFA6")]
    [InterfaceType(ComInterfaceType.InterfaceIsIInspectable)]
    public interface ICompositionTarget
    {
        Windows.UI.Composition.Visual Root
        {
            get;
            set;
        }
    }

    #endregion COM Interop
}
```

Using the Visual Layer with Windows Forms

Article • 08/21/2024

You can use Windows Runtime Composition APIs (also called the Visual layer) in your Windows Forms apps to create modern experiences that light up for Windows users.

The complete code for this tutorial is available on GitHub: Windows Forms HelloComposition sample.

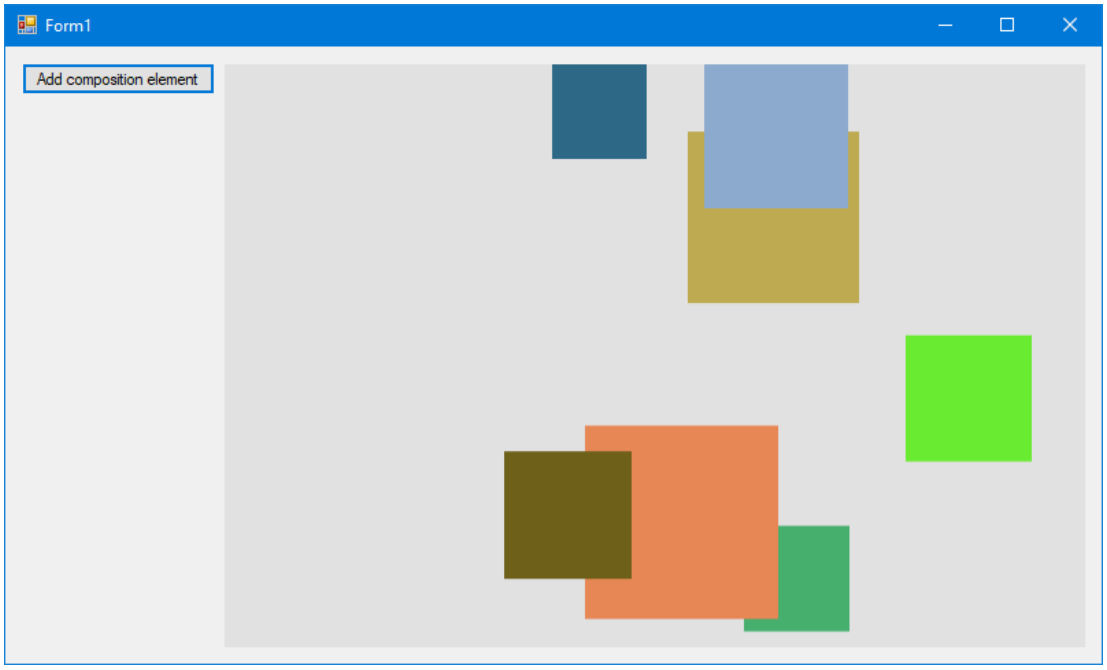
Prerequisites

The UWP hosting API has these prerequisites.

- We assume that you have some familiarity with app development using Windows Forms and UWP. For more info, see:
  - Getting Started with Windows Forms
  - Get started with Windows apps
  - Enhance your desktop application for Windows
- .NET Framework 4.7.2 or later
- Windows 10 version 1803 or later
- Windows 10 SDK 17134 or later

How to use Composition APIs in Windows Forms

In this tutorial, you create a simple Windows Forms UI and add animated Composition elements to it. Both the Windows Forms and Composition components are kept simple, but the interop code shown is the same regardless of the complexity of the components. The finished app looks like this.



Create a Windows Forms project

The first step is to create the Windows Forms app project, which includes an application definition and the main form for the UI.

To create a new Windows Forms Application project in Visual C# named *HelloComposition*:

1. Open Visual Studio and select **File** > **New** > **Project**.
   The **New Project** dialog opens.
2. Under the **Installed** category, expand the **Visual C#** node, and then select **Windows Desktop**.
3. Select the **Windows Forms App (.NET Framework)** template.
4. Enter the name *HelloComposition,* select Framework **.NET Framework 4.7.2**, then click **OK**.

Visual Studio creates the project and opens the designer for the default application window named Form1.cs.

Configure the project to use Windows Runtime APIs

To use Windows Runtime (WinRT) APIs in your Windows Forms app, you need to configure your Visual Studio project to access the Windows Runtime. In addition, vectors are used extensively by the Composition APIs, so you need to add the references required to use vectors.

NuGet packages are available to address both of these needs. Install the latest versions of these packages to add the necessary references to your project.

- Microsoft.Windows.SDK.Contracts (Requires default package management format set to PackageReference.)
- System.Numerics.Vectors

> **Note**
>
> While we recommend using the NuGet packages to configure your project, you can add the required references manually. For more info, see **Enhance your desktop application for Windows**. The following table shows the files that you need to add references to.

Expand table

File	Location
System.Runtime.WindowsRuntime	C:\Windows\Microsoft.NET\Framework\v4.0.30319
Windows.Foundation.UniversalApiContract.winmd	C:\Program Files (x86)\Windows Kits\10\References\<*sdk version*>\Windows.Foundation.UniversalApiContract\<*version*>
Windows.Foundation.FoundationContract.winmd	C:\Program Files (x86)\Windows Kits\10\References\<*sdk version*>\Windows.Foundation.FoundationContract\<*version*>
System.Numerics.Vectors.dll	C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Numerics.Vectors\v4.0_4.0.0.0__b03f5f7f11d50a3a
System.Numerics.dll	C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework.NETFramework\v4.7.2

Create a custom control to manage interop

To host content you create with the visual layer, you create a custom control derived from Control. This control gives you access to a window Handle, which you need in order to create the container for your visual layer content.

This is where you do most of the configuration for hosting Composition APIs. In this control, you use Platform Invocation Services (PInvoke) and COM Interop to bring Composition APIs into your Windows Forms app. For more info about PInvoke and COM Interop, see Interoperating with unmanaged code.

> **Tip**
>
> If you need to, check the complete code at the end of the tutorial to make sure all the code is in the right places as you work through the tutorial.

1. Add a new custom control file to your project that derives from Control.
   - In **Solution Explorer**, right click the *HelloComposition* project.
   - In the context menu, select **Add** > **New Item...**.
   - In the **Add New Item** dialog, select **Custom Control**.
   - Name the control *CompositionHost.cs*, then click **Add**. CompositionHost.cs opens in the Design view.
2. Switch to code view for CompositionHost.cs and add the following code to the class.

```
C#
```

```
// Add
// using Windows.UI.Composition;

IntPtr hwndHost;
object dispatcherQueue;
protected ContainerVisual containerVisual;
```

```
protected Compositor compositor;

private ICompositionTarget compositionTarget;

public Visual Child
{
    set
    {
        if (compositor == null)
        {
            InitComposition(hwndHost);
        }
        compositionTarget.Root = value;
    }
}
```

3. Add code to the constructor.

   In the constructor, you call the *InitializeCoreDispatcher* and *InitComposition* methods. You create these methods in the next steps.

```
C#

public CompositionHost()
{
    InitializeComponent();

    // Get the window handle.
    hwndHost = Handle;

    // Create dispatcher queue.
    dispatcherQueue = InitializeCoreDispatcher();

    // Build Composition tree of content.
    InitComposition(hwndHost);
}
```

4. Initialize a thread with a CoreDispatcher. The core dispatcher is responsible for processing window messages and dispatching events for WinRT APIs. New instances of **Compositor** must be created on a thread that has a CoreDispatcher.

   - Create a method named *InitializeCoreDispatcher* and add code to set up the dispatcher queue.

```
C#

// Add
// using System.Runtime.InteropServices;

private object InitializeCoreDispatcher()
{
    DispatcherQueueOptions options = new DispatcherQueueOptions();
    options.apartmentType = DISPATCHERQUEUE_THREAD_APARTMENTTYPE.DQTAT_COM_STA;
    options.threadType = DISPATCHERQUEUE_THREAD_TYPE.DQTYPE_THREAD_CURRENT;
    options.dwSize = Marshal.SizeOf(typeof(DispatcherQueueOptions));

    object queue = null;
    CreateDispatcherQueueController(options, out queue);
    return queue;
}
```

   - The dispatcher queue requires a PInvoke declaration. Place this declaration at the end of the code for the class. (We place this code inside a region to keep the class code tidy.)

```
C#

#region PInvoke declarations

//typedef enum DISPATCHERQUEUE_THREAD_APARTMENTTYPE
//{
//    DQTAT_COM_NONE,
//    DQTAT_COM_ASTA,
//    DQTAT_COM_STA
//};
internal enum DISPATCHERQUEUE_THREAD_APARTMENTTYPE
{
    DQTAT_COM_NONE = 0,
    DQTAT_COM_ASTA = 1,
```

```
    DQTAT_COM_STA = 2
};

//typedef enum DISPATCHERQUEUE_THREAD_TYPE
//{
//    DQTYPE_THREAD_DEDICATED,
//    DQTYPE_THREAD_CURRENT
//};
internal enum DISPATCHERQUEUE_THREAD_TYPE
{
    DQTYPE_THREAD_DEDICATED = 1,
    DQTYPE_THREAD_CURRENT = 2,
};

//struct DispatcherQueueOptions
//{
//    DWORD dwSize;
//    DISPATCHERQUEUE_THREAD_TYPE threadType;
//    DISPATCHERQUEUE_THREAD_APARTMENTTYPE apartmentType;
//};
[StructLayout(LayoutKind.Sequential)]
internal struct DispatcherQueueOptions
{
    public int dwSize;

    [MarshalAs(UnmanagedType.I4)]
    public DISPATCHERQUEUE_THREAD_TYPE threadType;

    [MarshalAs(UnmanagedType.I4)]
    public DISPATCHERQUEUE_THREAD_APARTMENTTYPE apartmentType;
};

//HRESULT CreateDispatcherQueueController(
//  DispatcherQueueOptions options,
//  ABI::Windows::System::IDispatcherQueueController** dispatcherQueueController
//);
[DllImport("coremessaging.dll", EntryPoint = "CreateDispatcherQueueController", CharSet = CharSet.Unicode)]
internal static extern IntPtr CreateDispatcherQueueController(DispatcherQueueOptions options,
                                        [MarshalAs(UnmanagedType.IUnknown)]
                                        out object dispatcherQueueController);

#endregion PInvoke declarations
```

You now have the dispatcher queue ready and can begin to initialize and create Composition content.

5. Initialize the Compositor. The Compositor is a factory that creates a variety of types in the Windows.UI.Composition namespace spanning the visual layer, effects system, and animation system. The Compositor class also manages the lifetime of objects created from the factory.

C#

```
private void InitComposition(IntPtr hwndHost)
{
    ICompositorDesktopInterop interop;

    compositor = new Compositor();
    object iunknown = compositor as object;
    interop = (ICompositorDesktopInterop)iunknown;
    IntPtr raw;
    interop.CreateDesktopWindowTarget(hwndHost, true, out raw);

    object rawObject = Marshal.GetObjectForIUnknown(raw);
    compositionTarget = (ICompositionTarget)rawObject;

    if (raw == null) { throw new Exception("QI Failed"); }

    containerVisual = compositor.CreateContainerVisual();
    Child = containerVisual;
}
```

- **ICompositorDesktopInterop** and **ICompositionTarget** require COM imports. Place this code after the *CompositionHost* class, but inside the namespace declaration.

C#

```
#region COM Interop

/*
#undef INTERFACE
#define INTERFACE ICompositorDesktopInterop
    DECLARE_INTERFACE_IID_(ICompositorDesktopInterop, IUnknown, "29E691FA-4567-4DCA-B319-D0F207EB6807")
    {
        IFACEMETHOD(CreateDesktopWindowTarget)(
            _In_ HWND hwndTarget,
            _In_ BOOL isTopmost,
            _COM_Outptr_ IDesktopWindowTarget * *result
            ) PURE;
    };
*/
[ComImport]
[Guid("29E691FA-4567-4DCA-B319-D0F207EB6807")]
[InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
public interface ICompositorDesktopInterop
{
    void CreateDesktopWindowTarget(IntPtr hwndTarget, bool isTopmost, out IntPtr test);
}

//[contract(Windows.Foundation.UniversalApiContract, 2.0)]
//[exclusiveto(Windows.UI.Composition.CompositionTarget)]
//[uuid(A1BEA8BA - D726 - 4663 - 8129 - 6B5E7927FFA6)]
//interface ICompositionTarget : IInspectable
//{
//    [propget] HRESULT Root([out] [retval] Windows.UI.Composition.Visual** value);
//    [propput] HRESULT Root([in] Windows.UI.Composition.Visual* value);
//}

[ComImport]
[Guid("A1BEA8BA-D726-4663-8129-6B5E7927FFA6")]
[InterfaceType(ComInterfaceType.InterfaceIsIInspectable)]
public interface ICompositionTarget
{
    Windows.UI.Composition.Visual Root
    {
        get;
        set;
    }
}

#endregion COM Interop
```

Create a custom control to host composition elements

It's a good idea to put the code that generates and manages your composition elements in a separate control that derives from CompositionHost. This keeps the interop code you created in the CompositionHost class reusable.

Here, you create a custom control derived from CompositionHost. This control is added to the Visual Studio toolbox so you can add it to your form.

1. Add a new custom control file to your project that derives from CompositionHost.

   - In **Solution Explorer**, right click the *HelloComposition* project.
   - In the context menu, select **Add** > **New Item...**.
   - In the **Add New Item** dialog, select **Custom Control**.
   - Name the control *CompositionHostControl.cs*, then click **Add**. CompositionHostControl.cs opens in the Design view.

2. In the Properties pane for CompositionHostControl.cs design view, set the **BackColor** property to **ControlLight**.

   Setting the background color is optional. We do it here so you can see your custom control against the form background.

3. Switch to code view for CompositionHostControl.cs and update the class declaration to derive from CompositionHost.

   ```
   C#

   class CompositionHostControl : CompositionHost
   ```

4. Update the constructor to call the base constructor.

```
C#

public CompositionHostControl() : base()
{

}
```

Add composition elements

With the infrastructure in place, you can now add Composition content to the app UI.

For this example, you add code to the CompositionHostControl class that creates and animates a simple SpriteVisual.

1. Add a composition element.

   In CompositionHostControl.cs, add these methods to the CompositionHostControl class.

```
C#

// Add
// using Windows.UI.Composition;

public void AddElement(float size, float offsetX, float offsetY)
{
    var visual = compositor.CreateSpriteVisual();
    visual.Size = new Vector2(size, size); // Requires references
    visual.Brush = compositor.CreateColorBrush(GetRandomColor());
    visual.Offset = new Vector3(offsetX, offsetY, 0);
    containerVisual.Children.InsertAtTop(visual);

    AnimateSquare(visual, 3);
}

private void AnimateSquare(SpriteVisual visual, int delay)
{
    float offsetX = (float)(visual.Offset.X);
    Vector3KeyFrameAnimation animation = compositor.CreateVector3KeyFrameAnimation();
    float bottom = Height - visual.Size.Y;
    animation.InsertKeyFrame(1f, new Vector3(offsetX, bottom, 0f));
    animation.Duration = TimeSpan.FromSeconds(2);
    animation.DelayTime = TimeSpan.FromSeconds(delay);
    visual.StartAnimation("Offset", animation);
}

private Windows.UI.Color GetRandomColor()
{
    Random random = new Random();
    byte r = (byte)random.Next(0, 255);
    byte g = (byte)random.Next(0, 255);
    byte b = (byte)random.Next(0, 255);
    return Windows.UI.Color.FromArgb(255, r, g, b);
}
```

Add the control to your form

Now that you have a custom control to host Composition content, you can add it to the app UI. Here, you add an instance of the CompositionHostControl you created in the previous step. CompositionHostControl is automatically added to the Visual Studio toolbox under ***project name*** **Components**.

1. In Form1.CS design view, add a Button to the UI.

   - Drag a Button from the toolbox onto Form1. Place it in the upper left corner of the form. (See the image at the start of the tutorial to check the placement of the controls.)
   - In the Properties pane, change the **Text** property from *button1* to *Add composition element*.
   - Resize the Button so that all the text shows.

   (For more info, see How to: Add Controls to Windows Forms.)

2. Add a CompositionHostControl to the UI.

- Drag a CompositionHostControl from the toolbox onto Form1. Place it to the right of the Button.
- Resize the CompositionHost so that it fills the remainder of the form.

3. Handle the button click event.

   - In the Properties pane, click the lightning bolt to switch to the Events view.
   - In the events list, select the **Click** event, type *Button_Click*, and press Enter.
   - This code is added in Form1.cs:

```
C#

private void Button_Click(object sender, EventArgs e)
{

}
```

4. Add code to the button click handler to create new elements.

   - In Form1.cs, add code to the *Button_Click* event handler you created previously. This code calls *CompositionHostControl1.AddElement* to create a new element with a randomly generated size and offset. (The instance of CompositionHostControl was automatically named *compositionHostControl1* when you dragged it onto the form.)

```
C#

// Add
// using System;

private void Button_Click(object sender, RoutedEventArgs e)
{
    Random random = new Random();
    float size = random.Next(50, 150);
    float offsetX = random.Next(0, (int)(compositionHostControl1.Width - size));
    float offsetY = random.Next(0, (int)(compositionHostControl1.Height/2 - size));
    compositionHostControl1.AddElement(size, offsetX, offsetY);
}
```

You can now build and run your Windows Forms app. When you click the button, you should see animated squares added to the UI.

Next steps

For a more complete example that builds on the same infrastructure, see the Windows Forms Visual layer integration sample on GitHub.

Additional resources

- Getting Started with Windows Forms (.NET)
- Interoperating with unmanaged code (.NET)
- Get started with Windows apps (UWP)
- Enhance your desktop application for Windows (UWP)
- Windows.UI.Composition namespace (UWP)

Complete code

Here's the complete code for this tutorial.

Form1.cs

```
C#

using System;
using System.Windows.Forms;

namespace HelloComposition
{
```

```
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, EventArgs e)
        {
            Random random = new Random();
            float size = random.Next(50, 150);
            float offsetX = random.Next(0, (int)(compositionHostControl1.Width - size));
            float offsetY = random.Next(0, (int)(compositionHostControl1.Height/2 - size));
            compositionHostControl1.AddElement(size, offsetX, offsetY);
        }
    }
}
```

CompositionHostControl.cs

C#

```
using System;
using System.Numerics;
using Windows.UI.Composition;

namespace HelloComposition
{
    class CompositionHostControl : CompositionHost
    {
        public CompositionHostControl() : base()
        {

        }

        public void AddElement(float size, float offsetX, float offsetY)
        {
            var visual = compositor.CreateSpriteVisual();
            visual.Size = new Vector2(size, size); // Requires references
            visual.Brush = compositor.CreateColorBrush(GetRandomColor());
            visual.Offset = new Vector3(offsetX, offsetY, 0);
            containerVisual.Children.InsertAtTop(visual);

            AnimateSquare(visual, 3);
        }

        private void AnimateSquare(SpriteVisual visual, int delay)
        {
            float offsetX = (float)(visual.Offset.X);
            Vector3KeyFrameAnimation animation = compositor.CreateVector3KeyFrameAnimation();
            float bottom = Height - visual.Size.Y;
            animation.InsertKeyFrame(1f, new Vector3(offsetX, bottom, 0f));
            animation.Duration = TimeSpan.FromSeconds(2);
            animation.DelayTime = TimeSpan.FromSeconds(delay);
            visual.StartAnimation("Offset", animation);
        }

        private Windows.UI.Color GetRandomColor()
        {
            Random random = new Random();
            byte r = (byte)random.Next(0, 255);
            byte g = (byte)random.Next(0, 255);
            byte b = (byte)random.Next(0, 255);
            return Windows.UI.Color.FromArgb(255, r, g, b);
        }
    }
}
```

CompositionHost.cs

C#

```
using System;
using System.Runtime.InteropServices;
```

```
using System.Windows.Forms;
using Windows.UI.Composition;

namespace HelloComposition
{
    public partial class CompositionHost : Control
    {
        IntPtr hwndHost;
        object dispatcherQueue;
        protected ContainerVisual containerVisual;
        protected Compositor compositor;
        private ICompositionTarget compositionTarget;

        public Visual Child
        {
            set
            {
                if (compositor == null)
                {
                    InitComposition(hwndHost);
                }
                compositionTarget.Root = value;
            }
        }

        public CompositionHost()
        {
            // Get the window handle.
            hwndHost = Handle;

            // Create dispatcher queue.
            dispatcherQueue = InitializeCoreDispatcher();

            // Build Composition tree of content.
            InitComposition(hwndHost);
        }

        private object InitializeCoreDispatcher()
        {
            DispatcherQueueOptions options = new DispatcherQueueOptions();
            options.apartmentType = DISPATCHERQUEUE_THREAD_APARTMENTTYPE.DQTAT_COM_STA;
            options.threadType = DISPATCHERQUEUE_THREAD_TYPE.DQTYPE_THREAD_CURRENT;
            options.dwSize = Marshal.SizeOf(typeof(DispatcherQueueOptions));

            object queue = null;
            CreateDispatcherQueueController(options, out queue);
            return queue;
        }

        private void InitComposition(IntPtr hwndHost)
        {
            ICompositorDesktopInterop interop;

            compositor = new Compositor();
            object iunknown = compositor as object;
            interop = (ICompositorDesktopInterop)iunknown;
            IntPtr raw;
            interop.CreateDesktopWindowTarget(hwndHost, true, out raw);

            object rawObject = Marshal.GetObjectForIUnknown(raw);
            compositionTarget = (ICompositionTarget)rawObject;

            if (raw == null) { throw new Exception("QI Failed"); }

            containerVisual = compositor.CreateContainerVisual();
            Child = containerVisual;
        }

        protected override void OnPaint(PaintEventArgs pe)
        {
            base.OnPaint(pe);
        }

        #region PInvoke declarations

        //typedef enum DISPATCHERQUEUE_THREAD_APARTMENTTYPE
        //{
        //    DQTAT_COM_NONE,
        //    DQTAT_COM_ASTA,
        //    DQTAT_COM_STA
```

```
        //};
        internal enum DISPATCHERQUEUE_THREAD_APARTMENTTYPE
        {
            DQTAT_COM_NONE = 0,
            DQTAT_COM_ASTA = 1,
            DQTAT_COM_STA = 2
        };

        //typedef enum DISPATCHERQUEUE_THREAD_TYPE
        //{
        //    DQTYPE_THREAD_DEDICATED,
        //    DQTYPE_THREAD_CURRENT
        //};
        internal enum DISPATCHERQUEUE_THREAD_TYPE
        {
            DQTYPE_THREAD_DEDICATED = 1,
            DQTYPE_THREAD_CURRENT = 2,
        };

        //struct DispatcherQueueOptions
        //{
        //    DWORD dwSize;
        //    DISPATCHERQUEUE_THREAD_TYPE threadType;
        //    DISPATCHERQUEUE_THREAD_APARTMENTTYPE apartmentType;
        //};
        [StructLayout(LayoutKind.Sequential)]
        internal struct DispatcherQueueOptions
        {
            public int dwSize;

            [MarshalAs(UnmanagedType.I4)]
            public DISPATCHERQUEUE_THREAD_TYPE threadType;

            [MarshalAs(UnmanagedType.I4)]
            public DISPATCHERQUEUE_THREAD_APARTMENTTYPE apartmentType;
        };

        //HRESULT CreateDispatcherQueueController(
        //  DispatcherQueueOptions options,
        //  ABI::Windows::System::IDispatcherQueueController** dispatcherQueueController
        //);
        [DllImport("coremessaging.dll", EntryPoint = "CreateDispatcherQueueController", CharSet = CharSet.Unicode)]
        internal static extern IntPtr CreateDispatcherQueueController(DispatcherQueueOptions options,
                                                [MarshalAs(UnmanagedType.IUnknown)]
                                        out object dispatcherQueueController);

        #endregion PInvoke declarations
    }

    #region COM Interop

    /*
    #undef INTERFACE
    #define INTERFACE ICompositorDesktopInterop
        DECLARE_INTERFACE_IID_(ICompositorDesktopInterop, IUnknown, "29E691FA-4567-4DCA-B319-D0F207EB6807")
        {
            IFACEMETHOD(CreateDesktopWindowTarget)(
                _In_ HWND hwndTarget,
                _In_ BOOL isTopmost,
                _COM_Outptr_ IDesktopWindowTarget * *result
                ) PURE;
        };
    */
    [ComImport]
    [Guid("29E691FA-4567-4DCA-B319-D0F207EB6807")]
    [InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
    public interface ICompositorDesktopInterop
    {
        void CreateDesktopWindowTarget(IntPtr hwndTarget, bool isTopmost, out IntPtr test);
    }

    //[contract(Windows.Foundation.UniversalApiContract, 2.0)]
    //[exclusiveto(Windows.UI.Composition.CompositionTarget)]
    //[uuid(A1BEA8BA - D726 - 4663 - 8129 - 6B5E7927FFA6)]
    //interface ICompositionTarget : IInspectable
    //{
    //    [propget] HRESULT Root([out] [retval] Windows.UI.Composition.Visual** value);
    //    [propput] HRESULT Root([in] Windows.UI.Composition.Visual* value);
    //}
```

```
    [ComImport]
    [Guid("A1BEA8BA-D726-4663-8129-6B5E7927FFA6")]
    [InterfaceType(ComInterfaceType.InterfaceIsIInspectable)]
    public interface ICompositionTarget
    {
        Windows.UI.Composition.Visual Root
        {
            get;
            set;
        }
    }
    #endregion COM Interop
}
```

Using the Visual Layer with Win32

Article • 08/21/2024

You can use Windows Runtime Composition APIs (also called the Visual layer) in your Win32 apps to create modern experiences that light up for Windows users.

The complete code for this tutorial is available on GitHub: Win32 HelloComposition sample.

Universal Windows Applications that need precise control over their UI composition have access to the Windows.UI.Composition namespace to exert fine grained control over how their UI is composed and rendered. This composition API is not limited to UWP apps, however. Win32 desktop applications can take advantage of the modern composition systems in UWP and Windows.

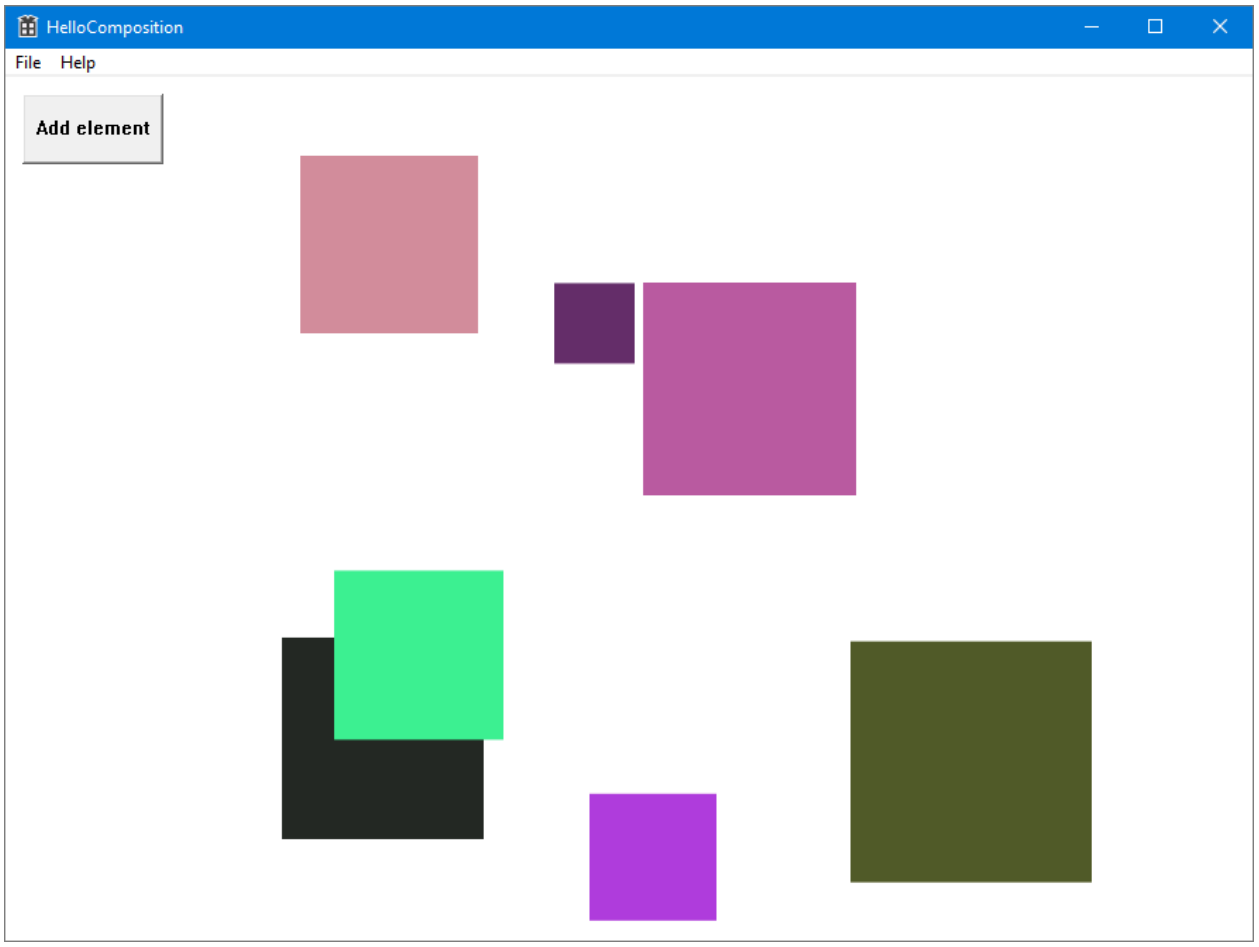
Prerequisites

The UWP hosting API has these prerequisites.

- We assume that you have some familiarity with app development using Win32 and UWP. For more info, see:
  - Get Started with Win32 and C++
  - Get started with Windows apps
  - Enhance your desktop application for Windows
- Windows 10 version 1803 or later
- Windows 10 SDK 17134 or later

How to use Composition APIs from a Win32 desktop application

In this tutorial, you create a simple Win32 C++ app and add UWP Composition elements to it. The focus is on correctly configuring the project, creating the interop code, and drawing something simple using Windows Composition APIs. The finished app looks like this.



Create a C++ Win32 project in Visual Studio

The first step is to create the Win32 app project in Visual Studio.

To create a new Win32 Application project in C++ named *HelloComposition*:

1. Open Visual Studio and select **File** > **New** > **Project**.

   The **New Project** dialog opens.

2. Under the **Installed** category, expand the **Visual C++** node, and then select **Windows Desktop**.

3. Select the **Windows Desktop Application** template.

4. Enter the name *HelloComposition*, then click **OK**.

   Visual Studio creates the project and opens the editor for the main app file.

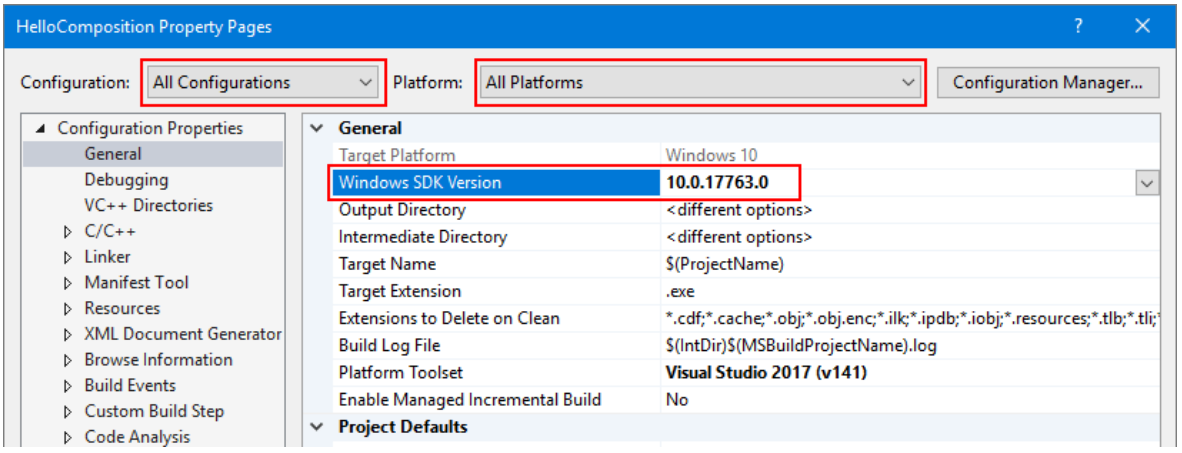
Configure the project to use Windows Runtime APIs

To use Windows Runtime (WinRT) APIs in your Win32 app, we use C++/WinRT. You need to configure your Visual Studio project to add C++/WinRT support.

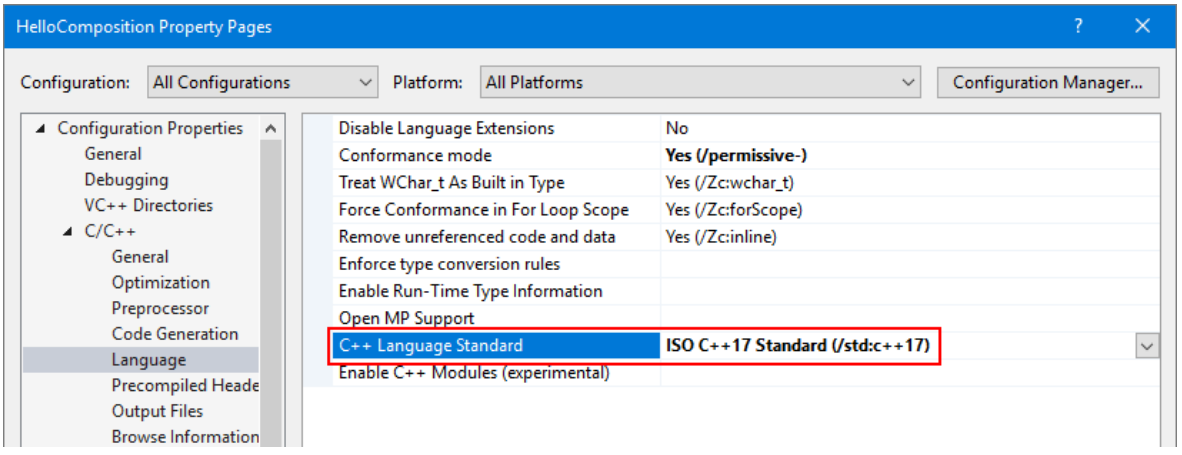
(For details, see Get started with C++/WinRT - Modify a Windows Desktop application project to add C++/WinRT support).

1. From the **Project** menu, open the project properties (*HelloComposition Properties*) and ensure the following settings are set to the specified values:

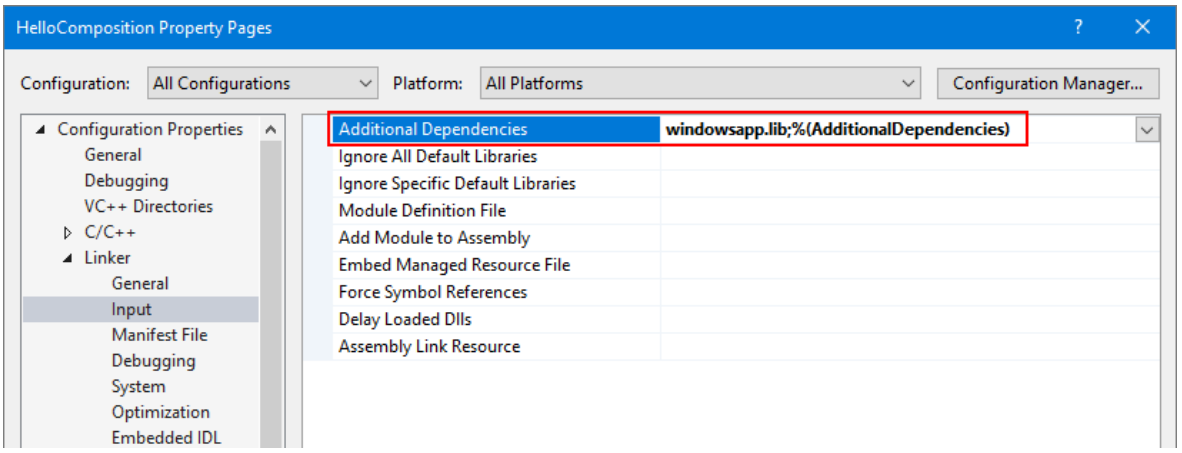
    - For **Configuration**, select *All Configurations*. For **Platform**, select *All Platforms*.
    - **Configuration Properties** > **General** > **Windows SDK Version** = *10.0.17763.0* or greater



    - **C/C++** > **Language** > **C++ Language Standard** = *ISO C++ 17 Standard (/stf:c++17)*



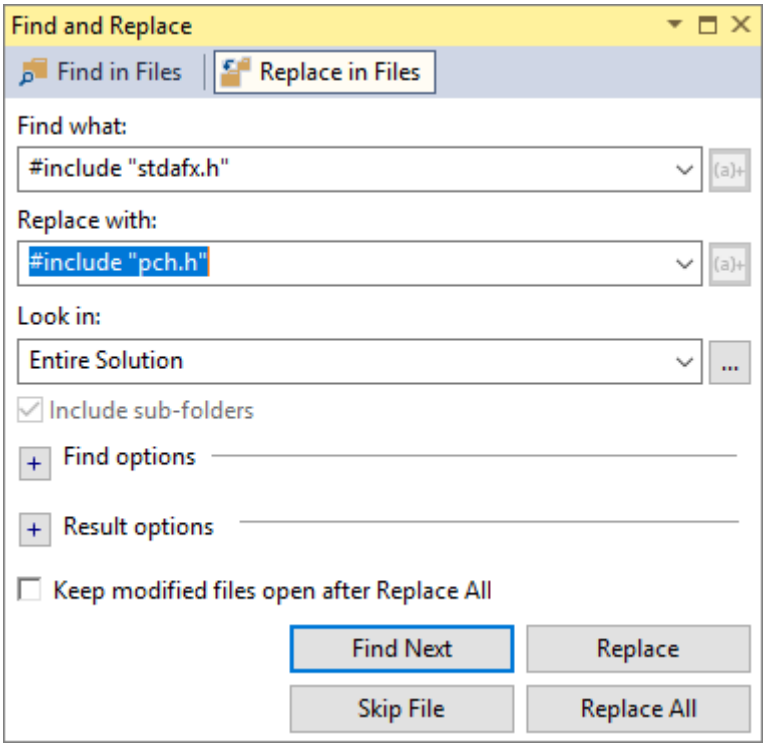
    - **Linker** > **Input** > **Additional Dependencies** must include "*windowsapp.lib*". If it's not included in the list, add it.



2. Update the precompiled header

- Rename `stdafx.h` and `stdafx.cpp` to `pch.h` and `pch.cpp`, respectively.
- Set project property **C/C++** > **Precompiled Headers** > **Precompiled Header File** to *pch.h*.
- Find and replace `#include "stdafx.h"` with `#include "pch.h"` in all files.

  (**Edit** > **Find and Replace** > **Find in Files**)



- In `pch.h`, include `winrt/base.h` and `unknwn.h`.

C++/WinRT

```
// reference additional headers your program requires here
#include <unknwn.h>
#include <winrt/base.h>
```

It's a good idea to build the project at this point to make sure there are no errors before going on.

Create a class to host composition elements

To host content you create with the visual layer, create a class (*CompositionHost*) to manage interop and create composition elements. This is where you do most of the configuration for hosting Composition APIs, including:

- getting a Compositor, which creates and manages objects in the Windows.UI.Composition namespace.
- creating a DispatcherQueueController/DispatcherQueue to manage tasks for the WinRT APIs.
- creating a DesktopWindowTarget and Composition container to display the composition objects.

We make this class a singleton to avoid threading issues. For example, you can only create one dispatcher queue per thread, so instantiating a second instance of CompositionHost on the same thread would cause an error.

> **Tip**
>
> If you need to, check the complete code at the end of the tutorial to make sure all the code is in the right places as you work through the tutorial.

1. Add a new class file to your project.

    - In **Solution Explorer**, right click the *HelloComposition* project.
    - In the context menu, select **Add** > **Class...**.
    - In the **Add Class** dialog, name the class *CompositionHost.cs*, then click **Add**.

2. Include headers and *usings* required for composition interop.

    - In CompositionHost.h, add these *includes* at the top of the file.

    C++/WinRT

    ```
    #pragma once
    #include <winrt/Windows.UI.Composition.Desktop.h>
    #include <windows.ui.composition.interop.h>
    #include <DispatcherQueue.h>
    ```

- In CompositionHost.cpp, add these *usings* at the top of the file, after any *includes*.

C++/WinRT

```
using namespace winrt;
using namespace Windows::System;
using namespace Windows::UI;
using namespace Windows::UI::Composition;
using namespace Windows::UI::Composition::Desktop;
using namespace Windows::Foundation::Numerics;
```

3. Edit the class to use the singleton pattern.

- In CompositionHost.h, make the constructor private.
- Declare a public static *GetInstance* method.

C++/WinRT

```
class CompositionHost
{
public:
    ~CompositionHost();
    static CompositionHost* GetInstance();

private:
    CompositionHost();
};
```

- In CompositionHost.cpp, add the definition of the *GetInstance* method.

C++/WinRT

```
CompositionHost* CompositionHost::GetInstance()
{
    static CompositionHost instance;
    return &instance;
}
```

4. In CompositionHost.h, declare private member variables for the Compositor, DispatcherQueueController, and DesktopWindowTarget.

C++/WinRT

```
winrt::Windows::UI::Composition::Compositor m_compositor{ nullptr };
winrt::Windows::System::DispatcherQueueController
m_dispatcherQueueController{ nullptr };
```

```
winrt::Windows::UI::Composition::Desktop::DesktopWindowTarget m_target{
nullptr };
```

5. Add a public method to initialize the composition interop objects.

> **Note**
>
> In *Initialize*, you call the *EnsureDispatcherQueue*, *CreateDesktopWindowTarget*, and *CreateCompositionRoot* methods. You create these methods in the next steps.

- In CompositionHost.h, declare a public method named *Initialize* that takes an HWND as an argument.

C++/WinRT

```
void Initialize(HWND hwnd);
```

- In CompositionHost.cpp, add the definition of the *Initialize* method.

C++/WinRT

```
void CompositionHost::Initialize(HWND hwnd)
{
    EnsureDispatcherQueue();
    if (m_dispatcherQueueController) m_compositor = Compositor();

    CreateDesktopWindowTarget(hwnd);
    CreateCompositionRoot();
}
```

6. Create a dispatcher queue on the thread that will be using Windows Composition.

   A Compositor must be created on a thread that has a dispatcher queue, so this method is called first during initialization.

- In CompositionHost.h, declare a private method named *EnsureDispatcherQueue*.

C++/WinRT

```
void EnsureDispatcherQueue();
```

- In CompositionHost.cpp, add the definition of the *EnsureDispatcherQueue* method.

C++/WinRT

```
void CompositionHost::EnsureDispatcherQueue()
{
    namespace abi = ABI::Windows::System;

    if (m_dispatcherQueueController == nullptr)
    {
        DispatcherQueueOptions options
        {
            sizeof(DispatcherQueueOptions), /* dwSize */
            DQTYPE_THREAD_CURRENT,          /* threadType */
            DQTAT_COM_ASTA                  /* apartmentType */
        };

        Windows::System::DispatcherQueueController controller{ nullptr
};
        check_hresult(CreateDispatcherQueueController(options,
reinterpret_cast<abi::IDispatcherQueueController**>
(put_abi(controller))));
        m_dispatcherQueueController = controller;
    }
}
```

7. Register your app's window as a composition target.

- In CompositionHost.h, declare a private method named *CreateDesktopWindowTarget* that takes an HWND as an argument.

C++/WinRT

```
void CreateDesktopWindowTarget(HWND window);
```

- In CompositionHost.cpp, add the definition of the *CreateDesktopWindowTarget* method.

C++/WinRT

```
void CompositionHost::CreateDesktopWindowTarget(HWND window)
{
    namespace abi = ABI::Windows::UI::Composition::Desktop;

    auto interop = m_compositor.as<abi::ICompositorDesktopInterop>();
    DesktopWindowTarget target{ nullptr };
    check_hresult(interop->CreateDesktopWindowTarget(window, false,
reinterpret_cast<abi::IDesktopWindowTarget**>(put_abi(target))));
```

```
    m_target = target;
}
```

8. Create a root visual container to hold visual objects.

    - In CompositionHost.h, declare a private method named *CreateCompositionRoot*.

C++/WinRT

```
void CreateCompositionRoot();
```

    - In CompositionHost.cpp, add the definition of the *CreateCompositionRoot* method.

C++/WinRT

```
void CompositionHost::CreateCompositionRoot()
{
    auto root = m_compositor.CreateContainerVisual();
    root.RelativeSizeAdjustment({ 1.0f, 1.0f });
    root.Offset({ 124, 12, 0 });
    m_target.Root(root);
}
```

Build the project now to make sure there are no errors.

These methods set up the components needed for interop between the UWP visual layer and Win32 APIs. Now you can add content to your app.

Add composition elements

With the infrastructure in place, you can now generate the Composition content you want to show.

For this example, you add code that creates a randomly-colored square SpriteVisual with an animation that causes it to drop after a short delay.

1. Add a composition element.

    - In CompositionHost.h, declare a public method named *AddElement* that takes 3 **float** values as arguments.

C++/WinRT

```
void AddElement(float size, float x, float y);
```

- In CompositionHost.cpp, add the definition of the *AddElement* method.

C++/WinRT

```
void CompositionHost::AddElement(float size, float x, float y)
{
    if (m_target.Root())
    {
        auto visuals = m_target.Root().as<ContainerVisual>
().Children();
        auto visual = m_compositor.CreateSpriteVisual();

        auto element = m_compositor.CreateSpriteVisual();
        uint8_t r = (double)(double)(rand() % 255);;
        uint8_t g = (double)(double)(rand() % 255);;
        uint8_t b = (double)(double)(rand() % 255);;

        element.Brush(m_compositor.CreateColorBrush({ 255, r, g, b }));
        element.Size({ size, size });
        element.Offset({ x, y, 0.0f, });

        auto animation = m_compositor.CreateVector3KeyFrameAnimation();
        auto bottom = (float)600 - element.Size().y;
        animation.InsertKeyFrame(1, { element.Offset().x, bottom, 0 });

        using timeSpan = std::chrono::duration<int, std::ratio<1, 1>>;

        std::chrono::seconds duration(2);
        std::chrono::seconds delay(3);

        animation.Duration(timeSpan(duration));
        animation.DelayTime(timeSpan(delay));
        element.StartAnimation(L"Offset", animation);
        visuals.InsertAtTop(element);

        visuals.InsertAtTop(visual);
    }
}
```

Create and show the window

Now, you can add a button and the UWP composition content to your Win32 UI.

1. In HelloComposition.cpp, at the top of the file, include *CompositionHost.h*, define BTN_ADD, and get an instance of CompositionHost.

C++/WinRT

```
#include "CompositionHost.h"

// #define MAX_LOADSTRING 100 // This is already in the file.
#define BTN_ADD 1000

CompositionHost* compHost = CompositionHost::GetInstance();
```

2. In the `InitInstance` method, change the size of the window that's created. (In this line, change `CW_USEDEFAULT, 0` to `900, 672`.)

C++/WinRT

```
HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, 0, 900, 672, nullptr, nullptr, hInstance, nullptr);
```

3. In the WndProc function, add `case WM_CREATE` to the *message* switch block. In this case, you initialize the CompositionHost and create the button.

C++/WinRT

```
case WM_CREATE:
{
    compHost->Initialize(hWnd);
    srand(time(nullptr));

    CreateWindow(TEXT("button"), TEXT("Add element"),
        WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON,
        12, 12, 100, 50,
        hWnd, (HMENU)BTN_ADD, nullptr, nullptr);
}
break;
```

4. Also in the WndProc function, handle the button click to add a composition element to the UI.

   Add `case BTN_ADD` to the *wmId* switch block inside the WM_COMMAND block.

C++/WinRT

```
case BTN_ADD: // addButton click
{
    double size = (double)(rand() % 150 + 50);
    double x = (double)(rand() % 600);
    double y = (double)(rand() % 200);
    compHost->AddElement(size, x, y);
```

```
        break;
    }
```

You can now build and run your app. If you need to, check the complete code at the end of the tutorial to make sure all the code is in the right places.

When you run the app and click the button, you should see animated squares added to the UI.

Additional resources

- Win32 HelloComposition sample (GitHub)
- Get Started with Win32 and C++
- Get started with Windows apps (UWP)
- Enhance your desktop application for Windows (UWP)
- Windows.UI.Composition namespace (UWP)

Complete code

Here's the complete code for the CompositionHost class and the InitInstance method.

CompositionHost.h

C++/WinRT

```
#pragma once
#include <winrt/Windows.UI.Composition.Desktop.h>
#include <windows.ui.composition.interop.h>
#include <DispatcherQueue.h>

class CompositionHost
{
public:
    ~CompositionHost();
    static CompositionHost* GetInstance();

    void Initialize(HWND hwnd);
    void AddElement(float size, float x, float y);

private:
    CompositionHost();

    void CreateDesktopWindowTarget(HWND window);
    void EnsureDispatcherQueue();
    void CreateCompositionRoot();
```

```
    winrt::Windows::UI::Composition::Compositor m_compositor{ nullptr };
    winrt::Windows::UI::Composition::Desktop::DesktopWindowTarget m_target{
nullptr };
    winrt::Windows::System::DispatcherQueueController
m_dispatcherQueueController{ nullptr };
};
```

CompositionHost.cpp

C++/WinRT

```
#include "pch.h"
#include "CompositionHost.h"

using namespace winrt;
using namespace Windows::System;
using namespace Windows::UI;
using namespace Windows::UI::Composition;
using namespace Windows::UI::Composition::Desktop;
using namespace Windows::Foundation::Numerics;

CompositionHost::CompositionHost()
{
}

CompositionHost* CompositionHost::GetInstance()
{
    static CompositionHost instance;
    return &instance;
}

CompositionHost::~CompositionHost()
{
}

void CompositionHost::Initialize(HWND hwnd)
{
    EnsureDispatcherQueue();
    if (m_dispatcherQueueController) m_compositor = Compositor();

    if (m_compositor)
    {
        CreateDesktopWindowTarget(hwnd);
        CreateCompositionRoot();
    }
}

void CompositionHost::EnsureDispatcherQueue()
{
    namespace abi = ABI::Windows::System;

    if (m_dispatcherQueueController == nullptr)
```

```
    {
        DispatcherQueueOptions options
        {
            sizeof(DispatcherQueueOptions), /* dwSize */
            DQTYPE_THREAD_CURRENT,          /* threadType */
            DQTAT_COM_ASTA                  /* apartmentType */
        };

        Windows::System::DispatcherQueueController controller{ nullptr };
        check_hresult(CreateDispatcherQueueController(options,
reinterpret_cast<abi::IDispatcherQueueController**>(put_abi(controller))));
        m_dispatcherQueueController = controller;
    }
}

void CompositionHost::CreateDesktopWindowTarget(HWND window)
{
    namespace abi = ABI::Windows::UI::Composition::Desktop;

    auto interop = m_compositor.as<abi::ICompositorDesktopInterop>();
    DesktopWindowTarget target{ nullptr };
    check_hresult(interop->CreateDesktopWindowTarget(window, false,
reinterpret_cast<abi::IDesktopWindowTarget**>(put_abi(target))));
    m_target = target;
}

void CompositionHost::CreateCompositionRoot()
{
    auto root = m_compositor.CreateContainerVisual();
    root.RelativeSizeAdjustment({ 1.0f, 1.0f });
    root.Offset({ 124, 12, 0 });
    m_target.Root(root);
}

void CompositionHost::AddElement(float size, float x, float y)
{
    if (m_target.Root())
    {
        auto visuals = m_target.Root().as<ContainerVisual>().Children();
        auto visual = m_compositor.CreateSpriteVisual();

        auto element = m_compositor.CreateSpriteVisual();
        uint8_t r = (double)(double)(rand() % 255);;
        uint8_t g = (double)(double)(rand() % 255);;
        uint8_t b = (double)(double)(rand() % 255);;

        element.Brush(m_compositor.CreateColorBrush({ 255, r, g, b }));
        element.Size({ size, size });
        element.Offset({ x, y, 0.0f, });

        auto animation = m_compositor.CreateVector3KeyFrameAnimation();
        auto bottom = (float)600 - element.Size().y;
        animation.InsertKeyFrame(1, { element.Offset().x, bottom, 0 });

        using timeSpan = std::chrono::duration<int, std::ratio<1, 1>>;
```

```
        std::chrono::seconds duration(2);
        std::chrono::seconds delay(3);

        animation.Duration(timeSpan(duration));
        animation.DelayTime(timeSpan(delay));
        element.StartAnimation(L"Offset", animation);
        visuals.InsertAtTop(element);

        visuals.InsertAtTop(visual);
    }
}
```

HelloComposition.cpp (Partial)

C++/WinRT

```
#include "pch.h"
#include "HelloComposition.h"
#include "CompositionHost.h"

#define MAX_LOADSTRING 100
#define BTN_ADD 1000

CompositionHost* compHost = CompositionHost::GetInstance();

// Global Variables:

// ...
// ... code not shown ...
// ...

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // Store instance handle in our global variable

   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, 900, 672, nullptr, nullptr, hInstance, nullptr);

// ...
// ... code not shown ...
// ...
}

// ...

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
    switch (message)
    {
// Add this...
```

```
    case WM_CREATE:
    {
        compHost->Initialize(hWnd);
        srand(time(nullptr));

        CreateWindow(TEXT("button"), TEXT("Add element"),
            WS_VISIBLE | WS_CHILD | BS_PUSHBUTTON,
            12, 12, 100, 50,
            hWnd, (HMENU)BTN_ADD, nullptr, nullptr);
    }
    break;
// ...
    case WM_COMMAND:
    {
        int wmId = LOWORD(wParam);
        // Parse the menu selections:
        switch (wmId)
        {
        case IDM_ABOUT:
            DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
            break;
        case IDM_EXIT:
            DestroyWindow(hWnd);
            break;
// Add this...
        case BTN_ADD: // addButton click
        {
            double size = (double)(rand() % 150 + 50);
            double x = (double)(rand() % 600);
            double y = (double)(rand() % 200);
            compHost->AddElement(size, x, y);
            break;
        }
// ...
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
    }
    break;
    case WM_PAINT:
    {
        PAINTSTRUCT ps;
        HDC hdc = BeginPaint(hWnd, &ps);
        // TODO: Add any drawing code that uses hdc here...
        EndPaint(hWnd, &ps);
    }
    break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

```
// ...
// ... code not shown ...
// ...
```

C#/WinRT

Article • 03/16/2023

C#/WinRT is a NuGet-packaged toolkit that provides Windows Runtime (WinRT) projection support for the C# language. A projection assembly is an interop assembly, which enables programming WinRT APIs in a natural and familiar way for the target language. The C#/WinRT projection hides the details of interop between C# and WinRT interfaces, and provides mappings of many WinRT types to appropriate .NET equivalents, such as strings, URIs, common value types, and generic collections.

C#/WinRT currently provides support for consuming WinRT APIs through the use of Target Framework Monikers (TFMs) in .NET. Setting the TFM with a specific Windows SDK version adds references to the Windows SDK projection and runtime assemblies generated by C#/WinRT.

The C#/WinRT NuGet package enables you to create and reference your own WinRT interop assemblies for .NET consumers. The latest C#/WinRT version also includes a preview of authoring WinRT types in C#.

For additional information, see the C#/WinRT GitHub repo.

Motivation for C#/WinRT

.NET (previously known as .NET Core) is an open-source, cross-platform runtime that can be used to build device, cloud, and IoT applications.

Previous versions of .NET Framework and .NET Core had built-in knowledge of WinRT—a Windows-specific technology. To support the portability and efficiency goals of .NET 6+, we lifted the WinRT projection support out of the .NET compiler and runtime and moved it into the C#/WinRT toolkit (see Built-in support for WinRT is removed from .NET). The goal of C#/WinRT is to provide parity with the built-in WinRT support provided by earlier versions of the C# compiler and .NET runtime. For details, see .NET mappings of Windows Runtime types.

C#/WinRT also supports components in the Windows App SDK, including WinUI 3. The Windows App SDK lifts native Microsoft UI controls and other native components out of the operating system. This enables app developers to use the latest controls and components on Windows 10, version 1809, and later releases.

Finally, C#/WinRT is a general toolkit and is intended to support other scenarios where built-in support for WinRT is not available in the C# compiler or .NET runtime.

What's new

The latest C#/WinRT releases can be found on our release notes page in the Github repo.

Usage

The C#/WinRT NuGet package can be used to both generate C# projections (also called interop assemblies) from WinRT components and in Authoring C#/WinRT components. For more details regarding the usage scenarios for C#/WinRT, refer to the usage guide on our repo.

Generate and distribute an interop assembly

WinRT APIs are defined in Windows Metadata (WinMD) files. The C#/WinRT NuGet package (Microsoft.Windows.CsWinRT) includes the C#/WinRT compiler, **cswinrt.exe**, which you can use to process WinMD files and generate .NET C# code. C#/WinRT compiles these source files into an interop assembly, similar to how C++/WinRT generates headers for the C++ language projection. You can then distribute the C#/WinRT interop assembly along with the implementation assembly for .NET applications to reference, typically as a NuGet package.

For more details on how to generate and distribute an interop assembly, see Generate a C# projection from a C++/WinRT component, distribute as a NuGet for .NET apps.

Reference an interop assembly

Typically, C#/WinRT interop assemblies are referenced by application projects. But they may also be referenced in turn by intermediate interop assemblies. For example, the WinUI interop assembly would reference the Windows SDK interop assembly.

If you distribute a third-party WinRT component without an official interop assembly, an application project may follow the procedure for generating an interop assembly to generate its own private projection sources. We don't recommend this approach, because it can produce conflicting projections of the same type within a process. NuGet packaging, following the Semantic Versioning scheme, is designed to prevent this. An official third-party interop assembly is preferred.

Embedded support for WinRT types (Preview)

Starting with C#/WinRT version 1.4.1, support is included for embedding Windows SDK projection and runtime sources for both .NET and .NET Standard 2.0 into your library or app's output. This is useful in cases where usage of Windows SDK types is self-contained. Embedded support removes dependencies on WinRT.Runtime.dll and Microsoft.Windows.SDK.NET.dll which reduces the library or app output size. It also allows library developers to provide downlevel support and removes the need for multi-targeting.

For more details, see the C#/WinRT embedded documentation on our repo.

WinRT type activation

C#/WinRT supports activation of WinRT types hosted by the operating system, as well as third-party components such as Win2D. Support for third-party component activation in a desktop application is enabled with registration free WinRT activation (see Enhancing Non-packaged Desktop Apps using Windows Runtime Components), available in Windows 10, version 1903 and later. Native C++ components should set the **Windows Desktop Compatible** property to *True* either via the project properties or the `.vcxproj` file, in order to reference and forward the **Microsoft.VCLibs.Desktop** binaries to consuming apps. Otherwise, the VCRT Forwarders package will be required by consuming apps if the component targets UWP apps only.

C#/WinRT also provides an activation fallback path if Windows fails to activate the type as described above. In this case, C#/WinRT attempts to locate a native implementation DLL based on the fully qualified type name, progressively removing elements. For example, the fallback logic would attempt to activate the Contoso.Controls.Widget type from the following modules, in sequence:

1. Contoso.Controls.Widget.dll
2. Contoso.Controls.dll
3. Contoso.dll

C#/WinRT uses the LoadLibrary alternate search order to locate an implementation DLL. An app relying on this fallback behavior should package the implementation DLL alongside the app module.

Common errors and troubleshooting

- Error: "Windows Metadata not provided or detected."

  You can specify Windows Metadata by using the `<CsWinRTWindowsMetadata>` project property, for example:

```xml
<CsWinRTWindowsMetadata>10.0.19041.0</CsWinRTWindowsMetadata>
```

In C#/WinRT version 1.2.1 and later, this property defaults to `TargetPlatformVersion`, which is derived from the Windows SDK version specified in the `TargetFramework` property.

- Error CS0246: The type or namespace name 'Windows' could not be found (are you missing a using directive or an assembly reference?)

  To address this error, edit your `<TargetFramework>` property to target a specific Windows version, for example:

  ```xml
  <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
  ```

  Refer to the docs on Calling Windows Runtime APIs for more details on specifying the `<TargetFramework>` property.

- **System.InvalidCastException** when casting to an interface that has the `ComImport` attribute

  When casting an object to an interface that has the `ComImport` attribute, you'll need to use the `.As<>` operator instead of using an explicit cast expression. For example:

  ```C#
  someObject.As<SomeComImportInterface>
  ```

  For more details, see the COM interop guide.

- System.Runtime.InteropServices.COMException: Class not registered (0x80040154 (REGDB_E_CLASSNOTREG))
  - If you see this exception when consuming a C#/WinRT projection from a C++/WinRT component, make sure the component has set the **Windows Desktop Compatible** property to *True* either via the project properties or via the `.vcxproj` file.

.NET SDK versioning errors

You may encounter the following errors or warnings in a project that is built with an earlier .NET SDK version than any of its dependencies.

Expand table

Error or warning message	Reason
Warning MSB3277: Found conflicts between different versions of WinRT.Runtime or Microsoft.Windows.SDK.NET that could not be resolved.	This build warning occurs when referencing a library that exposes Windows SDK types on its API surface.
Error CS1705: Assembly 'AssemblyName1' uses 'TypeName' which has a higher version than referenced assembly 'AssemblyName2'	This build compiler error occurs when referencing and consuming exposed Windows SDK types in a library.
System.IO.FileLoadException	This runtime error may occur when calling certain APIs in a library that does not expose Windows SDK types.

To fix these errors, update your .NET SDK to the latest version. Doing so will ensure that the runtime and Windows SDK assembly versions used by your application are compatible with all dependencies. These errors may occur with early servicing/feature updates to the .NET SDK, because runtime fixes may require updates to our assembly versions.

Known issues

Known issues and breaking changes are noted in the C#/WinRT GitHub repo.

If you encounter any functional issues with the C#/WinRT NuGet package, the cswinrt.exe compiler, or the generated projection sources, submit issues to us via the C#/WinRT issues page.

Additional resources

- C#/WinRT GitHub repo

.NET mappings of WinRT types in C#/WinRT

Article • 10/24/2022

This article lists the mappings that C#/WinRT makes between Windows Runtime (WinRT) types and .NET types in desktop apps that target .NET 6 (or later). In these apps, Visual Studio IntelliSense shows the .NET type instead of the Windows Runtime type. For example, if a Windows Runtime method takes a parameter of type **IVector<string>**, then IntelliSense shows a parameter of type **IList<string>**. Similarly, in a WinRT component authored using C#/WinRT, you use the .NET type in member signatures. When you use C#/WinRT to generate a Windows Runtime component, the .NET type is translated into the corresponding WinRT type.

The C#/WinRT custom type mappings are categorized by types in the Windows SDK or in WinUI 3 (WinUI 3 is part of the Windows App SDK). The WinRT types for Windows SDK mappings live under the **Windows.*** namespaces, and the WinRT types for WinUI 3 mappings live under the **Microsoft.UI.Xaml.*** namespaces. There are two reasons for custom type mappings that C#/WinRT makes for WinRT types:

- **WinRT types that map to .NET types with a different name and/or namespace.** These custom mappings are for mapping WinRT types to existing .NET equivalent types. There are also cases where the mapping is to a different type (e.g., a value type maps to a class type).
- **WinRT types that map to .NET types with the same name and namespace.** These custom mappings are generally for performance or enhancement reasons, and are implemented directly in C#. Most of the types that have the same namespace name and type name in WinRT and .NET are structures (or types associated with structures, such as enumerations). In WinRT, structures have no members other than fields, and require helper types, which .NET hides. The .NET versions of these structures have properties and methods that provide the functionality of the hidden helper types (for example, **Windows.UI.Color**).

> ⓘ **Note**
>
> For a list of mappings between WinRT and .NET types in the context of UWP apps, see **.NET mappings of WinRT types in UWP**.

Mappings for WinRT types in the Windows SDK

Types with a different name and/or namespace

WinRT type/namespace	.NET type/namespace
DateTime (Windows.Foundation)	DateTimeOffset (System)
EventHandler<T> (Windows.Foundation)	EventHandler<T> (System)
EventRegistrationToken (Windows.Foundation)	EventRegistrationToken (WinRT)
HResult (Windows.Foundation)	Exception (System)
IClosable (Windows.Foundation)	IDisposable (System)
IReference<T> (Windows.Foundation)	Nullable<T> (System)
TimeSpan (Windows.Foundation)	TimeSpan (System)
Uri (Windows.Foundation)	Uri (System)
IIterable<T> (Windows.Foundation.Collections)	IEnumerable<T> (System.Collections.Generic)
IIterator<T> (Windows.Foundation.Collections)	IEnumerator<T> (System.Collections.Generic)
IMap<K,V> (Windows.Foundation.Collections)	IDictionary<TKey,TValue> (System.Collections.Generic)
IMapView<K,V> (Windows.Foundation.Collections)	IReadOnlyDictionary<TKey,TValue> (System.Collections.Generic)
IKeyValuePair<K,V> (Windows.Foundation.Collections)	KeyValuePair<TKey,TValue> (System.Collections.Generic)
IVector<T> (Windows.Foundation.Collections)	IList<T> (System.Collections.Generic)
IVectorView<T> (Windows.Foundation.Collections)	IReadOnlyList<T> (System.Collections.Generic)
AttributeTargets (Windows.Foundation.Metadata)	AttributeTargets (System)
AttributeUsageAttribute (Windows.Foundation.Metadata)	AttributeUsageAttribute (System)
Matrix3x2 (Windows.Foundation.Numerics)	Matrix3x2 (System.Numerics)
Matrix4x4 (Windows.Foundation.Numerics)	Matrix4x4 (System.Numerics)

WinRT type/namespace	.NET type/namespace
Plane (Windows.Foundation.Numerics)	Plane (System.Numerics)
Quaternion (Windows.Foundation.Numerics)	Quaternion (System.Numerics)
Vector2 (Windows.Foundation.Numerics)	Vector2 (System.Numerics)
Vector3 (Windows.Foundation.Numerics)	Vector3 (System.Numerics)
Vector4 (Windows.Foundation.Numerics)	Vector4 (System.Numerics)
IBindableIterable (Windows.UI.Xaml.Interop)	IEnumerable (System.Collections)
IBindableVector (Windows.UI.Xaml.Interop)	IList (System.Collections)
TypeName (Windows.UI.Xaml.Interop)	Type (System)

Types with the same name and namespace

Type	Namespace
IPropertyValue	Windows.Foundation
IReferenceArray<T>	Windows.Foundation
Point	Windows.Foundation
Rect	Windows.Foundation
Size	Windows.Foundation
Color	Windows.UI
TypeKind	Windows.UI.Xaml.Interop

Mappings for WinRT types in WinUI

Types with a different name and/or namespace

WinRT type/namespace	.NET type/namespace
INotifyCollectionChanged (Microsoft.UI.Xaml.Data)	INotifyCollectionChanged (System.Collections.Specialized)
NotifyCollectionChangedEventHandler (Microsoft.UI.Xaml.Data)	NotifyCollectionChangedEventHandler (System.Collections.Specialized)

WinRT type/namespace	.NET type/namespace
NotifyCollectionChangedEventArgs (Microsoft.UI.Xaml.Data)	NotifyCollectionChangedEventArgs (System.Collections.Specialized)
NotifyCollectionChangedAction (Microsoft.UI.Xaml.Data)	NotifyCollectionChangedAction (System.Collections.Specialized)
DataErrorsChangedEventArgs (Microsoft.UI.Xaml.Data)	DataErrorsChangedEventArgs (System.ComponentModel)
INotifyDataErrorInfo (Microsoft.UI.Xaml.Data)	INotifyDataErrorInfo (System.ComponentModel)
INotifyPropertyChanged (Microsoft.UI.Xaml.Data)	INotifyPropertyChanged (System.ComponentModel)
PropertyChangedEventHandler (Microsoft.UI.Xaml.Data)	PropertyChangedEventHandler (System.ComponentModel)
PropertyChangedEventArgs (Microsoft.UI.Xaml.Data)	PropertyChangedEventArgs (System.ComponentModel)
ICommand (Microsoft.UI.Xaml.Input)	ICommand (System.Windows.Input)
IXamlServiceProvider (Microsoft.UI.Xaml)	IServiceProvider (System)

Types with the same name and namespace

Type	Namespace
CornerRadius	Microsoft.UI.Xaml
Duration	Microsoft.UI.Xaml
DurationType	Microsoft.UI.Xaml
GridLength	Microsoft.UI.Xaml
GridUnitType	Microsoft.UI.Xaml
Thickness	Microsoft.UI.Xaml
GeneratorPosition	Microsoft.UI.Xaml.Controls.Primitives
Matrix	Microsoft.UI.Xaml.Media
KeyTime	Microsoft.UI.Xaml.Media.Animation
RepeatBehavior	Microsoft.UI.Xaml.Media.Animation
RepeatBehaviorType	Microsoft.UI.Xaml.Media.Animation

Type	Namespace
Matrix3D	(Microsoft.UI.Xaml.Media.Media3D

Agile objects in C#/WinRT

Article • 01/15/2022

Most Windows Runtime classes are *agile*, meaning they can be accessed from any threads across different apartments. C#/WinRT types that you author are agile by default, and it is not possible to opt out of this behavior for those types.

However, projected C#/WinRT types (which includes Windows Runtime types that are provided by the Windows SDK and the WinUI library) may or may not be agile. For example, many types that represent UI objects are not agile. When you consume non-agile types, you need to take into consideration their threading model and marshaling behavior. C#/WinRT provides support for agile references if you need to marshal a non-agile object across apartments in a safe way.

> **Note**
>
> The Windows Runtime is based on COM. In COM terms, an agile class is registered with `ThreadingModel` = *Both*. For more info about COM threading models, and apartments, see **Understanding and Using COM Threading Models**.

Check for agile support

To check whether a Windows Runtime object is agile, use the following code to determine whether the object supports the IAgileObject interface.

C#

```
var queryAgileObject = testObject.As<IAgileObject>();

if (queryAgileObject != null) {
    // testObject is agile.
}
```

Create an agile reference

To create an agile reference for a non-agile object, you can use the `AsAgile` extension method. `AsAgile` is a generic extension method that can be applied to any projected C#/WinRT type. If the type is not a projected type, an exception is thrown. Here is an example using a PopupMenu object, which is a non-agile type from the Windows SDK.

C#

```
var nonAgileObj = new Windows.UI.Popups.PopupMenu();
AgileReference<Windows.UI.Popups.PopupMenu> agileReference =
nonAgileObj.AsAgile();
```

You can now pass `agileReference` to a thread in a different apartment, and use it there.

C#

```
await Task.Run(() => {
        Windows.UI.Popups.PopupMenu nonAgileObjAgain = agileReference.Get()
    });
```

Generate a C# projection from a C++/WinRT component, distribute as a NuGet for .NET apps

Article • 03/23/2023

In this topic, we walk through using C#/WinRT to generate a C# .NET projection (or interop) assembly from a C++/WinRT Windows Runtime component, and distribute it as a NuGet package for .NET applications.

In .NET 6 and later, consumption of Windows metadata (WinMD) files is no longer supported (see Built-in support for WinRT is removed from .NET). Instead, the C#/WinRT tool can be used to generate a projection assembly for any WinMD file, which then enables consumption of WinRT components from .NET applications. A projection assembly is also known as an interop assembly. This walkthrough shows you how to do the following:

- Use the C#/WinRT package to generate a C# projection from a C++/WinRT component.
- Distribute the component, along with the projection assembly, as a NuGet package.
- Consume the NuGet package from a .NET console application.

Prerequisites

This walkthrough and the corresponding sample require the following tools and components:

- Visual Studio 2022 (or Visual Studio 2019) with the Universal Windows Platform development workload installed. In **Installation Details** > **Universal Windows Platform development**, check the **C++ (v14x) Universal Windows Platform tools** option.
- .NET 6.0 SDK or later.

Visual Studio 2019 only. The C++/WinRT VSIX extension, which gives you C++/WinRT project templates in Visual Studio. The project templates are built in to Visual Studio 2022.

We'll be using Visual Studio 2022 and .NET 6 in this walkthrough.

> **ⓘ Important**
>
> Also, you'll need to download or clone the sample code for this topic from the C#/WinRT projection sample on GitHub. Visit CsWinRT, and click the green **Code** button to get the `git clone` url. Be sure to read the README.md file for the sample.

Create a simple C++/WinRT Windows Runtime component

To follow this walkthrough, you must first have a C++/WinRT Windows Runtime component (WRC) from which to generate the C# projection assembly.

This walkthrough uses the **SimpleMathComponent** WRC from the C#/WinRT projection sample on GitHub, which you already downloaded or cloned. **SimpleMathComponent** was created from the **Windows Runtime Component (C++/WinRT)** Visual Studio project template (which comes with Visual Studio 2022, or with the C++/WinRT VSIX extension).

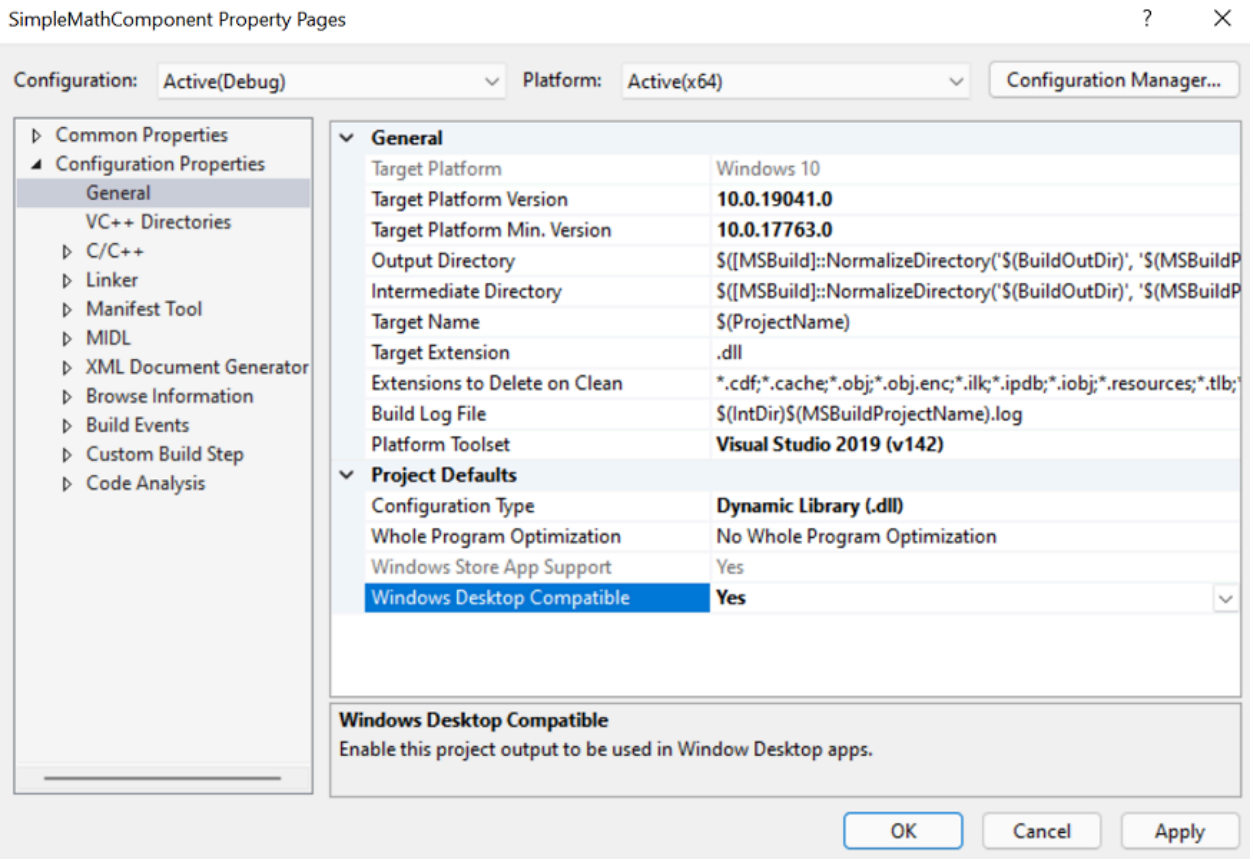
To open the **SimpleMathComponent** project in Visual Studio, open the `\CsWinRT\src\Samples\NetProjectionSample\CppWinRTComponentProjectionSample.sln` file, which you'll find in your download or clone of the repo.

The code in this project provides the functionality for the basic math operations shown in the header file below.

C++

```
// SimpleMath.h
...
namespace winrt::SimpleMathComponent::implementation
{
    struct SimpleMath: SimpleMathT<SimpleMath>
    {
        SimpleMath() = default;
        double add(double firstNumber, double secondNumber);
        double subtract(double firstNumber, double secondNumber);
        double multiply(double firstNumber, double secondNumber);
        double divide(double firstNumber, double secondNumber);
    };
}
```

You can confirm that the **Windows Desktop Compatible** property is set to *Yes* for the **SimpleMathComponent** C++/WinRT Windows Runtime component project. To do that, in the project properties for **SimpleMathComponent**, under **Configuration Properties** > **General** > **Project Defaults**, set the property **Windows Desktop Compatible** to *Yes*. That ensures that the correct runtime binaries are loaded for consuming .NET desktop apps.



For more detailed steps about creating a C++/WinRT component and generating a WinMD file, see Windows Runtime components with C++/WinRT.

> **Note**
>
> If you're implementing IInspectable::GetRuntimeClassName in your component, then it **must** return a valid WinRT class name. Because C#/WinRT uses the class name string for interop, an incorrect runtime class name will raise an **InvalidCastException**.

Add a projection project to the component solution

First, with the **CppWinRTComponentProjectionSample** solution still open in Visual Studio, remove the **SimpleMathProjection** project from that solution. Then delete from

your file system the **SimpleMathProjection** folder (or rename it if you prefer). Those steps are necessary so that you can follow this walkthrough step by step.

1. Add a new C# library project to your solution.
   a. In **Solution Explorer**, right-click your solution node and click **Add** > **New Project**.
   b. In the **Add a new project** dialog box, type **Class Library** in the search box. Choose **C#** from the language list, and then choose **Windows** from the platform list. Choose the C# project template that's called simply **Class Library** (with no prefixes nor suffixes), and click **Next**.
   c. Name the new project *SimpleMathProjection*. The location should already be set to the same `\CsWinRT\src\Samples\NetProjectionSample` folder that the **SimpleMathComponent** folder is in; but confirm that. Then click **Next**.
   d. On the **Additional information** page, select **.NET 6.0 (Long-term support)**, and then choose **Create**.

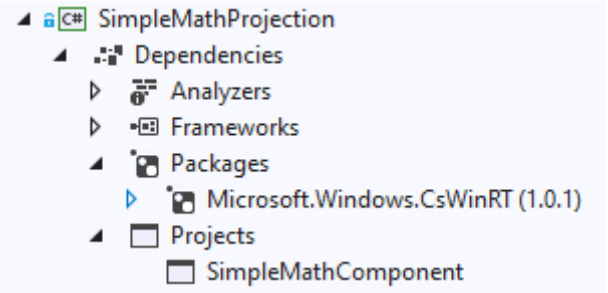
2. Delete the stub **Class1.cs** file from the project.

3. Use the steps below to install the C#/WinRT NuGet package.
   a. In **Solution Explorer**, right-click your **SimpleMathProjection** project and select **Manage NuGet Packages**.
   b. In the **Browse** tab, type or paste **Microsoft.Windows.CsWinRT** into the search box, in search results select the item with the latest version, and then click **Install** to install the package into the **SimpleMathProjection** project.

4. Add to **SimpleMathProjection** a project reference to the **SimpleMathComponent** project. In **Solution Explorer**, right-click the **Dependencies** node under the **SimpleMathProjection** project node, select **Add Project Reference**, and select the **SimpleMathComponent** project > **OK**.

Don't try to build the project yet. We'll be doing that in a later step.

So far, your **Solution Explorer** should look similar to this (your version numbers will be different).



Build projects out of source

For the **CppWinRTComponentProjectionSample** solution in the C#/WinRT projection sample (which you downloaded or cloned from GitHub, and now have open), the build output location is configured with the **Directory.Build.props** file to build *out of source*. That means that files from the build output are generated outside of the source folder. We recommend that you build out of source when you use the C#/WinRT tool. That prevents the C# compiler from inadvertently picking up all **.cs* files under the project root directory, which can cause duplicate type errors (for example when compiling for multiple configurations and/or platforms).

Even though this is already configured for the **CppWinRTComponentProjectionSample** solution, follow the steps below to get practice in doing the configuration for yourself.

To configure your solution to build out of source:

1. With the **CppWinRTComponentProjectionSample** solution still open, right-click on the solution node, and select **Add** > **New Item**. Select the **XML File** item, and name it **Directory.Build.props** (without a `.xml` extension). Click **Yes** to overwrite the existing file.

2. Replace the contents of **Directory.Build.props** with the configuration below.

   XML

   ```
   <Project>
     <PropertyGroup>
       <BuildOutDir>$([MSBuild]::NormalizeDirectory('$(SolutionDir)',
   '_build', '$(Platform)', '$(Configuration)'))</BuildOutDir>
       <OutDir>$([MSBuild]::NormalizeDirectory('$(BuildOutDir)',
   '$(MSBuildProjectName)', 'bin'))</OutDir>
       <IntDir>$([MSBuild]::NormalizeDirectory('$(BuildOutDir)',
   '$(MSBuildProjectName)', 'obj'))</IntDir>
     </PropertyGroup>
   </Project>
   ```

3. Save and close the **Directory.Build.props** file.

Edit the project file to execute C#/WinRT

Before you can invoke the `cswinrt.exe` tool to generate the projection assembly, you must first edit the project file to specify a few project properties.

1. In **Solution Explorer**, double-click the **SimpleMathProjection** node to open the project file in the editor.

2. Update the `TargetFramework` element to target a specific Windows SDK version. This adds assembly dependencies that are necessary for the interop and projection support. This sample targets the Windows SDK version **net6.0-windows10.0.19041.0** (also known as Windows 10, version 2004). Set the `Platform` element to **AnyCPU** so that the resulting projection assembly can be referenced from any app architecture. To allow referencing applications to support earlier Windows SDK versions, you can also set the `TargetPlatformMinimumVersion` property.

   XML

   ```
   <PropertyGroup>
     <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
     <!-- Set Platform to AnyCPU to allow consumption of the projection
   assembly from any architecture. -->
     <Platform>AnyCPU</Platform>
   </PropertyGroup>
   ```

   > **Note**
   >
   > For this walkthrough and the related sample code, the solution is built for **x64** and **Release**. Note that the **SimpleMathProjection** project is configured to build for AnyCPU for all solution architecture configurations.

3. Add a second `PropertyGroup` element (immediately after the first) that sets several C#/WinRT properties.

   XML

   ```
   <PropertyGroup>
     <CsWinRTIncludes>SimpleMathComponent</CsWinRTIncludes>
     <CsWinRTGeneratedFilesDir>$(OutDir)</CsWinRTGeneratedFilesDir>
   </PropertyGroup>
   ```

   Here are some details about the settings in this example:

   - The `CsWinRTIncludes` property specifies which namespaces to project.
   - The `CsWinRTGeneratedFilesDir` property sets the output directory in which the projection source files are generated. This property is set to `OutDir`, defined in **Directory.Build.props** from the section above.

4. Save and close the **SimpleMathProjection.csproj** file, and click to **Reload projects** if necessary.

Create a NuGet package with the projection

To distribute the projection assembly for .NET application developers, you can automatically create a NuGet package when building the solution by adding some additional project properties. For .NET targets, the NuGet package needs to include the projection assembly and the implementation assembly from the component.

1. Use the steps below to add a NuGet spec (`.nuspec`) file to the **SimpleMathProjection** project.
   a. In **Solution Explorer**, right-click the **SimpleMathProjection** node, choose **Add** > **New Folder**, and name the folder **nuget**.
   b. Right-click the **nuget** folder, choose **Add** > **New Item**, choose **XML file**, and name it **SimpleMathProjection.nuspec**.

2. In **Solution Explorer**, double-click the **SimpleMathProjection** node to open the project file in the editor. Add the following property group to the now-open **SimpleMathProjection.csproj** (immediately after the two existing `PropertyGroup` elements) to automatically generate the package. These properties specify the `NuspecFile` and the directory to generate the NuGet package.

   XML

   ```
   <PropertyGroup>
     <GeneratedNugetDir>.\nuget\</GeneratedNugetDir>

   <NuspecFile>$(GeneratedNugetDir)SimpleMathProjection.nuspec</NuspecFile
   >
     <OutputPath>$(GeneratedNugetDir)</OutputPath>
     <GeneratePackageOnBuild>true</GeneratePackageOnBuild>
   </PropertyGroup>
   ```

   > **Note**
   >
   > If you prefer generating a package separately, then you can also choose to run the `nuget.exe` tool from the command line. For more information about creating a NuGet package, see Create a package using the nuget.exe CLI.

3. Open the **SimpleMathProjection.nuspec** file to edit the package creation properties, and paste the following code. The snippet below is an example NuGet spec for distributing **SimpleMathComponent** to multiple target frameworks. Note that projection assembly, **SimpleMathProjection.dll**, is specified instead of **SimpleMathComponent.winmd** for the target `lib\net6.0-`

`windows10.0.19041.0\SimpleMathProjection.dll`. This behavior is new in .NET 6 and later, and is enabled by C#/WinRT. The implementation assembly, `SimpleMathComponent.dll`, must also be distributed, and will be loaded at runtime.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<package
xmlns="http://schemas.microsoft.com/packaging/2012/06/nuspec.xsd">
  <metadata>
    <id>SimpleMathComponent</id>
    <version>0.1.0-prerelease</version>
    <authors>Contoso Math Inc.</authors>
    <description>A simple component with basic math
operations</description>
    <dependencies>
      <group targetFramework="net6.0-windows10.0.19041.0" />
      <group targetFramework=".NETCoreApp3.0" />
      <group targetFramework="UAP10.0" />
      <group targetFramework=".NETFramework4.6" />
    </dependencies>
  </metadata>
  <files>
    <!--Support .NET 6, .NET Core 3, UAP, .NET Framework 4.6, C++ -->
    <!--Architecture-neutral assemblies-->
    <file
src="..\..\_build\AnyCPU\Release\SimpleMathProjection\bin\SimpleMathPro
jection.dll" target="lib\net6.0-
windows10.0.19041.0\SimpleMathProjection.dll" />
    <file
src="..\..\_build\x64\Release\SimpleMathComponent\bin\SimpleMathCompone
nt\SimpleMathComponent.winmd"
target="lib\netcoreapp3.0\SimpleMathComponent.winmd" />
    <file
src="..\..\_build\x64\Release\SimpleMathComponent\bin\SimpleMathCompone
nt\SimpleMathComponent.winmd"
target="lib\uap10.0\SimpleMathComponent.winmd" />
    <file
src="..\..\_build\x64\Release\SimpleMathComponent\bin\SimpleMathCompone
nt\SimpleMathComponent.winmd"
target="lib\net46\SimpleMathComponent.winmd" />
    <!--Architecture-specific implementation DLLs should be copied into
RID-relative folders-->
    <file
src="..\..\_build\x64\Release\SimpleMathComponent\bin\SimpleMathCompone
nt\SimpleMathComponent.dll" target="runtimes\win10-
x64\native\SimpleMathComponent.dll" />
    <!--To support x86 and Arm64, build SimpleMathComponent for those
other architectures and uncomment the entries below.-->
    <!--<file
src="..\..\_build\Win32\Release\SimpleMathComponent\bin\SimpleMathCompo
nent\SimpleMathComponent.dll" target="runtimes\win10-
x86\native\SimpleMathComponent.dll" />-->
```

```
    <!--<file
src="..\..\_build\arm64\Release\SimpleMathComponent\bin\SimpleMathCompo
nent\SimpleMathComponent.dll" target="runtimes\win10-
arm64\native\SimpleMathComponent.dll" />-->
  </files>
</package>
```

> **Note**
>
> **SimpleMathComponent.dll**, the implementation assembly for the component, is architecture-specific. If you're supporting other platforms (for example, x86 or Arm64), then you must first build **SimpleMathComponent** for the desired platforms, and add these assembly files to the appropriate RID-relative folder. The projection assembly **SimpleMathProjection.dll** and the component **SimpleMathComponent.winmd** are both architecture-neutral.

4. Save and close the files you just edited.

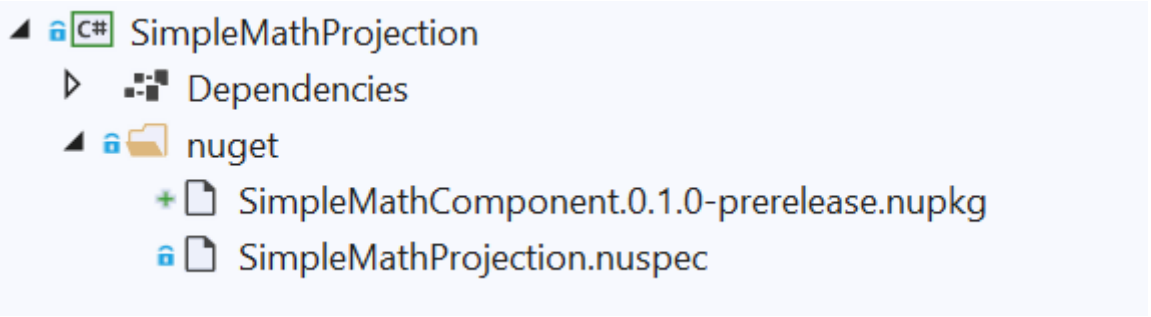
Build the solution to generate the projection and NuGet package

Before building the solution, make sure to check the **Configuration Manager** settings in Visual Studio, under **Build** > **Configuration Manager**. For this walkthrough, set the **Configuration** to **Release** and **Platform** to **x64** for the solution.

At this point you can now build the solution. Right-click on your solution node and select **Build Solution**. This will first build the **SimpleMathComponent** project, and then the **SimpleMathProjection** project. The component WinMD and implementation assembly (**SimpleMathComponent.winmd** and **SimpleMathComponent.dll**), the projection source files, and the projection assembly (**SimpleMathProjection.dll**), will all be generated under the **_build** output directory. You'll also be able to see the generated NuGet package, **SimpleMathComponent0.1.0-prerelease.nupkg**, under the **\SimpleMathProjection\nuget** folder.

> **Important**
>
> If any of the files mentioned above are not generated, then build the solution a second time. You might also need to close and reopen the solution before rebuilding.

You might need to close and reopen the solution for the `.nupkg` to appear in Visual Studio as illustrated (or just select and then deselect **Show All Files**).



Reference the NuGet package in a C# .NET 6 console application

To consume **SimpleMathComponent** from a .NET project, you can simply add to a new .NET project a reference to the **SimpleMathComponent0.1.0-prerelease.nupkg** NuGet package that we created in the previous section. The following steps demonstrate how to do that by creating a simple Console app in a separate solution.

1. Use the steps below to create a new solution containing a C# **Console App** project (creating this project in a new solution allows you to restore the **SimpleMathComponent** NuGet package independently).

   > **ⓘ Important**
   >
   > We'll be creating this new **Console App** project inside the `\CsWinRT\src\Samples\NetProjectionSample` folder, which you'll find in your downloaded or clone of the C#/WinRT projection sample.

   a. In a new instance of Visual Studio, select **File** > **New** > **Project**.
   b. In the **Create a new project** dialog box, search for the **Console App** project template. Choose the C# project template that's called simply **Console App** (with no prefixes nor suffixes), and click **Next**. If you're using Visual Studio 2019, then the project template is **Console Application**.
   c. Name the new project **SampleConsoleApp**, set its location to the same `\CsWinRT\src\Samples\NetProjectionSample` folder that the **SimpleMathComponent** and **SimpleMathProjection** folders are in, and click **Next**.
   d. On the **Additional information** page, select **.NET 6.0 (Long-term support)**, and then choose **Create**.

2. In **Solution Explorer**, double-click the **SampleConsoleApp** node to open the **SampleConsoleApp.csproj** project file, and edit the `TargetFramework` and `Platform` properties so that they look as shown in the following listing. Add the `Platform` element if it's not there.

```
XML

<PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>net6.0-windows10.0.19041.0</TargetFramework>
  <Platform>x64</Platform>
</PropertyGroup>
```

3. With the **SampleConsoleApp.csproj** project file still open, we'll next add to the **SampleConsoleApp** project a reference to the **SimpleMathComponent** NuGet package. To restore the **SimpleMathComponent** NuGet when building the project, you can use the `RestoreSources` property with the path to the **nuget** folder in your component solution. Copy the following configuration, and paste it into **SampleConsoleApp.csproj** (inside the `Project` element).

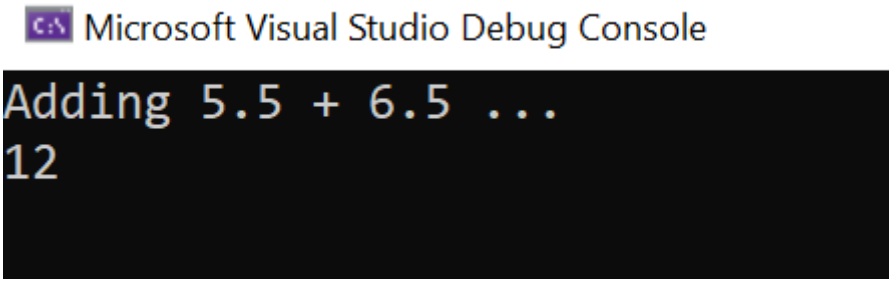
```
XML

<PropertyGroup>
  <RestoreSources>
    https://api.nuget.org/v3/index.json;
    ../SimpleMathProjection/nuget
  </RestoreSources>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="SimpleMathComponent" Version="0.1.0-
prerelease" />
</ItemGroup>
```

> **ⓘ Important**
>
> The `RestoreSources` path for the **SimpleMathComponent** package shown above is set to `../SimpleMathProjection/nuget`. That path is correct provided that you followed the steps in this walkthrough, so that the **SimpleMathComponent** and **SampleConsoleApp** projects are both in the same folder (the `NetProjectionSample` folder, in this case). If you've done something different, then you'll need to adjust that path accordingly. Alternatively, you can add a local NuGet package feed to your solution.

4. Edit the **Program.cs** file to use the functionality provided by **SimpleMathComponent**.

```
C#

var x = new SimpleMathComponent.SimpleMath();
Console.WriteLine("Adding 5.5 + 6.5 ...");
Console.WriteLine(x.add(5.5, 6.5).ToString());
```

5. Save and close the files you just edited, and build and run the console app. You should see the output below.



Known issues

- When building the projection project, you might see an error like: *Error MSB3271 There was a mismatch between the processor architecture of the project being built "MSIL" and the processor architecture, "x86", of the implementation file "..\SimpleMathComponent.dll" for "..\SimpleMathComponent.winmd". This mismatch may cause runtime failures. Please consider changing the targeted processor architecture of your project through the Configuration Manager so as to align the processor architectures between your project and implementation file, or choose a winmd file with an implementation file that has a processor architecture which matches the targeted processor architecture of your project.* To work around this error, add the following property to your C# library project file:

```
XML

<PropertyGroup>
    <!-- Workaround for MSB3271 error on processor architecture
mismatch -->

<ResolveAssemblyWarnOrErrorOnTargetArchitectureMismatch>None</ResolveAs
semblyWarnOrErrorOnTargetArchitectureMismatch>
</PropertyGroup>
```

Further considerations

The C# projection (or interop) assembly that we showed how to create in this topic is quite simple—it doesn't have dependencies on other components. But to generate a C# projection for a C++/WinRT component that has references to Windows App SDK types, in the projection project you'd need to add a reference to the Windows App SDK NuGet package. If any such references are missing, then you'll see errors such as "Type <T> could not be found".

Another thing that we do in this topic is to distribute the projection as a NuGet package. That *is* currently necessary.

Resources

- Full code sample for this walkthrough
- C#/WinRT NuGet documentation
- NuGet.exe documentation

Author Windows Runtime components with C#/WinRT

Article • 10/24/2022

> **Note**
>
> Authoring Windows Runtime components with C#/WinRT is supported on .NET 6 and later.

The C#/WinRT NuGet package provides support for authoring your own Windows Runtime types and components in C#, and consuming them from any Windows Runtime-compatible language such as C++/WinRT or Rust. C#/WinRT authoring and hosting support requires .NET 6 and Visual Studio 2022, and it is intended to support desktop application scenarios including the Windows App SDK, and WinUI3.

For a walkthrough showing how to author a Windows Runtime component with .NET 6, and how to consume it from a C++/WinRT console application, see Walkthrough: Create a C#/WinRT component and consume it from C++/WinRT.

For more details, and to search for or file any issues, refer to the C#/WinRT Github repo and Authoring C#/WinRT Components.

Declaring types in Windows Runtime components

While authoring your Windows Runtime component, follow the guidelines and type restrictions outlined in the existing UWP documentation about Windows Runtime components (see Windows Runtime components with C# and Visual Basic). The component can for the most part be implemented like any other C# library. However, there are restrictions on the public types in the component that will be exposed to the Windows Runtime and declared in the generated `.winmd` for others to consume.

Externally, you can expose only Windows Runtime types for parameters and return values. You can use built-in C# types as part of the public surface of the component as long as there is a mapping from the .NET type to WinRT (see .NET mappings of WinRT types in C#/WinRT), and they will appear to users of the component as the corresponding Windows Runtime types. Windows Runtime types from other Windows Runtime components and the Windows SDK can also be used as part of the public

implementation of the component, such as `in` parameters, return types, and class inheritance.

> **Note**
>
> There are some Windows Runtime types that are mapped to .NET types (see **.NET mappings of WinRT types in C#/WinRT**). These .NET types can be used in the public interface of your Windows Runtime component, and they will appear to users of the component as the corresponding Windows Runtime types.

Related topics

- Authoring C#/WinRT Components
- Walkthrough: Create a C#/WinRT component and consume it from C++/WinRT
- Diagnose C#/WinRT component errors
- .NET mappings of WinRT types in C#/WinRT

Walkthrough—Create a C#/WinRT component, and consume it from C++/WinRT

Article • 10/24/2022

C#/WinRT enables developers using .NET to author their own Windows Runtime components in C# using a class library project. Authored components can be consumed in native desktop applications as a package reference or as a project reference with a few modifications.

This walkthrough demonstrates how to create a simple Windows Runtime component using C#/WinRT, distribute the component as a NuGet package, and consume the component from a C++/WinRT console application. For the full sample that provides the code for this article, see the C#/WinRT authoring sample. For more details about authoring, see Authoring components.

For a walkthrough on authoring WinUI controls with C#/WinRT specifically for use in Windows App SDK applications, refer to the article Walkthrough: Author a C# component with WinUI 3 controls and consume from a C++ Windows App SDK application

Prerequisites

This walkthrough requires the following tools and components:

- Visual Studio 2022
- .NET 6.0 SDK or later
- C++/WinRT VSIX for C++/WinRT project templates

Create a simple Windows Runtime Component using C#/WinRT

Begin by creating a new project in Visual Studio. Select the **Class Library** project template, and name the project **AuthoringDemo**. You'll need to make the following additions and modifications to the project:

1. Update the `TargetFramework` in the **AuthoringDemo.csproj** file and add the following elements to the `PropertyGroup`:

XML

```
<PropertyGroup>
    <TargetFramework>net8.0-windows10.0.19041.0</TargetFramework>
    <Platforms>x64</Platforms>
</PropertyGroup>
```

To access Windows Runtime types, you need to set a specific Windows SDK version in the TFM. For more details on the supported version, see .NET 6 and later: Use the TFM option.

2. Install the Microsoft.Windows.CsWinRT NuGet package in your project.

   a. In Solution Explorer, right click on the project node and select **Manage NuGet Packages**.

   b. Search for the **Microsoft.Windows.CsWinRT** NuGet package and install the latest version. This walkthrough uses C#/WinRT version 1.4.1.

3. Add a new `PropertyGroup` element that sets the `CsWinRTComponent` property. This specifies that your project is a Windows Runtime component so that a `.winmd` file is generated when you build the project.

   XML

   ```
   <PropertyGroup>
       <CsWinRTComponent>true</CsWinRTComponent>
   </PropertyGroup>
   ```

   For a full list of C#/WinRT project properties, refer to the C#/WinRT NuGet documentation.

4. You can author your runtime classes using library `.cs` class files. Right click on the `Class1.cs` file, and rename it to `Example.cs`. Add the following code to this file, which adds a public property and method to the runtime class. Remember to mark any classes you want to expose in the runtime component as `public`.

   C#

   ```
   namespace AuthoringDemo
   {
       public sealed class Example
       {
           public int SampleProperty { get; set; }

           public static string SayHello()
   ```

```
        {
            return "Hello from your C# WinRT component";
        }
    }
}
```

5. You can now build the project to generate the `.winmd` file for your component. Right-click on the project in **Solution Explorer**, and click **Build**. You'll see the generated `AuthoringDemo.winmd` file in your build output folder.

Generate a NuGet package for the component

Most developers will want to distribute and share their Windows Runtime component as a NuGet package. Another option is to consume the component as a project reference. The following steps demonstrate how to package the **AuthoringDemo** component. When you generate the package, C#/WinRT configures the component and hosting assemblies in the package to enable consumption from native applications.

There are several ways to generate the NuGet package:

- If you want to generate a NuGet package every time you build the project, add the following property to the **AuthoringDemo** project file and then rebuild the project.

  XML

  ```
  <PropertyGroup>
      <GeneratePackageOnBuild>true</GeneratePackageOnBuild>
  </PropertyGroup>
  ```

- Alternatively, you can generate a NuGet package by right clicking the **AuthoringDemo** project in **Solution Explorer** and selecting **Pack**.

When you build the package, the **Build** window should indicate that the NuGet package `AuthoringDemo.1.0.0.nupkg` was successfully created. See Create a package using the dotnet CLI for more details on NuGet package properties with the .NET CLI.

Consume the component from a C++/WinRT app

C#/WinRT authored Windows Runtime components can be consumed from any Windows Runtime (WinRT)-compatible language. The following steps demonstrate how to call the authored component above in a C++/WinRT console application.

> **Note**
>
> Consuming a C#/WinRT component from C#/.NET apps is supported by both package reference or project reference. This scenario is equivalent to consuming any ordinary C# class library and does not involve WinRT activation in most cases. Starting with C#/WinRT 1.3.5, project references for C# consumers require .NET 6.

1. Add a new **C++/WinRT Console Application** project to your solution. Note that this project can also be part of a different solution if you choose so.

   a. In **Solution Explorer**, right click your solution node and click **Add** -> **New Project**.

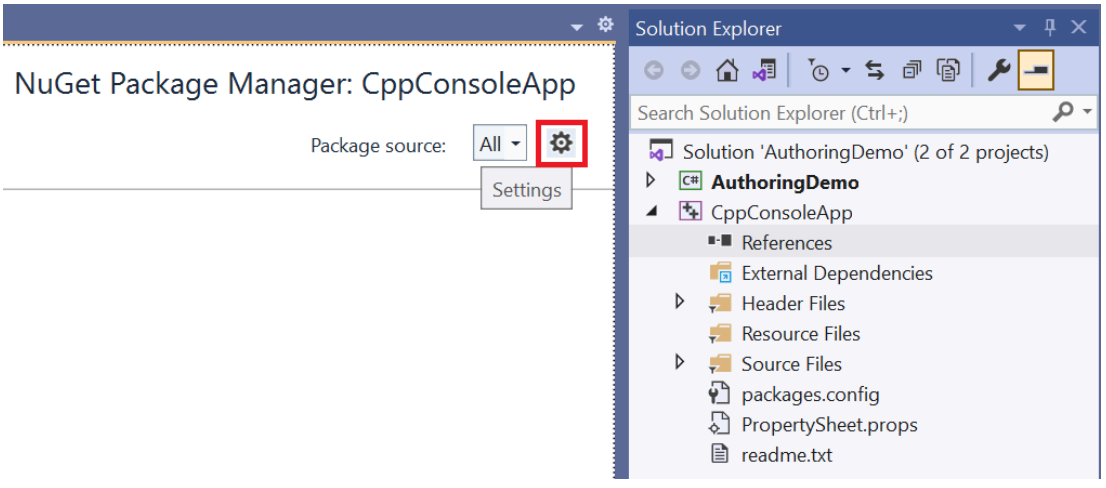
   b. In the **Add New Project dialog box**, search for the **C++/WinRT Console Application** project template. Select the template and click **Next**.

   c. Name the new project **CppConsoleApp** and click **Create**.

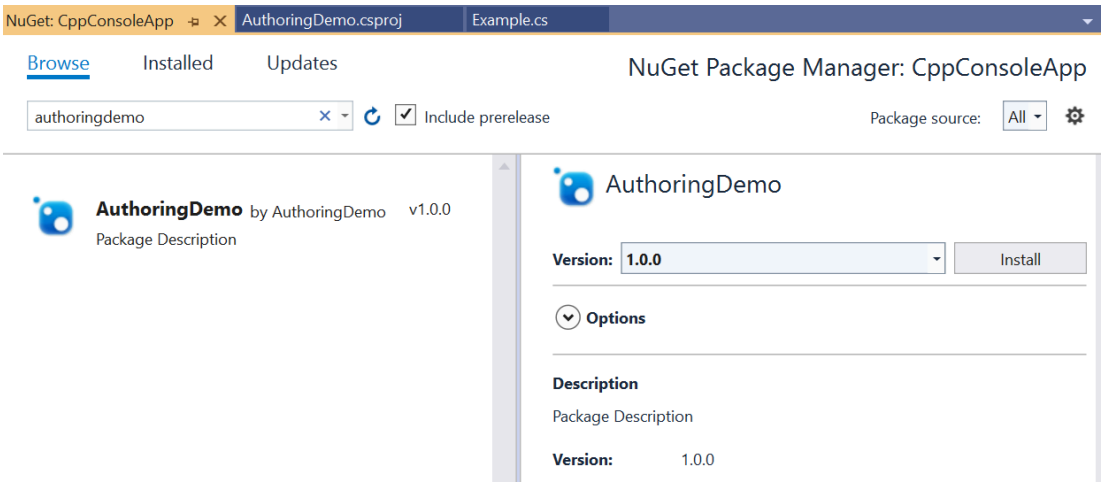
2. Add a reference to the AuthoringDemo component, either as a NuGet package or a project reference.

   - **Option 1 (Package reference)**:

     a. Right click the **CppConsoleApp** project and select **Manage NuGet packages**. You may need to configure your package sources to add a reference to the AuthoringDemo NuGet package. To do this, click the **Settings** icon in NuGet Package Manager and add a package source to the appropriate path.



     b. After configuring your package sources, search for the **AuthoringDemo** package and click **Install**.



- **Option 2 (Project reference)**:

  a. Right click the **CppConsoleApp** project and select **Add** -> **Reference**. Under the **Projects** node, add a reference to the **AuthoringDemo** project.

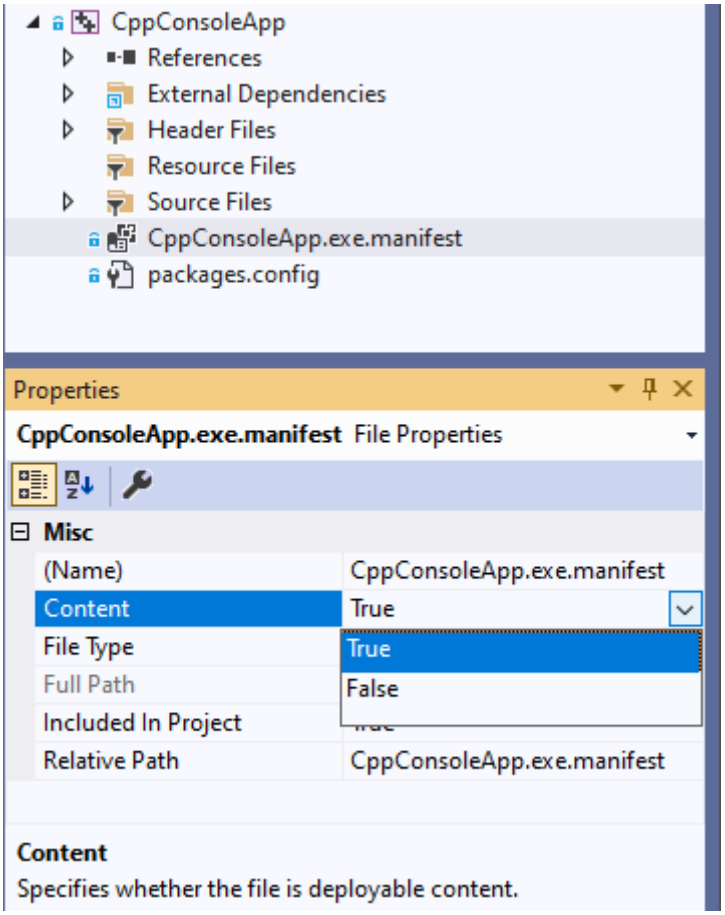
3. To host the component, you will need to add a manifest file for activatable class registrations. For more details about managed component hosting, see Managed component hosting.

   a. To add the manifest file, again right click on the project and choose **Add** -> **New Item**. Search for the **Text File** template and name it **CppConsoleApp.exe.manifest**. Paste the following contents, which specify the runtime classes using activatable class registration entries:

```XML
<?xml version="1.0" encoding="utf-8"?>
<assembly manifestVersion="1.0" xmlns="urn:schemas-microsoft-
com:asm.v1">
    <assemblyIdentity version="1.0.0.0" name="CppConsoleApp"/>
    <file name="WinRT.Host.dll">
        <activatableClass
            name="AuthoringDemo.Example"
            threadingModel="both"
            xmlns="urn:schemas-microsoft-com:winrt.v1" />
    </file>
</assembly>
```

   The application manifest file is required for apps that are not packaged. For packaged apps, the app consumer needs to register the activatable classes in their `Package.appxmanifest` package manifest file, as explained in Walkthrough: Create a C# component with WinUI 3 controls and consume from a C++ Windows App SDK application.

b. Modify the project to include the manifest file in the output when deploying the project. Click the **CppConsoleApp.exe.manifest** file in **Solution Explorer** and set the **Content** property to **True**. Here is an example of what this looks like.



4. Open **pch.h** under the project's Header Files, and add the following line of code to include your component.

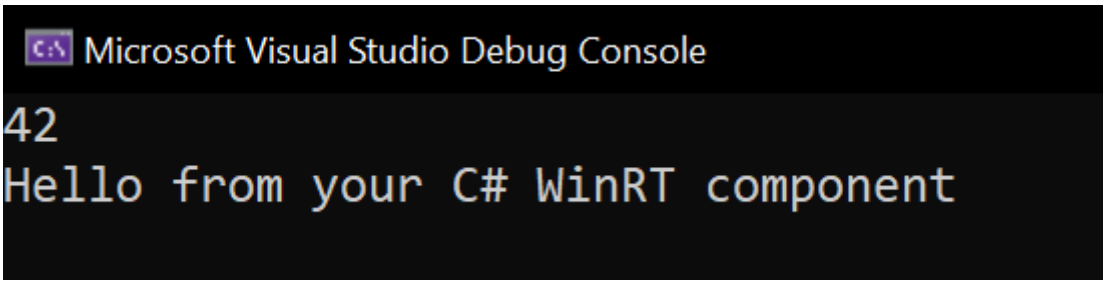
```
C++

#include <winrt/AuthoringDemo.h>
```

5. Open **main.cpp** under the project's Source Files, and replace it with the following contents.

```
C++

#include "pch.h"
#include "iostream"

using namespace winrt;
using namespace Windows::Foundation;

int main()
{
    init_apartment();
```

```
    AuthoringDemo::Example ex;
    ex.SampleProperty(42);
    std::wcout << ex.SampleProperty() << std::endl;
    std::wcout << ex.SayHello().c_str() << std::endl;
}
```

6. Build and run the **CppConsoleApp** project. You should now see the output below.



Related topics

- Sample code
- Authoring components
- Managed component hosting

Walkthrough—Create a C# component with WinUI 3 controls, and consume it from a C++/WinRT app that uses the Windows App SDK

Article • 04/19/2023

C#/WinRT provides support for authoring Windows Runtime components, including WinUI custom types and custom controls. These components can be consumed from either C# or C++/WinRT applications that use the Windows App SDK. We recommend using C#/WinRT v1.6.4 or later to author runtime components with NuGet packaging support.

For more details about the supported scenarios, refer to Authoring C#/WinRT components in the C#/WinRT GitHub repo.

This walkthrough demonstrates how to author a C# component with a custom WinUI 3 control, and how to consume that component from a C++/WinRT app, using the Windows App SDK project templates.

Prerequisites

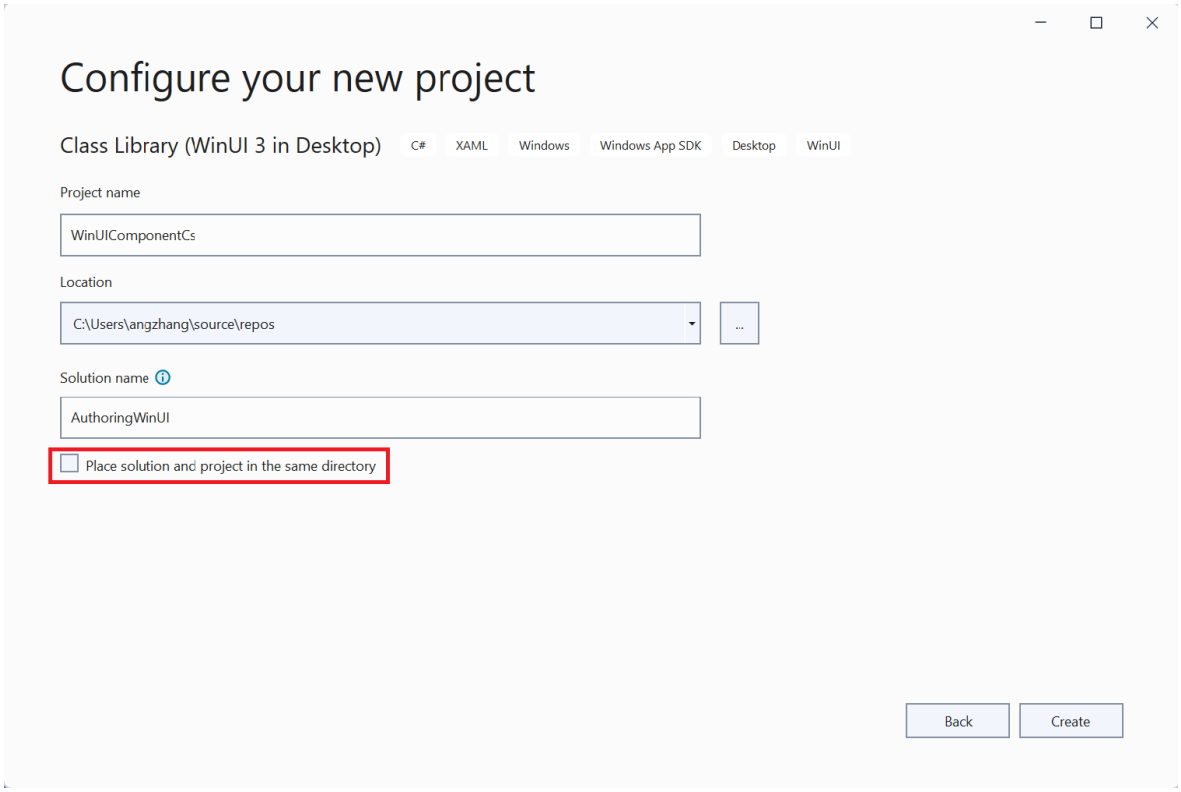
This walkthrough requires the following tools and components:

- Visual Studio 2022
- .NET 6.0 SDK
- Windows App SDK VSIX (1.1 from the stable channel)

Author your C#/WinRT component using the Windows App SDK

1. Create a new C# library project using the **Class Library (WinUI 3 in Desktop)** template provided by the Windows App SDK. For this walkthrough, we've named the library project **WinUIComponentCs**, and the solution **AuthoringWinUI**.

   Leave the **Place solution and project in the same directory** box unchecked (otherwise, the `packages` folder for the C++ application in the preceding section will end up interfering with the C# library project).



2. Delete the `Class1.cs` file that's included by default.

3. Install the latest Microsoft.Windows.CsWinRT NuGet package in your project.

   i. In Solution Explorer, right-click on the project node, and select **Manage NuGet Packages**.

   ii. Search for the **Microsoft.Windows.CsWinRT** NuGet package, and install the latest version.

4. Add the following properties to your library project:

   XML

   ```
   <PropertyGroup>
       <CsWinRTComponent>true</CsWinRTComponent>
   </PropertyGroup>
   ```

   - The `CsWinRTComponent` property specifies that your project is a Windows Runtime component so that a `.winmd` file is generated when you build the project.

5. Add a custom control or user control to your library. To do this, right-click on your project in Visual Studio, click **Add** > **New Item**, and select **WinUI** on the left pane. For this walkthrough, we added a new **User Control (WinUI 3)** and named it `NameReporter.xaml`. The **NameReporter** user control allows a user to enter a first

and last name into the appropriate **TextBox** control, and click a button. The control then displays a message box with the name that the user entered.

6. Paste the following code in the `NameReporter.xaml` file:

XML

```
<UserControl
x:Class="WinUIComponentCs.NameReporter"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:local="using:WinUIComponentCs"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d">

    <StackPanel HorizontalAlignment="Center">
        <StackPanel.Resources>
            <Style x:Key="BasicTextStyle" TargetType="TextBlock"
BasedOn="{StaticResource BodyTextBlockStyle}">
                <Setter Property="Margin" Value="10,10,10,10"/>
            </Style>
        </StackPanel.Resources>

        <TextBlock Text="Enter your name." Margin="0,0,0,10"/>
        <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
            <TextBlock Style="{StaticResource BasicTextStyle}">
                First Name:
            </TextBlock>
            <TextBox Name="firstName" />
        </StackPanel>
        <StackPanel Orientation="Horizontal" Margin="0,0,0,10">
            <TextBlock Style="{StaticResource BasicTextStyle}">
                Last Name:
            </TextBlock>
            <TextBox Name="lastName" />
        </StackPanel>
        <Button Content="Submit" Click="Button_Click"
Margin="0,0,0,10"/>
        <TextBlock Name="result" Style="{StaticResource
BasicTextStyle}" Margin="0,0,0,10"/>
    </StackPanel>
</UserControl>
```

7. Add the following method to `NameReporter.xaml.cs`:

C#

```
using System.Text;
...
private void Button_Click(object sender, RoutedEventArgs e)
{
```

```
    StringBuilder displayText = new StringBuilder("Hello, ");
    displayText.AppendFormat("{0} {1}.", firstName.Text,
lastName.Text);
    result.Text = displayText.ToString();
}
```

8. You can now build the **WinUIComponentCs** project to generate a `.winmd` file for the component.

> **Note**
>
> You can also package the component as a NuGet package for end app consumers to reference. For more details, see **Authoring C#/WinRT components** on the C#/WinRT Github repo.

Reference the component from a Windows App SDK C++/WinRT app

The following steps show how to consume the component created from the previous section from a C++/WinRT Windows App SDK application. Consuming a C#/WinRT component from C++ currently requires using the single-project **Blank App, Packaged (WinUI 3 in Desktop)** template. Note that C# components can also be referenced from C# packaged apps without class registrations.

Consumption from packaged apps that use a separate **Windows Application Packaging (WAP)** project is not currently supported. See Authoring C#/WinRT components in the C#/WinRT GitHub repo for the latest updates on supported project configurations.

1. Add a new C++ Windows App SDK application project to your solution. Right-click on your solution in Visual Studio, and select **Add** > **New Project**. Select the C++ **Blank App, Packaged (WinUI 3 in Desktop)** template provided by the Windows App SDK. For this walkthrough, we named the app **CppApp**.

2. Add a project reference from the C++ app to the C# component. In Visual Studio, right-click on the C++ project and choose **Add** > **Reference**, and select the **WinUIComponentCs** project.

   > **Note**
   >
   > Consuming components as a NuGet package reference is supported with some limitations. Namely, components with custom user controls can't

currently be consumed as a NuGet package reference.

3. In the app's `pch.h` header file, add the following lines:

```C++
#include <winrt/WinUIComponentCs.h>
#include <winrt/WinUIComponentCs.WinUIComponentCs_XamlTypeInfo.h>
```

4. Open up the package manifest file, `Package.appxmanifest`.

> **Note**
>
> There's a known issue where the `Package.appxmanifest` file doesn't appear in Visual Studio Solution Explorer. To workaround that, right-click on your C++ project, select **Unload Project**, and double-click on the project to open the `CppApp.vcxproj` file. Add the following entry to the project file, and then reload the project:

```XML
<ItemGroup>
    <AppxManifest Include="Package.appxmanifest">
    <SubType>Designer</SubType>
    </AppxManifest>
</ItemGroup>
```

In `Package.appxmanifest`, add the following activatable class registrations. You'll also need an additional `ActivatableClass` entry for the **WinUIComponentCs.WinUIComponentCs_XamlTypeInfo.XamlMetaDataProvider** class in order to activate the WinUI types. Right-click on the `Package.appxmanifest` file and select **Open With** > **XML (Text Editor)** in order to edit the file.

```XML
<!--In order to host the C# component from C++, you must add the
following Extension group and list the activatable classes-->
<Extensions>
    <Extension Category="windows.activatableClass.inProcessServer">
        <InProcessServer>
            <Path>WinRT.Host.dll</Path>
            <ActivatableClass
ActivatableClassId="WinUIComponentCs.NameReporter"
ThreadingModel="both" />
```

```
            <ActivatableClass
ActivatableClassId="WinUIComponentCs.WinUIComponentCs_XamlTypeInfo.Xaml
MetaDataProvider" ThreadingModel="both" />
        </InProcessServer>
    </Extension>
</Extensions>
```

5. Open the `MainWindow.xaml` file.

   i. Add a reference to the component's namespace at the top of the file.

   XML

   ```
   xmlns:custom="using:WinUIComponentCs"
   ```

   ii. Add the user control to the existing XAML code.

   XML

   ```
   <StackPanel>
       ...
       <custom:NameReporter/>
   </StackPanel>
   ```

6. Set **CppApp** as the startup project—right-click on **CppApp**, and select **Set as Startup Project**. Set the solution configuration to `x86`. Before building, you might also need to retarget your solution to build with the Visual Studio 2022 build tools. Right-click on the solution, select **Retarget solution**, and upgrade the Platform Toolset to **v143**.

7. Build and run the app to see the custom **NameReporter** control.

Known issues

- Consuming a C# component as a project reference requires `PublishReadyToRun` to be set to `False`. See Github Issue #1151 for more details.
- Consuming a C# component built for `AnyCPU` from C++ is supported only from `x86` applications currently. `x64` and `Arm64` apps result in a runtime error similar to: *%1 is not a valid Win32 application.* See Github Issue #1151 for more details.

Related topics

- C#/WinRT Authoring Sample app

- Authoring C#/WinRT Components
- Managed component hosting

Diagnose C#/WinRT component errors

Article • 05/25/2022

This article provides additional information about restrictions on Windows Runtime Components written with C#/WinRT. It expands on the information that is provided in error messages from C#/WinRT when an author builds their component. For existing UWP .NET Native managed components, the metadata for C# WinRT components is generated using Winmdexp.exe, a .NET tool. Now that Windows Runtime support is decoupled from .NET, C#/WinRT provides the tooling to generate a .winmd file from your component. The Windows Runtime has more constraints on code than a C# class library, and the C#/WinRT Diagnostic Scanner alerts you of these before generating a .winmd file.

This article covers the errors reported in your build from C#/WinRT. This article serves as an updated version of the information on restrictions for existing UWP .NET Native managed components which use the Winmdexp.exe tool.

Search for the error message text (omitting specific values for placeholders) or the message number. If you don't find the information you need here, you can help us improve the documentation by using the feedback button at the end of this article. In your feedback, please include the error message. Alternatively, you can file a bug at the C#/WinRT repo.

This article organizes error messages by scenario.

Implementing interfaces that aren't valid Windows Runtime interfaces

C#/WinRT components cannot implement certain Windows Runtime interfaces, such as the Windows Runtime interfaces that represent asynchronous actions or operations (**IAsyncAction**, **IAsyncActionWithProgress<TProgress>**, **IAsyncOperation<TResult>**, or **IAsyncOperationWithProgress<TResult,TProgress>**). Instead, use the **AsyncInfo** class for generating async operations in Windows Runtime components. Note: these are not the only invalid interfaces, for example a class cannot implement **System.Exception**.

Error number	Message text
CsWinRT1008	Windows Runtime component type {0} cannot implement interface {1}, as the interface is not a valid Windows Runtime interface

Attribute related errors

In the Windows Runtime, overloaded methods can have the same number of parameters only if one overload is specified as the default overload. Use the attribute **Windows.Foundation.Metadata.DefaultOverload** (CsWinRT1015, 1016).

When arrays are used as inputs or outputs in either functions or properties, they must be either read-only or write-only (CsWinRT 1025). The attributes **System.Runtime.InteropServices.WindowsRuntime.ReadOnlyArray** and **System.Runtime.InteropServices.WindowsRuntime.WriteOnlyArray** are provided for you to use. The provided attributes are only for use on parameters of the array type (CsWinRT1026), and only one should be applied per parameter (CsWinRT1023).

You do not need to apply any attribute to an array parameter marked **out**, as it is assumed to be write-only. There is an error message if you decorate it as read-only in this case (CsWinRT1024).

The attributes **System.Runtime.InteropServices.InAttribute** and **System.Runtime.InteropServices.OutAttribute** should not be used on parameters of any type (CsWinRT1021,1022).

Error number	Message text
CsWinRT1015	In class {2}: Multiple {0}-parameter overloads of '{1}' are decorated with Windows.Foundation.Metadata.DefaultOverloadAttribute. The attribute may only be applied to one overload of the method.
CsWinRT1016	In class {2}: The {0}-parameter overloads of {1} must have exactly one method specified as the default overload by decorating it with the attribute Windows.Foundation.Metadata.DefaultOverloadAttribute.
CsWinRT1021	Method '{0}' has parameter '{1}' which is an array, and which has either a System.Runtime.InteropServices.InAttribute or a System.Runtime.InteropServices.OutAttribute. In the Windows Runtime, array parameters must have either ReadOnlyArray or WriteOnlyArray. Please remove these attributes or replace them with the appropriate Windows Runtime attribute if necessary.
CsWinRT1022	Method '{0}' has parameter '{1}' with a System.Runtime.InteropServices.InAttribute or System.Runtime.InteropServices.OutAttribute.Windows Runtime does not support marking parameters with System.Runtime.InteropServices.InAttribute or System.Runtime.InteropServices.OutAttribute. Please consider removing System.Runtime.InteropServices.InAttribute and replace System.Runtime.InteropServices.OutAttribute with 'out' modifier instead.

Error number	Message text
CsWinRT1023	Method '{0}' has parameter '{1}' which is an array, and which has both ReadOnlyArray and WriteOnlyArray. In the Windows Runtime, the contents array parameters must be either readable or writable. Please remove one of the attributes from '{1}'.
CsWinRT1024	Method '{0}' has an output parameter '{1}' which is an array, but which has ReadOnlyArray attribute. In the Windows Runtime, the contents of output arrays are writable. Please remove the attribute from '{1}'.
CsWinRT1025	Method '{0}' has parameter '{1}' which is an array. In the Windows Runtime, the contents of array parameters must be either readable or writable. Please apply either ReadOnlyArray or WriteOnlyArray to '{1}'.
CsWinRT1026	Method '{0}' has parameter '{1}' which is not an array, and which has either a ReadOnlyArray attribute or a WriteOnlyArray attribute. Windows Runtime does not support marking non-array parameters with ReadOnlyArray or WriteOnlyArray."

Namespace errors and invalid names for the output file

In the Windows Runtime, all the public types in a Windows metadata (.winmd) file must be in a namespace that shares the .winmd file name, or in sub-namespaces of the file name. For example, if your Visual Studio project is named A.B (that is, your Windows Runtime component is A.B.winmd), it can contain public classes A.B.Class1 and A.B.C.Class2, but not A.Class3 or D.Class4.

> ⓘ **Note**
>
> These restrictions apply only to public types, not to private types used in your implementation.

In the case of A.Class3, you can either move Class3 to another namespace or change the name of the Windows Runtime component to A.winmd. In the previous example, code that calls A.B.winmd will be unable to locate A.Class3.

In the case of D.Class4, no file name can contain both D.Class4 and classes in the A.B namespace, so changing the name of the Windows Runtime component is not an option. You can either move D.Class4 to another namespace, or put it in another Windows Runtime component.

The file system can't distinguish between uppercase and lowercase, so namespaces that differ by case are not allowed (CsWinRT1002).

Error number	Message text
CsWinRT1001	A public type has a namespace ('{1}') that shares no common prefix with other namespaces ('{0}'). All types within a Windows Metadata file must exist in a sub namespace of the namespace that is implied by the file name.
CsWinRT1002	Multiple namespaces found with the name '{0}'; namespace names cannot differ only by case in the Windows Runtime.

Exporting types that aren't valid Windows Runtime types

The public interface of your component must expose only Windows Runtime types. However, .NET provides mappings for a number of commonly used types that are slightly different in .NET and the Windows Runtime. This enables the .NET developer to work with familiar types instead of learning new ones. You can use these mapped .NET Framework types in the public interface of your component. For more information, see Declaring types in Windows Runtime Components, Passing Windows Runtime types to managed code, and .NET mappings of Windows Runtime types.

Many of these mappings are interfaces. For example, IList<T> maps to the Windows Runtime interface IVector<T>. If you use List<string> instead of IList<string> as a parameter type, C#/WinRT provides a list of alternatives that includes all the mapped interfaces implemented by List<T>. If you use nested generic types, such as List<Dictionary<int, string>>, C#/WinRT offers choices for each level of nesting. These lists can become quite long.

In general, the best choice is the interface that is closest to the type. For example, for Dictionary<int, string>, the best choice is most likely IDictionary<int, string>.

Error number	Message text
CsWinRT1006	The member '{0}' has the type '{1}' in its signature. The type '{1}' is not a valid Windows Runtime type. Yet, the type (or its generic parameters) implement interfaces that are valid Windows Runtime types. Consider changing the type '{1} in the member signature to one of the following types from System.Collections.Generic: {2}.

In the Windows Runtime, arrays in member signatures must be one-dimensional with a lower bound of 0 (zero). Nested array types such as `myArray[][]` (CsWinRT1017) and `myArray[,]` (CsWinRT1018) are not allowed.

> **Note**
>
> This restriction does not apply to arrays you use internally in your implementation.

Error number	Message text
CsWinRT1017	Method {0} has a nested array of type {1} in its signature. Arrays in Windows Runtime method signature cannot be nested.
CsWinRT1018	Method '{0}' has a multi-dimensional array of type '{1}' in its signature. Arrays in Windows Runtime method signatures must be one dimensional.

Structures that contain fields of disallowed types

In the Windows Runtime, a structure can contain only fields, and only structures can contain fields. Those fields must be public. Valid field types include enumerations, structures, and primitive types.

Error number	Message text
CsWinRT1007	Structure {0} contains no public fields. Windows Runtime structures must contain at least one public field.
CsWinRT1011	Structure {0} has non-public field. All fields must be public for Windows Runtime structures.
CsWinRT1012	Structure {0} has const field. Constants can only appear on Windows Runtime enumerations.
CsWinRT1013	Structure {0} has field of type {1}; {1} is not a valid Windows Runtime field type. Each field in a Windows Runtime structure can only be UInt8, Int16, UInt16, Int32, UInt32, Int64, UInt64, Single, Double, Boolean, String, Enum, or itself a structure.

Parameter name conflicts with generated code

In the Windows Runtime, return values are considered to be output parameters, and the names of parameters must be unique. By default, C#/WinRT gives the return value the name `__retval`. If your method has a parameter named `__retval`, you will get error CsWinRT1010. To correct this, give your parameter a name other than `__retvalue`.

Error number	Message text
CsWinRT1010	The parameter name {1} in method {0} is the same as the return value parameter name used in the generated C#/WinRT interop; use a different parameter name.

Miscellaneous

Other restrictions in a C#/WinRT authored component include:

- You cannot expose overloaded operators on public types.
- Classes and interfaces cannot be generic.
- Classes must be sealed.
- Parameters cannot be passed by reference, e.g. using the **ref** keyword.
- Properties must have a public get method.
- There must be at least one public type (class or interface) in your component's namespace.

Error number	Message text
CsWinRT1014	'{0}' is an operator overload. Managed types cannot expose operator overloads in the Windows Runtime.
CsWinRT1004	Type {0} is generic. Windows Runtime types cannot be generic.
CsWinRT1005	Exporting unsealed types is not supported in CsWinRT. Please mark type {0} as sealed.
CsWinRT1020	Method '{0}' has parameter '{1}' marked `ref`. Reference parameters are not allowed in Windows Runtime.
CsWinRT1000	Property '{0}' does not have a public getter method. Windows Runtime does not support setter-only properties.
CsWinRT1003	Windows Runtime components must have at least one public type

In the Windows Runtime, a class can have only one constructor with a given number of parameters. For example, you can't have one constructor that has a single parameter of

type **string** and another constructor that has a single parameter of type **int**. The only workaround is to use a different number of parameters for each constructor.

Error number	Message text
CsWinRT1009	Classes cannot have multiple constructors of the same arity in the Windows Runtime, class {0} has multiple {1}-arity constructors.

Using brushes to paint backgrounds, foregrounds, and outlines

Article • 11/26/2024

Use **Brush** objects to paint the interiors and outlines of XAML shapes, text, and controls, making them visible in your application UI.

> **Important APIs**: Brush class

Introduction to brushes

To paint a **Shape**, text, or parts of a **Control** that is displayed on the app canvas, set the **Fill** property of the **Shape** or the **Background** and **Foreground** properties of a **Control** to a **Brush** value.

The different types of brushes are:

- **AcrylicBrush**
- **SolidColorBrush**
- **LinearGradientBrush**
- **RadialGradientBrush**
- **ImageBrush**
- **WebViewBrush**
- **XamlCompositionBrushBase**

Solid color brushes

A **SolidColorBrush** paints an area with a single **Color**, such as red or blue. This is the most basic brush. In XAML, there are three ways to define a **SolidColorBrush** and the color it specifies: predefined color names, hexadecimal color values, or the property element syntax.

Predefined color names

You can use a predefined color name, such as **Yellow** or **Magenta**. There are 256 available named colors. The XAML parser converts the color name to a **Color** structure with the correct color channels. The 256 named colors are based on the *X11* color names from the Cascading Style Sheets, Level 3 (CSS3) specification, so you may already

be familiar with this list of named colors if you have previous experience with web development or design.

Here's an example that sets the **Fill** property of a **Rectangle** to the predefined color **Red**.

XAML

```
<Rectangle Width="100" Height="100" Fill="Red" />
```

SolidColorBrush applied to a Rectangle

If you are defining a **SolidColorBrush** using code rather than XAML, each named color is available as a static property value of the **Colors** class. For example, to declare a **Color** value of a **SolidColorBrush** to represent the named color "Orchid", set the **Color** value to the static value **Colors.Orchid**.

Hexadecimal color values

You can use a hexadecimal format string to declare precise 24-bit color values with 8-bit alpha channel for a **SolidColorBrush**. Two characters in the range 0 to F define each component value, and the component value order of the hexadecimal string is: alpha channel (opacity), red channel, green channel, and blue channel (**ARGB**). For example, the hexadecimal value "#FFFF0000" defines fully opaque red (alpha="FF", red="FF", green="00", and blue="00").

This XAML example sets the **Fill** property of a **Rectangle** to the hexadecimal value "#FFFF0000", and gives an identical result to using the named color **Colors.Red**.

XML

```
<StackPanel>
  <Rectangle Width="100" Height="100" Fill="#FFFF0000" />
</StackPanel>
```

Property element syntax

You can use property element syntax to define a **SolidColorBrush**. This syntax is more verbose than the previous methods, but you can specify additional property values on an element, such as the **Opacity**. For more info on XAML syntax, including property element syntax, see the XAML overview and XAML syntax guide.

In the previous examples, the brush being created is created implicitly and automatically, as part of a deliberate XAML language shorthand that helps keep UI definitions simple for the most common cases. The next example creates a **Rectangle** and explicitly creates the **SolidColorBrush** as an element value for a **Rectangle.Fill** property. The **Color** of the **SolidColorBrush** is set to **Blue** and the **Opacity** is set to 0.5.

XML

```
<Rectangle Width="200" Height="150">
    <Rectangle.Fill>
        <SolidColorBrush Color="Blue" Opacity="0.5" />
    </Rectangle.Fill>
</Rectangle>
```

Linear gradient brushes

A **LinearGradientBrush** paints an area with a gradient that's defined along a line. This line is called the *gradient axis*. You specify the gradient's colors and their locations along the gradient axis using **GradientStop** objects. By default, the gradient axis runs from the upper left corner to the lower right corner of the area that the brush paints, resulting in a diagonal shading.

The **GradientStop** is the basic building block of a gradient brush. A gradient stop specifies what the **Color** of the brush is at an **Offset** along the gradient axis, when the brush is applied to the area being painted.

The gradient stop's **Color** property specifies the color of the gradient stop. You can set the color by using a predefined color name or by specifying the hexadecimal **ARGB** values.

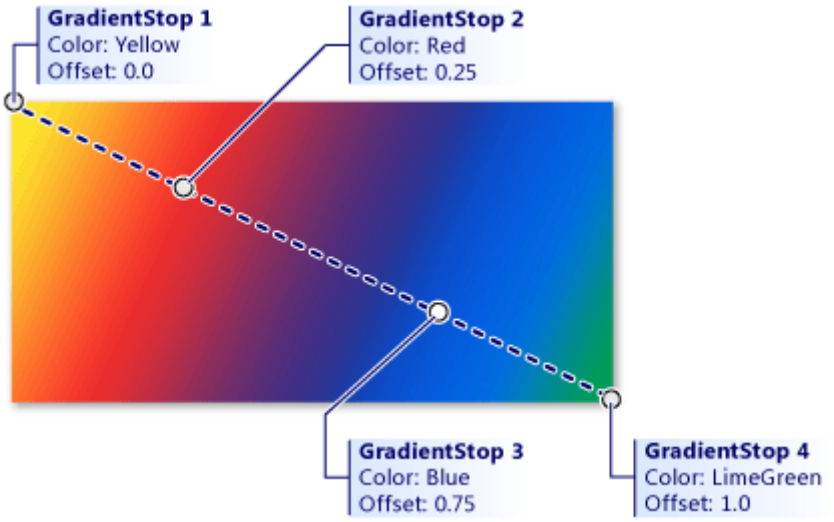
The **Offset** property of a **GradientStop** specifies the position of each **GradientStop** along the gradient axis. The **Offset** is a **double** that ranges from 0 to 1. An **Offset** of 0 places the **GradientStop** at the start of the gradient axis, in other words near its **StartPoint**. An **Offset** of 1 places the **GradientStop** at the **EndPoint**. At a minimum, a useful **LinearGradientBrush** should have two **GradientStop** values, where each **GradientStop** should specify a different **Color** and have a different **Offset** between 0 and 1.

This example creates a linear gradient with four colors and uses it to paint a Rectangle.

XML

```
<!-- This rectangle is painted with a diagonal linear gradient. -->
<Rectangle Width="200" Height="100">
    <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
            <GradientStop Color="Yellow" Offset="0.0"
x:Name="GradientStop1"/>
            <GradientStop Color="Red" Offset="0.25" x:Name="GradientStop2"/>
            <GradientStop Color="Blue" Offset="0.75"
x:Name="GradientStop3"/>
            <GradientStop Color="LimeGreen" Offset="1.0"
x:Name="GradientStop4"/>
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

The color of each point between gradient stops is linearly interpolated as a combination of the color specified by the two bounding gradient stops. The following image highlights the gradient stops in the previous example. The circles mark the position of the gradient stops, and the dashed line shows the gradient axis.



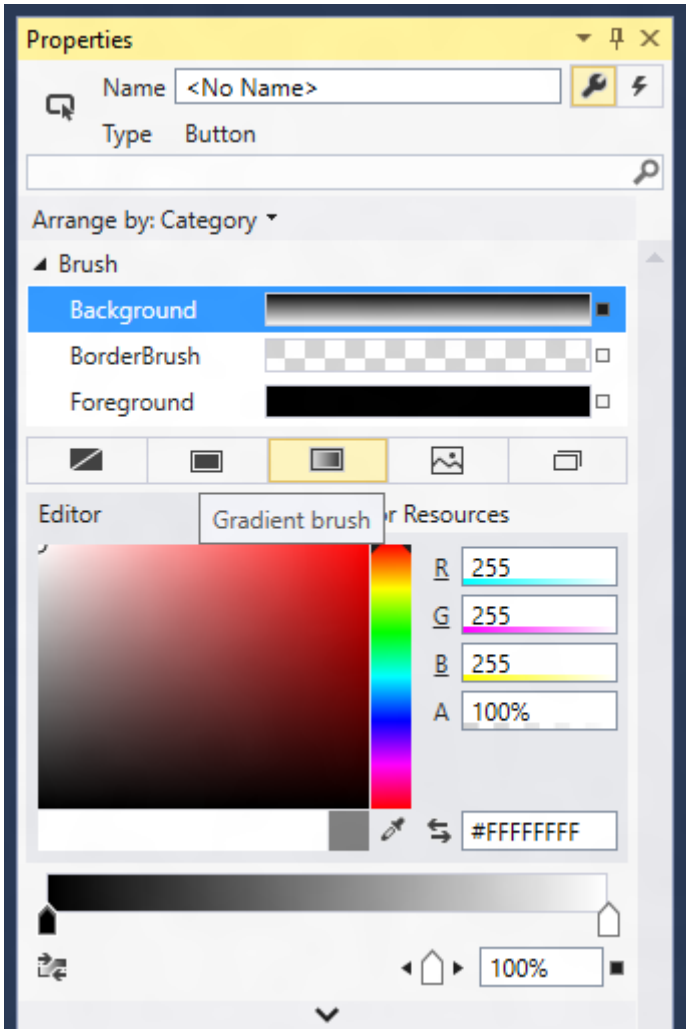
Combination of colors specified by the two bounding gradient stops

You can change the line at which the gradient stops are positioned by setting the StartPoint and EndPoint properties to be different values than the `(0,0)` and `(1,1)` starting defaults. By changing the **StartPoint** and **EndPoint** coordinate values, you can create horizontal or vertical gradients, reverse the gradient direction, or condense the gradient spread to apply to a smaller range than the full painted area. To condense the gradient, you set values of **StartPoint** and/or **EndPoint** to be something that is between the values 0 and 1. For example, if you want a horizontal gradient where the fade all

happens on the left half of the brush and the right side is solid to your last GradientStop color, you specify a **StartPoint** of `(0,0)` and an **EndPoint** of `(0.5,0)`.

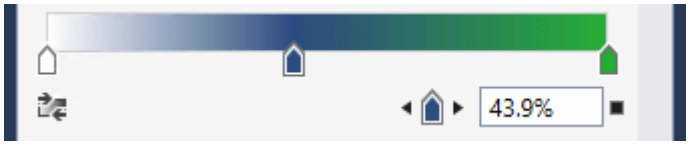
Use tools to make gradients

Now that you know how linear gradients work, you can use Visual Studio or Blend to make creating these gradients easier. To create a gradient, select the object you want to apply a gradient to on the design surface or in XAML view. Expand **Brush** and select the **Linear Gradient** tab.



Creating a linear gradient in Visual Studio

Now you can change the colors of the gradient stops and slide their positions using the bar on the bottom. You can also add new gradient stops by clicking on the bar and remove them by dragging the stops off of the bar (see next screenshot).



Gradient setting slider

Radial gradient brushes

A **RadialGradientBrush** is drawn within an ellipse that is defined by the **Center**, **RadiusX**, and **RadiusY** properties. Colors for the gradient start at the center of the ellipse and end at the radius.

The colors for the radial gradient are defined by color stops added to the **GradientStops** collection property. Each gradient stop specifies a color and an offset along the gradient.

The gradient origin defaults to center and can be offset using the **GradientOrigin** property.

MappingMode defines whether **Center**, **RadiusX**, **RadiusY**, and **GradientOrigin** represent relative or absolute coordinates.

When **MappingMode** is set to `RelativeToBoundingBox`, the X and Y values of the three properties are treated as relative to the element bounds, where `(0,0)` represents the top left and `(1,1)` represents the bottom right of the element bounds for the **Center**, **RadiusX**, and **RadiusY** properties and `(0,0)` represents the center for the **GradientOrigin** property.

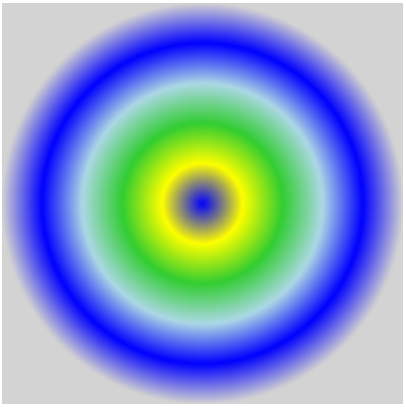
When **MappingMode** is set to `Absolute`, the X and Y values of the three properties are treated as absolute coordinates within the element bounds.

This example creates a linear gradient with four colors and uses it to paint a **Rectangle**.

XML

```
<!-- This rectangle is painted with a radial gradient. -->
<Rectangle Width="200" Height="200">
    <Rectangle.Fill>
        <media:RadialGradientBrush>
            <GradientStop Color="Blue" Offset="0.0" />
            <GradientStop Color="Yellow" Offset="0.2" />
            <GradientStop Color="LimeGreen" Offset="0.4" />
            <GradientStop Color="LightBlue" Offset="0.6" />
            <GradientStop Color="Blue" Offset="0.8" />
            <GradientStop Color="LightGray" Offset="1" />
        </media:RadialGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

The color of each point between gradient stops is radially interpolated as a combination of the color specified by the two bounding gradient stops. The following image highlights the gradient stops in the previous example.



Gradient stops

Image brushes

An **ImageBrush** paints an area with an image, with the image to paint coming from an image file source. You set the **ImageSource** property with the path of the image to load. Typically, the image source comes from a **Content** item that is part of your app's resources.

By default, an **ImageBrush** stretches its image to completely fill the painted area, possibly distorting the image if the painted area has a different aspect ratio than the image. You can change this behavior by changing the **Stretch** property from its default value of **Fill** and setting it as **None**, **Uniform**, or **UniformToFill**.

The next example creates an **ImageBrush** and sets the **ImageSource** to an image named licorice.jpg, which you must include as a resource in the app. The **ImageBrush** then paints the area defined by an **Ellipse** shape.

XML

```
<Ellipse Height="200" Width="300">
   <Ellipse.Fill>
     <ImageBrush ImageSource="licorice.jpg" />
   </Ellipse.Fill>
</Ellipse>
```

A rendered ImageBrush

ImageBrush and **Image** both reference an image source file by Uniform Resource Identifier (URI), where that image source file uses several possible image formats. These image source files are specified as URIs. For more info about specifying image sources, the usable image formats, and packaging them in an app, see Image and ImageBrush.

Brushes and text

You can also use brushes to apply rendering characteristics to text elements. For example, the **Foreground** property of **TextBlock** takes a **Brush**. You can apply any of the brushes described here to text. However, be careful with brushes applied to text, as any background might make the text can be unreadable if you use brushes that bleed into the background. Use **SolidColorBrush** for readability of text elements in most cases, unless you want the text element to be mostly decorative.

Even when you use a solid color, make sure that the text color you choose has enough contrast against the background color of the text's layout container. The level of contrast between text foreground and text container background is an accessibility consideration.

WebViewBrush

A **WebViewBrush** is a special type of brush that can access the content normally viewed in a **WebView** control. Instead of rendering the content in the rectangular **WebView** control area, **WebViewBrush** paints that content onto another element that has a **Brush**-type property for a render surface. **WebViewBrush** isn't appropriate for every brush scenario, but is useful for transitions of a **WebView**. For more info, see **WebViewBrush**.

XamlCompositionBrushBase

XamlCompositionBrushBase is a base class used to create custom brushes that use CompositionBrush to paint XAML UI elements.

This enables "drop down" interoperation between the Windows.UI.Xaml and Windows.UI.Composition layers as described in the Visual Layer overview.

To create a custom brush, create a new class that inherits from XamlCompositionBrushBase and implements the required methods.

For example, this can be used to apply effects to XAML UIElements using a CompositionEffectBrush, such as a **GaussianBlurEffect** or a SceneLightingEffect that controls the reflective properties of a XAML UIElement when being lit by a XamlLight.

For code examples, see XamlCompositionBrushBase.

Brushes as XAML resources

You can declare any brush to be a keyed XAML resource in a XAML resource dictionary. This makes it easy to replicate the same brush values as applied to multiple elements in a UI. The brush values are then shared and applied to any case where you reference the brush resource as a {StaticResource} usage in your XAML. This includes cases where you have a XAML control template that references the shared brush, and the control template is itself a keyed XAML resource.

Brushes in code

It's much more typical to specify brushes using XAML than it is to use code to define brushes. This is because brushes are usually defined as XAML resources, and because brush values are often the output of design tools or otherwise as part of a XAML UI definition. Still, for the occasional case where you might want to define a brush using code, all the Brush types are available for code instantiation.

To create a SolidColorBrush in code, use the constructor that takes a Color parameter. Pass a value that is a static property of the Colors class, like this:

cs

```
SolidColorBrush blueBrush = new SolidColorBrush(Windows.UI.Colors.Blue);
```

VB

```
Dim blueBrush as SolidColorBrush = New
```

```
SolidColorBrush(Windows.UI.Colors.Blue)
```

C++/WinRT

```
Windows::UI::Xaml::Media::SolidColorBrush blueBrush{
Windows::UI::Colors::Blue() };
```

C++

```
blueBrush = ref new SolidColorBrush(Windows::UI::Colors::Blue);
```

For **WebViewBrush** and **ImageBrush**, use the default constructor and then call other APIs before you attempt to use that brush for a UI property.

- **ImageSource** requires a **BitmapImage** (not a URI) when you define an **ImageBrush** using code. If your source is a stream , use the **SetSourceAsync** method to initialize the value. If your source is a URI, which includes content in your app that uses the **ms-appx** or **ms-resource** schemes, use the **BitmapImage** constructor that takes a URI. You might also consider handling the **ImageOpened** event if there are any timing issues with retrieving or decoding the image source, where you might need alternate content to display until the image source is available.
- For **WebViewBrush** you might need to call **Redraw** if you've recently reset the **SourceName** property or if the content of the **WebView** is also being changed with code.

For code examples, see **WebViewBrush**, **ImageBrush**, and **XamlCompositionBrushBase**.

XAML styles

Article • 11/26/2024

You can customize the appearance of your apps in many ways by using the XAML framework. Styles let you set control properties and reuse those settings for a consistent appearance across multiple controls.

WinUI and styles

Starting with WinUI 2.2, we have used WinUI to deliver new visual style updates across our UI components. If you notice your UI is not updating to the latest styles, be sure to update to the latest WinUI NuGet package.

Starting with WinUI 2.6, we provide new styles for most of the controls, and a new versioning system that let's you revert to the previous control styles if needed. We encourage you to use the new styles, as they better match the design direction of Windows. However, if your scenario cannot support the new styles, the previous versions are still available.

You can change the style version by setting the `ControlsResourcesVersion` property on the `XamlControlsResources` that you include in your `Application.Resources` when you use WinUI version 2. `ControlsResourcesVersion` defaults to the enum value `Version2`.

Setting this value to `Version1` causes `XamlControlsResources` to load the previous style versions instead of the new styles used by the latest WinUI version. Changing this property at runtime is not supported and VisualStudio's hot reload functionality will not work; however, after you rebuild your application you will see the control styles change.

XAML

```
<Application.Resources>
    <XamlControlsResources xmlns="using:Microsoft.UI.Xaml.Controls"
                           ControlsResourcesVersion="Version1"/>
</Application.Resources>
```

Style basics

Use styles to extract visual property settings into reusable resources. Here's an example that shows 3 buttons with a style that sets the BorderBrush, BorderThickness and

Foreground properties. By applying a style, you can make the controls appear the same without having to set these properties on each control separately.

You can define a style inline in the XAML for a control, or as a reusable resource. Define resources in an individual page's XAML file, in the App.xaml file, or in a separate resource dictionary XAML file. A resource dictionary XAML file can be shared across apps, and more than one resource dictionary can be merged in a single app. Where the resource is defined determines the scope in which it can be used. Page-level resources are available only in the page where they are defined. If resources with the same key are defined in both App.xaml and in a page, the resource in the page overrides the resource in App.xaml. If a resource is defined in a separate resource dictionary file, its scope is determined by where the resource dictionary is referenced.

In the Style definition, you need a TargetType attribute and a collection of one or more Setter elements. The **TargetType** attribute is a string that specifies a FrameworkElement type to apply the style to. The **TargetType** value must specify a **FrameworkElement**-derived type that's defined by the Windows Runtime or a custom type that's available in a referenced assembly. If you try to apply a style to a control and the control's type doesn't match the **TargetType** attribute of the style you're trying to apply, an exception occurs.

Each Setter element requires a Property and a Value. These property settings indicate what control property the setting applies to, and the value to set for that property. You can set the **Setter.Value** with either attribute or property element syntax. The XAML here shows the style applied to the buttons shown previously. In this XAML, the first two **Setter** elements use attribute syntax, but the last **Setter**, for the BorderBrush property, uses property element syntax. The example doesn't use the x:Key attribute attribute, so the style is implicitly applied to the buttons. Applying styles implicitly or explicitly is explained in the next section.

XAML

```
<Page.Resources>
    <Style TargetType="Button">
        <Setter Property="BorderThickness" Value="5" />
        <Setter Property="Foreground" Value="Black" />
        <Setter Property="BorderBrush" >
            <Setter.Value>
                <LinearGradientBrush StartPoint="0.5,0" EndPoint="0.5,1">
                    <GradientStop Color="Yellow" Offset="0.0" />
                    <GradientStop Color="Red" Offset="0.25" />
                    <GradientStop Color="Blue" Offset="0.75" />
```

```
                    <GradientStop Color="LimeGreen" Offset="1.0" />
                </LinearGradientBrush>
            </Setter.Value>
        </Setter>
    </Style>
</Page.Resources>

<StackPanel Orientation="Horizontal">
    <Button Content="Button"/>
    <Button Content="Button"/>
    <Button Content="Button"/>
</StackPanel>
```

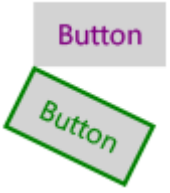
Apply an implicit or explicit style

If you define a style as a resource, there are two ways to apply it to your controls:

- Implicitly, by specifying only a TargetType for the Style.
- Explicitly, by specifying a TargetType and an x:Key attribute attribute for the Style and then by setting the target control's Style property with a {StaticResource} markup extension reference that uses the explicit key.

If a style contains the x:Key attribute, you can only apply it to a control by setting the Style property of the control to the keyed style. In contrast, a style without an x:Key attribute is automatically applied to every control of its target type, that doesn't otherwise have an explicit style setting.

Here are two buttons that demonstrate implicit and explicit styles.



In this example, the first style has an x:Key attribute and its target type is Button. The first button's Style property is set to this key, so this style is applied explicitly. The second style is applied implicitly to the second button because its target type is **Button** and the style doesn't have an x:Key attribute.

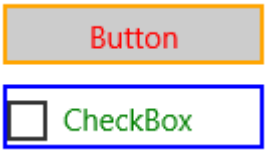
XAML

```
<Page.Resources>
    <Style x:Key="PurpleStyle" TargetType="Button">
        <Setter Property="FontFamily" Value="Segoe UI"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="Foreground" Value="Purple"/>
```

```
    </Style>

    <Style TargetType="Button">
        <Setter Property="FontFamily" Value="Segoe UI"/>
        <Setter Property="FontSize" Value="14"/>
        <Setter Property="RenderTransform">
            <Setter.Value>
                <RotateTransform Angle="25"/>
            </Setter.Value>
        </Setter>
        <Setter Property="BorderBrush" Value="Green"/>
        <Setter Property="BorderThickness" Value="2"/>
        <Setter Property="Foreground" Value="Green"/>
    </Style>
</Page.Resources>

<Grid x:Name="LayoutRoot">
    <Button Content="Button" Style="{StaticResource PurpleStyle}"/>
    <Button Content="Button"/>
</Grid>
```

Use based-on styles

To make styles easier to maintain and to optimize style reuse, you can create styles that inherit from other styles. You use the BasedOn property to create inherited styles. Styles that inherit from other styles must target the same type of control or a control that derives from the type targeted by the base style. For example, if a base style targets ContentControl, styles that are based on this style can target **ContentControl** or types that derive from **ContentControl** such as Button and ScrollViewer. If a value is not set in the based-on style, it's inherited from the base style. To change a value from the base style, the based-on style overrides that value. The next example shows a **Button** and a CheckBox with styles that inherit from the same base style.



The base style targets ContentControl, and sets the Height, and Width properties. The styles based on this style target CheckBox and Button, which derive from **ContentControl**. The based-on styles set different colors for the BorderBrush and Foreground properties. (You don't typically put a border around a **CheckBox**. We do it here to show the effects of the style.)

XAML

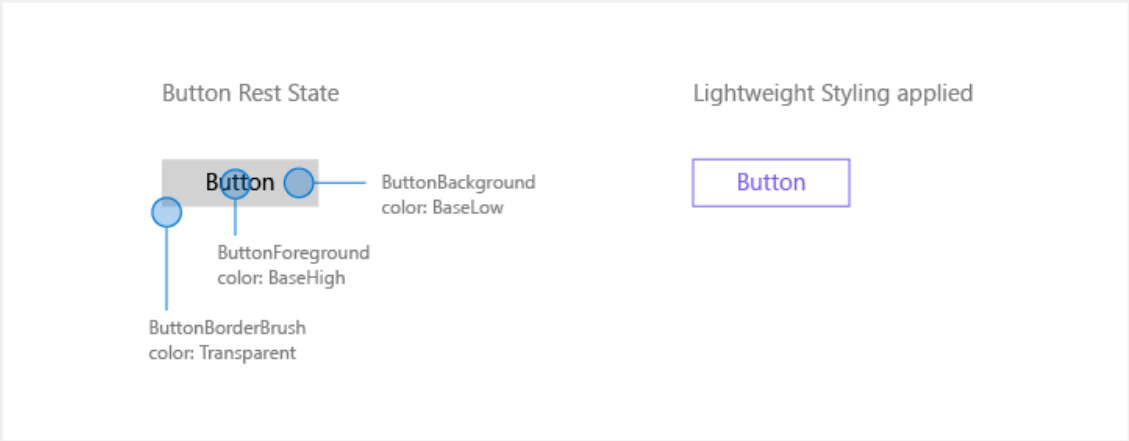
```
<Page.Resources>
    <Style x:Key="BasicStyle" TargetType="ContentControl">
        <Setter Property="Width" Value="130" />
        <Setter Property="Height" Value="30" />
    </Style>

    <Style x:Key="ButtonStyle" TargetType="Button"
           BasedOn="{StaticResource BasicStyle}">
        <Setter Property="BorderBrush" Value="Orange" />
        <Setter Property="BorderThickness" Value="2" />
        <Setter Property="Foreground" Value="Red" />
    </Style>

    <Style x:Key="CheckBoxStyle" TargetType="CheckBox"
           BasedOn="{StaticResource BasicStyle}">
        <Setter Property="BorderBrush" Value="Blue" />
        <Setter Property="BorderThickness" Value="2" />
        <Setter Property="Foreground" Value="Green" />
    </Style>
</Page.Resources>

<StackPanel>
    <Button Content="Button" Style="{StaticResource ButtonStyle}"
Margin="0,10"/>
    <CheckBox Content="CheckBox" Style="{StaticResource CheckBoxStyle}"/>
</StackPanel>
```

Use tools to work with styles easily

A fast way to apply styles to your controls is to right-click on a control on the Microsoft Visual Studio XAML design surface and select **Edit Style** or **Edit Template** (depending on the control you are right-clicking on). You can then apply an existing style by selecting **Apply Resource** or define a new style by selecting **Create Empty**. If you create an empty style, you are given the option to define it in the page, in the App.xaml file, or in a separate resource dictionary.

Lightweight styling

Overriding the system brushes is generally done at the App or Page level, and in either case the color override will affect all controls that reference that brush – and in XAML many controls can reference the same system brush.



XAML

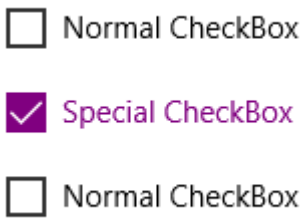
```
<Page.Resources>
    <ResourceDictionary>
        <ResourceDictionary.ThemeDictionaries>
            <ResourceDictionary x:Key="Light">
                <SolidColorBrush x:Key="ButtonBackground"
Color="Transparent"/>
                <SolidColorBrush x:Key="ButtonForeground"
Color="MediumSlateBlue"/>
                <SolidColorBrush x:Key="ButtonBorderBrush"
Color="MediumSlateBlue"/>
            </ResourceDictionary>
        </ResourceDictionary.ThemeDictionaries>
    </ResourceDictionary>
</Page.Resources>
```

For states like PointerOver (mouse is hovered over the button), **PointerPressed** (button has been invoked), or Disabled (button is not interactable). These endings are appended onto the original Lightweight styling names: **ButtonBackgroundPointerOver**, **ButtonForegroundPressed**, **ButtonBorderBrushDisabled**, etc. Modifying those brushes as well, will make sure that your controls are colored consistently to your app's theme.

Placing these brush overrides at the **App.Resources** level, will alter all the buttons within the entire app, instead of on a single page.

Per-control styling

In other cases, changing a single control on one page only to look a certain way, without altering any other versions of that control, is desired:



XAML

```
<CheckBox Content="Normal CheckBox" Margin="5"/>
<CheckBox Content="Special CheckBox" Margin="5">
    <CheckBox.Resources>
        <ResourceDictionary>
            <ResourceDictionary.ThemeDictionaries>
                <ResourceDictionary x:Key="Light">
                    <SolidColorBrush x:Key="CheckBoxForegroundUnchecked"
                        Color="Purple"/>
                    <SolidColorBrush x:Key="CheckBoxForegroundChecked"
                        Color="Purple"/>
                    <SolidColorBrush
x:Key="CheckBoxCheckGlyphForegroundChecked"
                        Color="White"/>
                    <SolidColorBrush
x:Key="CheckBoxCheckBackgroundStrokeChecked"
                        Color="Purple"/>
                    <SolidColorBrush
x:Key="CheckBoxCheckBackgroundFillChecked"
                        Color="Purple"/>
                </ResourceDictionary>
            </ResourceDictionary.ThemeDictionaries>
        </ResourceDictionary>
    </CheckBox.Resources>
</CheckBox>
<CheckBox Content="Normal CheckBox" Margin="5"/>
```

This would only effect that one "Special CheckBox" on the page where that control existed.

Custom controls

When you are building your own custom controls that may be visually and/or functionally aligned with our built-in controls, consider using implicit styling and Lightweight styling resources to define your custom content. You can either use the resources directly, or create a new alias for the resource.

Using control resources directly

For example, if you are writing a control that looks like a Button, you can have your control reference the button resources directly, like this:

XAML

```
<Style TargetType="local:MyCustomControl">
  <Setter Property="Background" Value="{ThemeResource ButtonBackground}" />
  <Setter Property="BorderBrush" Value="{ThemeResource ButtonBorderBrush}"
/>
</Style>
```

Aliasing control resources to new names

Alternatively, if you prefer to make your own resources, you should alias those custom names to our default Lightweight styling resources.

For example, your custom control's style might have special resource definitions:

XAML

```
<Style TargetType="local:MyCustomControl">
  <Setter Property="Background" Value="{ThemeResource
MyCustomControlBackground}" />
  <Setter Property="BorderBrush" Value="{ThemeResource
MyCustomControlBorderBrush}"/>
</Style>
```

In your Resource Dictionary or main definition, you would hook up the Lightweight styling resources to your custom ones:

XAML

```
<ResourceDictionary.ThemeDictionaries>
    <ResourceDictionary x:Key="Default">
        <StaticResource x:Key="MyCustomControlBackground"
ResourceKey="ButtonBackground" />
        <StaticResource x:Key="MyCustomControlBorderBrush"
ResourceKey="ButtonBorderBrush" />
    </ResourceDictionary>
    <ResourceDictionary x:Key="Light">
        <StaticResource x:Key="MyCustomControlBackground"
ResourceKey="ButtonBackground" />
        <StaticResource x:Key="MyCustomControlBorderBrush"
ResourceKey="ButtonBorderBrush" />
    </ResourceDictionary>
    <ResourceDictionary x:Key="HighContrast">
        <StaticResource x:Key="MyCustomControlBackground"
ResourceKey="ButtonBackground" />
```

```
        <StaticResource x:Key="MyCustomControlBorderBrush"
ResourceKey="ButtonBorderBrush" />
    </ResourceDictionary>
</ResourceDictionary.ThemeDictionaries>
```

Its required that you use a `ThemeDictionary` that is duplicated three times in order to handle the three different theme changes properly (`Default`, `Light`, `HighContrast`).

> **⊗ Caution**
>
> If you assign a Lightweight styling resource to a new alias, and also redefine the Lightweight styling resource, your customization might not be applied if the resource lookup is not in the correct order. For example, if you override `ButtonBackground` in a spot that is searched before `MyCustomControlBackground` is found, the override would be missed.

Avoid restyling controls

The WinUI 2.2 or later includes new styles and templates for both WinUI and system controls.

The best way to stay current with our latest visual styles is to use the latest WinUI 2 package and avoid custom styles and templates (also known as re-templating). Styles are still a convenient way to apply a set of values consistently across controls in your app. When doing this, make sure to be based on our latest styles.

For system controls that use WinUI styles (`Windows.UI.Xaml.Controls` namespace), set `BasedOn="{StaticResource Default<ControlName>Style}"`, where `<ControlName>` is the name of the control. For example:

XAML

```
<Style TargetType="TextBox" BasedOn="{StaticResource DefaultTextBoxStyle}">
    <Setter Property="Foreground" Value="Blue"/>
</Style>
```

For WinUI 2 controls (`Microsoft.UI.Xaml.Controls` namespace), the default style is defined in the metadata, so omit `BasedOn`.

Derived controls

If you derive a custom control from an existing XAML control, it will not get the WinUI 2 styles by default. To apply the WinUI 2 styles:

- Create a new Style with its TargetType set to your custom control.
- Base the Style on the default style of the control you derived from.

One common scenario for this is to derive a new control from ContentDialog. This example shows how to create a new Style that applies `DefaultContentDialogStyle` to your custom dialog.

XAML

```
<ContentDialog
    x:Class="ExampleApp.SignInContentDialog"
    ... >

    <ContentDialog.Resources>
        <Style TargetType="local:SignInContentDialog" BasedOn="
{StaticResource DefaultContentDialogStyle}"/>
        ...
    </ContentDialog.Resources>
    <!-- CONTENT -->
</ContentDialog>
```

The template property

A style setter can be used for the Template property of a Control, and in fact this makes up the majority of a typical XAML style and an app's XAML resources. This is discussed in more detail in the topic Control templates.

Control templates

Article • 11/26/2024

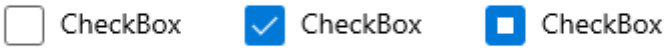
You can customize a control's visual structure and visual behavior by creating a control template in the XAML framework. Controls have many properties, such as **Background**, **Foreground**, and **FontFamily**, that you can set to specify different aspects of the control's appearance. But the changes that you can make by setting these properties are limited. You can specify additional customizations by creating a template using the **ControlTemplate** class. Here, we show you how to create a **ControlTemplate** to customize the appearance of a **CheckBox** control.

> **Important APIs**: **ControlTemplate class**, **Control.Template property**

Custom control template example

By default, a **CheckBox** control puts its content (the string or object next to the **CheckBox**) to the right of the selection box, and a check mark indicates that a user selected the **CheckBox**. These characteristics represent the visual structure and visual behavior of the **CheckBox**.

Here's a **CheckBox** using the default **ControlTemplate** shown in the `Unchecked`, `Checked`, and `Indeterminate` states.



You can change these characteristics by creating a **ControlTemplate** for the **CheckBox**. For example, if you want the content of the check box to be below the selection box, and you want to use an **X** to indicate that a user selected the check box. You specify these characteristics in the **ControlTemplate** of the **CheckBox**.

To use a custom template with a control, assign the **ControlTemplate** to the **Template** property of the control. Here's a **CheckBox** using a **ControlTemplate** called `CheckBoxTemplate1`. We show the Extensible Application Markup Language (XAML) for the **ControlTemplate** in the next section.

```
XAML
```

```
<CheckBox Content="CheckBox" Template="{StaticResource CheckBoxTemplate1}"
IsThreeState="True" Margin="20"/>
```

Here's how this **CheckBox** looks in the `Unchecked`, `Checked`, and `Indeterminate` states after we apply our template.

Specify the visual structure of a control

When you create a **ControlTemplate**, you combine **FrameworkElement** objects to build a single control. A **ControlTemplate** must have only one **FrameworkElement** as its root element. The root element usually contains other **FrameworkElement** objects. The combination of objects makes up the control's visual structure.

This XAML creates a **ControlTemplate** for a **CheckBox** that specifies that the content of the control is below the selection box. The root element is a **Border**. The example specifies a **Path** to create an **X** that indicates that a user selected the **CheckBox**, and an **Ellipse** that indicates an indeterminate state. Note that the **Opacity** is set to 0 on the **Path** and the **Ellipse** so that by default, neither appear.

A TemplateBinding is a special binding that links the value of a property in a control template to the value of some other exposed property on the templated control. TemplateBinding can only be used within a ControlTemplate definition in XAML. See TemplateBinding markup extension for more info.

> **Note**
>
> Starting with Windows 10, version 1809 (SDK 17763), you can use x:Bind markup extensions in places you use TemplateBinding. See TemplateBinding markup extension for more info.

XAML

```
<ControlTemplate x:Key="CheckBoxTemplate1" TargetType="CheckBox">
    <Border BorderBrush="{TemplateBinding BorderBrush}"
            BorderThickness="{TemplateBinding BorderThickness}"
```

```
            Background="{TemplateBinding Background}">
        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="*"/>
                <RowDefinition Height="25"/>
            </Grid.RowDefinitions>
            <Rectangle x:Name="NormalRectangle" Fill="Transparent"
Height="20" Width="20"
                       Stroke="{ThemeResource
SystemControlForegroundBaseMediumHighBrush}"
                       StrokeThickness="{ThemeResource
CheckBoxBorderThemeThickness}"
                       UseLayoutRounding="False"/>
            <!-- Create an X to indicate that the CheckBox is selected. -->
            <Path x:Name="CheckGlyph"
                  Data="M103,240 L111,240 119,248 127,240 135,240 123,252
135,264 127,264 119,257 111,264 103,264 114,252 z"
                  Fill="{ThemeResource CheckBoxForegroundThemeBrush}"
                  FlowDirection="LeftToRight"
                  Height="14" Width="16" Opacity="0" Stretch="Fill"/>
            <Ellipse x:Name="IndeterminateGlyph"
                     Fill="{ThemeResource CheckBoxForegroundThemeBrush}"
                     Height="8" Width="8" Opacity="0"
UseLayoutRounding="False" />
            <ContentPresenter x:Name="ContentPresenter"
                              ContentTemplate="{TemplateBinding
ContentTemplate}"
                              Content="{TemplateBinding Content}"
                              Margin="{TemplateBinding Padding}"
Grid.Row="1"
                              HorizontalAlignment="Center"
                              VerticalAlignment="{TemplateBinding
VerticalContentAlignment}"/>
        </Grid>
    </Border>
</ControlTemplate>
```

Specify the visual behavior of a control

A visual behavior specifies the appearance of a control when it is in a certain state. The **CheckBox** control has 3 check states: `Checked`, `Unchecked`, and `Indeterminate`. The value of the **IsChecked** property determines the state of the **CheckBox**, and its state determines what appears in the box.

This table lists the possible values of **IsChecked**, the corresponding states of the **CheckBox**, and the appearance of the **CheckBox**.

Expand table

IsChecked value	CheckBox state	CheckBox appearance
true	`Checked`	Contains an "X".
false	`Unchecked`	Empty.
null	`Indeterminate`	Contains a circle.

You specify the appearance of a control when it is in a certain state by using VisualState objects. A **VisualState** contains a Setter or Storyboard that changes the appearance of the elements in the ControlTemplate. When the control enters the state that the VisualState.Name property specifies, the property changes in the **Setter** or Storyboard are applied. When the control exits the state, the changes are removed. You add **VisualState** objects to VisualStateGroup objects. You add **VisualStateGroup** objects to the VisualStateManager.VisualStateGroups attached property, which you set on the root FrameworkElement of the **ControlTemplate**.

This XAML shows the VisualState objects for the `Checked`, `Unchecked`, and `Indeterminate` states. The example sets the VisualStateManager.VisualStateGroups attached property on the Border, which is the root element of the ControlTemplate. The `Checked` **VisualState** specifies that the Opacity of the Path named `CheckGlyph` (which we show in the previous example) is 1. The `Indeterminate` **VisualState** specifies that the **Opacity** of the Ellipse named `IndeterminateGlyph` is 1. The `Unchecked` **VisualState** has no Setter or Storyboard, so the CheckBox returns to its default appearance.

XAML

```
<ControlTemplate x:Key="CheckBoxTemplate1" TargetType="CheckBox">
    <Border BorderBrush="{TemplateBinding BorderBrush}"
            BorderThickness="{TemplateBinding BorderThickness}"
            Background="{TemplateBinding Background}">

        <VisualStateManager.VisualStateGroups>
            <VisualStateGroup x:Name="CheckStates">
                <VisualState x:Name="Checked">
                    <VisualState.Setters>
                        <Setter Target="CheckGlyph.Opacity" Value="1"/>
                    </VisualState.Setters>
                    <!-- This Storyboard is equivalent to the Setter. -->
                    <!--<Storyboard>
                        <DoubleAnimation Duration="0" To="1"
                         Storyboard.TargetName="CheckGlyph"
Storyboard.TargetProperty="Opacity"/>
                    </Storyboard>-->
                </VisualState>
                <VisualState x:Name="Unchecked"/>
                <VisualState x:Name="Indeterminate">
                    <VisualState.Setters>
```

```
                        <Setter Target="IndeterminateGlyph.Opacity"
Value="1"/>
                    </VisualState.Setters>
                    <!-- This Storyboard is equivalent to the Setter. -->
                    <!--<Storyboard>
                        <DoubleAnimation Duration="0" To="1"
                         Storyboard.TargetName="IndeterminateGlyph"
Storyboard.TargetProperty="Opacity"/>
                    </Storyboard>-->
                </VisualState>
            </VisualStateGroup>
        </VisualStateManager.VisualStateGroups>

        <Grid>
            <Grid.RowDefinitions>
                <RowDefinition Height="*"/>
                <RowDefinition Height="25"/>
            </Grid.RowDefinitions>
            <Rectangle x:Name="NormalRectangle" Fill="Transparent"
Height="20" Width="20"
                       Stroke="{ThemeResource
SystemControlForegroundBaseMediumHighBrush}"
                       StrokeThickness="{ThemeResource
CheckBoxBorderThemeThickness}"
                       UseLayoutRounding="False"/>
            <!-- Create an X to indicate that the CheckBox is selected. -->
            <Path x:Name="CheckGlyph"
                  Data="M103,240 L111,240 119,248 127,240 135,240 123,252
135,264 127,264 119,257 111,264 103,264 114,252 z"
                  Fill="{ThemeResource CheckBoxForegroundThemeBrush}"
                  FlowDirection="LeftToRight"
                  Height="14" Width="16" Opacity="0" Stretch="Fill"/>
            <Ellipse x:Name="IndeterminateGlyph"
                     Fill="{ThemeResource CheckBoxForegroundThemeBrush}"
                     Height="8" Width="8" Opacity="0"
UseLayoutRounding="False" />
            <ContentPresenter x:Name="ContentPresenter"
                              ContentTemplate="{TemplateBinding
ContentTemplate}"
                              Content="{TemplateBinding Content}"
                              Margin="{TemplateBinding Padding}"
Grid.Row="1"
                              HorizontalAlignment="Center"
                              VerticalAlignment="{TemplateBinding
VerticalContentAlignment}"/>
        </Grid>
    </Border>
</ControlTemplate>
```

To better understand how **VisualState** objects work, consider what happens when the **CheckBox** goes from the `Unchecked` state to the `Checked` state, then to the `Indeterminate` state, and then back to the `Unchecked` state. Here are the transitions.

Expand table

State transition	What happens	CheckBox appearance when the transition completes
From `Unchecked` to `Checked`.	The **Setter** value of the `Checked` **VisualState** is applied, so the **Opacity** of `CheckGlyph` is 1.	An X is displayed.
From `Checked` to `Indeterminate`.	The **Setter** value of the `Indeterminate` **VisualState** is applied, so the **Opacity** of `IndeterminateGlyph` is 1. The **Setter** value of the `Checked` **VisualState** is removed, so the **Opacity** of `CheckGlyph` is 0.	A circle is displayed.
From `Indeterminate` to `Unchecked`.	The **Setter** value of the `Indeterminate` **VisualState** is removed, so the **Opacity** of `IndeterminateGlyph` is 0.	Nothing is displayed.

For more info about how to create visual states for controls, and in particular how to use the **Storyboard** class and the animation types, see Storyboarded animations for visual states.

Use tools to work with themes easily

A fast way to apply themes to your controls is to right-click on a control on the Microsoft Visual Studio **Document Outline** and select **Edit Theme** or **Edit Style** (depending on the control you are right-clicking on). You can then apply an existing theme by selecting **Apply Resource** or define a new one by selecting **Create Empty**.

Controls and accessibility

When you create a new template for a control, in addition to possibly changing the control's behavior and visual appearance, you might also be changing how the control represents itself to accessibility frameworks. The Windows app supports the Microsoft UI Automation framework for accessibility. All of the default controls and their templates have support for common UI Automation control types and patterns that are appropriate for the control's purpose and function. These control types and patterns are interpreted by UI Automation clients such as assistive technologies, and this enables a control to be accessible as a part of a larger accessible app UI.

To separate the basic control logic and also to satisfy some of the architectural requirements of UI Automation, control classes include their accessibility support in a

separate class, an automation peer. The automation peers sometimes have interactions with the control templates because the peers expect certain named parts to exist in the templates, so that functionality such as enabling assistive technologies to invoke actions of buttons is possible.

When you create a completely new custom control, you sometimes also will want to create a new automation peer to go along with it. For more info, see Custom automation peers.

Learn more about a control's default template

The topics that document the styles and templates for XAML controls show you excerpts of the same starting XAML you'd see if you used the **Edit Theme** or **Edit Style** techniques explained previously. Each topic lists the names of the visual states, the theme resources used, and the full XAML for the style that contains the template. The topics can be useful guidance if you've already started modifying a template and want to see what the original template looked like, or to verify that your new template has all of the required named visual states.

Theme resources in control templates

For some of the attributes in the XAML examples, you may have noticed resource references that use the {ThemeResource} markup extension. This is a technique that enables a single control template to use resources that can be different values depending on which theme is currently active. This is particularly important for brushes and colors, because the main purpose of the themes is to enable users to choose whether they want a dark, light, or high contrast theme applied to the system overall. Apps that use the XAML resource system can use a resource set that's appropriate for that theme, so that the theme choices in an app's UI are reflective of the user's systemwide theme choice.

Get the sample code

- WinUI Gallery sample
- Custom text edit control sample

ResourceDictionary and XAML resource references

Article • 11/26/2024

You can define the UI or resources for your app using XAML. Resources are typically definitions of some object that you expect to use more than once. To refer to a XAML resource later, you specify a key for a resource that acts like its name. You can reference a resource throughout an app or from any XAML page within it. You can define your resources using a ResourceDictionary element from the Windows Runtime XAML. Then, you can reference your resources by using a StaticResource markup extension or ThemeResource markup extension.

The XAML elements you might want to declare most often as XAML resources include Style, ControlTemplate, animation components, and Brush subclasses. Here, we explain how to define a ResourceDictionary and keyed resources, and how XAML resources relate to other resources that you define as part of your app or app package. We also explain resource dictionary advanced features such as MergedDictionaries and ThemeDictionaries.

Prerequisites

A solid understanding of XAML markup. We recommend reading XAML overview.

Define and use XAML resources

XAML resources are objects that are referenced from markup more than once. Resources are defined in a ResourceDictionary, typically in a separate file or at the top of the markup page, like this.

XAML

```
<Page
    x:Class="MSDNSample.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Page.Resources>
        <x:String x:Key="greeting">Hello world</x:String>
        <x:String x:Key="goodbye">Goodbye world</x:String>
    </Page.Resources>

    <TextBlock Text="{StaticResource greeting}" Foreground="Gray"
```

```
VerticalAlignment="Center"/>
</Page>
```

In this example:

- `<Page.Resources>…</Page.Resources>` - Defines the resource dictionary.
- `<x:String>` - Defines the resource with the key "greeting".
- `{StaticResource greeting}` - Looks up the resource with the key "greeting", which is assigned to the Text property of the TextBlock.

> **Note** Don't confuse the concepts related to ResourceDictionary with the **Resource** build action, resource (.resw) files, or other "resources" that are discussed in the context of structuring the code project that produces your app package.

Resources don't have to be strings; they can be any shareable object, such as styles, templates, brushes, and colors. However, controls, shapes, and other FrameworkElements are not shareable, so they can't be declared as reusable resources. For more info about sharing, see the XAML resources must be shareable section later in this topic.

Here, both a brush and a string are declared as resources and used by controls in a page.

XAML

```
<Page
    x:Class="SpiderMSDN.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Page.Resources>
        <SolidColorBrush x:Key="myFavoriteColor" Color="green"/>
        <x:String x:Key="greeting">Hello world</x:String>
    </Page.Resources>

    <TextBlock Foreground="{StaticResource myFavoriteColor}" Text="
{StaticResource greeting}" VerticalAlignment="Top"/>
    <Button Foreground="{StaticResource myFavoriteColor}" Content="
{StaticResource greeting}" VerticalAlignment="Center"/>
</Page>
```

All resources need to have a key. Usually that key is a string defined with `x:Key="myString"`. However, there are a few other ways to specify a key:

- Style and ControlTemplate require a **TargetType**, and will use the **TargetType** as the key if x:Key is not specified. In this case, the key is the actual Type object, not a

string. (See examples below)

- DataTemplate resources that have a **TargetType** will use the **TargetType** as the key if x:Key is not specified. In this case, the key is the actual Type object, not a string.
- x:Name can be used instead of x:Key. However, x:Name also generates a code behind field for the resource. As a result, x:Name is less efficient than x:Key because that field needs to be initialized when the page is loaded.

The StaticResource markup extension can retrieve resources only with a string name (x:Key or x:Name). However, the XAML framework also looks for implicit style resources (those which use **TargetType** rather than x:Key or x:Name) when it decides which style & template to use for a control that hasn't set the Style and ContentTemplate or ItemTemplate properties.

Here, the Style has an implicit key of **typeof(Button)**, and since the Button at the bottom of the page doesn't specify a Style property, it looks for a style with key of **typeof(Button)**:

XAML

```
<Page
    x:Class="MSDNSample.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Page.Resources>
        <Style TargetType="Button">
            <Setter Property="Background" Value="Red"/>
        </Style>
    </Page.Resources>
    <Grid>
       <!-- This button will have a red background. -->
       <Button Content="Button" Height="100" VerticalAlignment="Center"
Width="100"/>
    </Grid>
</Page>
```

For more info about implicit styles and how they work, see Styling controls and Control templates.

Look up resources in code

You access members of the resource dictionary like any other dictionary.

> ⚠ **Warning**

When you perform a resource lookup in code, only the resources in the `Page.Resources` dictionary are looked at. Unlike the StaticResource markup extension, the code doesn't fall back to the `Application.Resources` dictionary if the resources aren't found in the first dictionary.

This example shows how to retrieve the `redButtonStyle` resource out of a page's resource dictionary:

XAML

```
<Page
    x:Class="MSDNSample.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Page.Resources>
        <Style TargetType="Button" x:Key="redButtonStyle">
            <Setter Property="Background" Value="red"/>
        </Style>
    </Page.Resources>
</Page>
```

C#

```
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            Style redButtonStyle = (Style)this.Resources["redButtonStyle"];
        }
    }
```

To look up app-wide resources from code, use **Application.Current.Resources** to get the app's resource dictionary, as shown here.

XAML

```
<Application
    x:Class="MSDNSample.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:SpiderMSDN">
    <Application.Resources>
        <Style TargetType="Button" x:Key="appButtonStyle">
            <Setter Property="Background" Value="red"/>
```

```
        </Style>
    </Application.Resources>

</Application>
```

C#

```
    public sealed partial class MainPage : Page
    {
        public MainPage()
        {
            this.InitializeComponent();
            Style appButtonStyle =
(Style)Application.Current.Resources["appButtonStyle"];
        }
    }
```

You can also add an application resource in code.

There are two things to keep in mind when doing this.

- First, you need to add the resources before any page tries to use the resource.
- Second, you can't add resources in the App's constructor.

You can avoid both problems if you add the resource in the Application.OnLaunched method, like this.

C#

```
// App.xaml.cs

sealed partial class App : Application
{
    protected override void OnLaunched(LaunchActivatedEventArgs e)
    {
        Frame rootFrame = Window.Current.Content as Frame;
        if (rootFrame == null)
        {
            SolidColorBrush brush = new
SolidColorBrush(Windows.UI.Color.FromArgb(255, 0, 255, 0)); // green
            this.Resources["brush"] = brush;
            // … Other code that VS generates for you …
        }
    }
}
```

Every FrameworkElement can have a ResourceDictionary

FrameworkElement is a base class that controls inherit from, and it has a Resources property. So, you can add a local resource dictionary to any **FrameworkElement**.

Here, both the Page and the Border have resource dictionaries, and they both have a resource called "greeting". The TextBlock named 'textBlock2' is inside the **Border**, so its resource lookup looks first to the **Border**'s resources, then the **Page**'s resources, and then the Application resources. The **TextBlock** will read "Hola mundo".

To access that element's resources from code, use that element's Resources property. Accessing a FrameworkElement's resources in code, rather than XAML, will look only in that dictionary, not in parent element's dictionaries.

XAML

```
<Page
    x:Class="MSDNSample.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Page.Resources>
        <x:String x:Key="greeting">Hello world</x:String>
    </Page.Resources>

    <StackPanel>
        <!-- Displays "Hello world" -->
        <TextBlock x:Name="textBlock1" Text="{StaticResource greeting}"/>

        <Border x:Name="border">
            <Border.Resources>
                <x:String x:Key="greeting">Hola mundo</x:String>
            </Border.Resources>
            <!-- Displays "Hola mundo" -->
            <TextBlock x:Name="textBlock2" Text="{StaticResource
greeting}"/>
        </Border>

        <!-- Displays "Hola mundo", set in code. -->
        <TextBlock x:Name="textBlock3"/>
    </StackPanel>
</Page>
```

C#

```
    public sealed partial class MainPage : Page
    {
```

```
        public MainPage()
        {
            this.InitializeComponent();
            textBlock3.Text = (string)border.Resources["greeting"];
        }
    }
```

Merged resource dictionaries

A *merged resource dictionary* combines one resource dictionary into another, usually in another file.

> **Tip** You can create a resource dictionary file in Microsoft Visual Studio by using the **Add** > **New Item...** > **Resource Dictionary** option from the **Project** menu.

Here, you define a resource dictionary in a separate XAML file called Dictionary1.xaml.

XAML

```
<!-- Dictionary1.xaml -->
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MSDNSample">

    <SolidColorBrush x:Key="brush" Color="Red"/>

</ResourceDictionary>
```

To use that dictionary, you merge it with your page's dictionary:

XAML

```
<Page
    x:Class="MSDNSample.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Page.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Dictionary1.xaml"/>
            </ResourceDictionary.MergedDictionaries>

            <x:String x:Key="greeting">Hello world</x:String>

        </ResourceDictionary>
    </Page.Resources>
```

```
    <TextBlock Foreground="{StaticResource brush}" Text="{StaticResource
greeting}" VerticalAlignment="Center"/>
</Page>
```

Here's what happens in this example. In `<Page.Resources>`, you declare `<ResourceDictionary>`. The XAML framework implicitly creates a resource dictionary for you when you add resources to `<Page.Resources>`; however, in this case, you don't want just any resource dictionary, you want one that contains merged dictionaries.

So you declare `<ResourceDictionary>`, then add things to its `<ResourceDictionary.MergedDictionaries>` collection. Each of those entries takes the form `<ResourceDictionary Source="Dictionary1.xaml"/>`. To add more than one dictionary, just add a `<ResourceDictionary Source="Dictionary2.xaml"/>` entry after the first entry.

After `<ResourceDictionary.MergedDictionaries>…</ResourceDictionary.MergedDictionaries>`, you can optionally put additional resources in your main dictionary. You use resources from a merged to dictionary just like a regular dictionary. In the example above, `{StaticResource brush}` finds the resource in the child/merged dictionary (Dictionary1.xaml), while `{StaticResource greeting}` finds its resource in the main page dictionary.

In the resource-lookup sequence, a MergedDictionaries dictionary is checked only after a check of all the other keyed resources of that ResourceDictionary. After searching that level, the lookup reaches the merged dictionaries, and each item in **MergedDictionaries** is checked. If multiple merged dictionaries exist, these dictionaries are checked in the inverse of the order in which they are declared in the **MergedDictionaries** property. In the following example, if both Dictionary2.xaml and Dictionary1.xaml declared the same key, the key from Dictionary2.xaml is used first because it's last in the **MergedDictionaries** set.

XAML

```
<Page
    x:Class="MSDNSample.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Page.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Dictionary1.xaml"/>
                <ResourceDictionary Source="Dictionary2.xaml"/>
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
```

```
    </Page.Resources>

    <TextBlock Foreground="{StaticResource brush}" Text="greetings!"
VerticalAlignment="Center"/>
</Page>
```

Within the scope of any one ResourceDictionary, the dictionary is checked for key uniqueness. However, that scope does not extend across different items in different MergedDictionaries files.

You can use the combination of the lookup sequence and lack of unique key enforcement across merged-dictionary scopes to create a fallback value sequence of ResourceDictionary resources. For example, you might store user preferences for a particular brush color in the last merged resource dictionary in the sequence, using a resource dictionary that synchronizes to your app's state and user preference data. However, if no user preferences exist yet, you can define that same key string for a **ResourceDictionary** resource in the initial MergedDictionaries file, and it can serve as the fallback value. Remember that any value you provide in a primary resource dictionary is always checked before the merged dictionaries are checked, so if you want to use the fallback technique, don't define that resource in a primary resource dictionary.

Theme resources and theme dictionaries

A ThemeResource is similar to a StaticResource, but the resource lookup is reevaluated when the theme changes.

In this example, you set the foreground of a TextBlock to a value from the current theme.

XAML

```
<TextBlock Text="hello world" Foreground="{ThemeResource
FocusVisualWhiteStrokeThemeBrush}" VerticalAlignment="Center"/>
```

A theme dictionary is a special type of merged dictionary that holds the resources that vary with the theme a user is currently using on his or her device. For example, the "light" theme might use a white color brush whereas the "dark" theme might use a dark color brush. The brush changes the resource that it resolves to, but otherwise the composition of a control that uses the brush as a resource could be the same. To reproduce the theme-switching behavior in your own templates and styles, instead of using MergedDictionaries as the property to merge items into the main dictionaries, use the ThemeDictionaries property.

Each ResourceDictionary element within ThemeDictionaries must have an x:Key value. The value is a string that names the relevant theme—for example, "Default", "Dark", "Light", or "HighContrast". Typically, `Dictionary1` and `Dictionary2` will define resources that have the same names but different values.

Here, you use red text for the light theme and blue text for the dark theme.

XAML

```
<!-- Dictionary1.xaml -->
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MSDNSample">

    <SolidColorBrush x:Key="brush" Color="Red"/>

</ResourceDictionary>

<!-- Dictionary2.xaml -->
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:MSDNSample">

    <SolidColorBrush x:Key="brush" Color="blue"/>

</ResourceDictionary>
```

In this example, you set the foreground of a TextBlock to a value from the current theme.

XAML

```
<Page
    x:Class="MSDNSample.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
    <Page.Resources>
        <ResourceDictionary>
            <ResourceDictionary.ThemeDictionaries>
                <ResourceDictionary Source="Dictionary1.xaml"
x:Key="Light"/>
                <ResourceDictionary Source="Dictionary2.xaml" x:Key="Dark"/>
            </ResourceDictionary.ThemeDictionaries>
        </ResourceDictionary>
    </Page.Resources>
    <TextBlock Foreground="{StaticResource brush}" Text="hello world"
VerticalAlignment="Center"/>
</Page>
```

For theme dictionaries, the active dictionary to be used for resource lookup changes dynamically, whenever ThemeResource markup extension is used to make the reference and the system detects a theme change. The lookup behavior that is done by the system is based on mapping the active theme to the x:Key of a specific theme dictionary.

It can be useful to examine the way that the theme dictionaries are structured in the default XAML design resources, which parallel the templates that the Windows Runtime uses by default for its controls. Open the XAML files in \(Program Files)\Windows Kits\10\DesignTime\CommonConfiguration\Neutral\UAP\<SDK version>\Generic using a text editor or your IDE. Note how the theme dictionaries are defined first in generic.xaml, and how each theme dictionary defines the same keys. Each such key is then referenced by elements of composition in the various keyed elements that are outside the theme dictionaries and defined later in the XAML. There's also a separate themeresources.xaml file for design that contains only the theme resources and extra templates, not the default control templates. The theme areas are duplicates of what you'd see in generic.xaml.

When you use XAML design tools to edit copies of styles and templates, the design tools extract sections from the XAML design resource dictionaries and place them as local copies of XAML dictionary elements that are part of your app and project.

For more info and for a list of the theme-specific and system resources that are available to your app, see XAML theme resources.

Lookup behavior for XAML resource references

Lookup behavior is the term that describes how the XAML resources system tries to find a XAML resource. The lookup occurs when a key is referenced as a XAML resource reference from somewhere in the app's XAML. First, the resources system has predictable behavior for where it will check for the existence of a resource based on scope. If a resource isn't found in the initial scope, the scope expands. The lookup behavior continues on throughout the locations and scopes that a XAML resource could possibly be defined by an app or by the system. If all possible resource lookup attempts fail, an error often results. It's usually possible to eliminate these errors during the development process.

The lookup behavior for XAML resource references starts with the object where the actual usage is applied and its own Resources property. If a ResourceDictionary exists there, that **ResourceDictionary** is checked for an item that has the requested key. This first level of lookup is rarely relevant because you usually do not define and then reference a resource on the same object. In fact, a **Resources** property often doesn't

exist here. You can make XAML resource references from nearly anywhere in XAML; you aren't limited to properties of FrameworkElement subclasses.

The lookup sequence then checks the next parent object in the runtime object tree of the app. If a FrameworkElement.Resources exists and holds a ResourceDictionary, the dictionary item with the specified key string is requested. If the resource is found, the lookup sequence stops and the object is provided to the location where the reference was made. Otherwise, the lookup behavior advances to the next parent level towards the object tree root. The search continues recursively upwards until the root element of the XAML is reached, exhausting the search of all possible immediate resource locations.

> **Note**
>
> It is a common practice to define all the immediate resources at the root level of a page, both to take advantage of this resource-lookup behavior and also as a convention of XAML markup style.

If the requested resource is not found in the immediate resources, the next lookup step is to check the Application.Resources property. **Application.Resources** is the best place to put any app-specific resources that are referenced by multiple pages in your app's navigation structure.

> **Important**
>
> The order of resources added to a ResourceDictionary affects the order in which they are applied. The `XamlControlsResources` dictionary overrides many default resource keys and should therefore be added to `Application.Resources` first so that it doesn't override any other custom styles or resources in your app.

Control templates have another possible location in the reference lookup: theme dictionaries. A theme dictionary is a single XAML file that has a ResourceDictionary element as its root. The theme dictionary might be a merged dictionary from Application.Resources. The theme dictionary might also be the control-specific theme dictionary for a templated custom control.

Finally, there is a resource lookup against platform resources. Platform resources include the control templates that are defined for each of the system UI themes, and which define the default appearance of all the controls that you use for UI in a Windows Runtime app. Platform resources also include a set of named resources that relate to system-wide appearance and themes. These resources are technically a MergedDictionaries item, and thus are available for lookup from XAML or code once the

app has loaded. For example, the system theme resources include a resource named "SystemColorWindowTextColor" that provides a Color definition to match app text color to a system window's text color that comes from the operating system and user preferences. Other XAML styles for your app can refer to this style, or your code can get a resource lookup value (and cast it to **Color** in the example case).

For more info and for a list of the theme-specific and system resources that are available to a Windows app that uses XAML, see XAML theme resources.

If the requested key is still not found in any of these locations, a XAML parsing error/exception occurs. In certain circumstances, the XAML parse exception may be a run-time exception that is not detected either by a XAML markup compile action, or by a XAML design environment.

Because of the tiered lookup behavior for resource dictionaries, you can deliberately define multiple resource items that each have the same string value as the key, as long as each resource is defined at a different level. In other words, although keys must be unique within any given ResourceDictionary, the uniqueness requirement does not extend to the lookup behavior sequence as a whole. During lookup, only the first such object that's successfully retrieved is used for the XAML resource reference, and then the lookup stops. You could use this behavior to request the same XAML resource by key at various positions within your app's XAML but get different resources back, depending on the scope from which the XAML resource reference was made and how that particular lookup behaves.

Forward references within a ResourceDictionary

XAML resource references within a particular resource dictionary must reference a resource that has already been defined with a key, and that resource must appear lexically before the resource reference. Forward references cannot be resolved by a XAML resource reference. For this reason, if you use XAML resource references from within another resource, you must design your resource dictionary structure so that the resources that are used by other resources are defined first in a resource dictionary.

Resources defined at the app level cannot make references to immediate resources. This is equivalent to attempting a forward reference, because the app resources are actually processed first (when the app first starts, and before any navigation-page content is loaded). However, any immediate resource can make a reference to an app resource, and this can be a useful technique for avoiding forward-reference situations.

XAML resources must be shareable

For an object to exist in a ResourceDictionary, that object must be *shareable*.

Being shareable is required because, when the object tree of an app is constructed and used at run time, objects cannot exist at multiple locations in the tree. Internally, the resource system creates copies of resource values to use in the object graph of your app when each XAML resource is requested.

A ResourceDictionary and Windows Runtime XAML in general supports these objects for shareable usage:

- Styles and templates (Style and classes derived from FrameworkTemplate)
- Brushes and colors (classes derived from Brush, and Color values)
- Animation types including Storyboard
- Transforms (classes derived from GeneralTransform)
- Matrix and Matrix3D
- Point values
- Certain other UI-related structures such as Thickness and CornerRadius
- XAML intrinsic data types

You can also use custom types as a shareable resource if you follow the necessary implementation patterns. You define such classes in your backing code (or in runtime components that you include) and then instantiate those classes in XAML as a resource. Examples are object data sources and IValueConverter implementations for data binding.

Custom types must have a default constructor, because that's what a XAML parser uses to instantiate a class. Custom types used as resources can't have the UIElement class in their inheritance, because a **UIElement** can never be shareable (it's always intended to represent exactly one UI element that exists at one position in the object graph of your runtime app).

UserControl usage scope

A UserControl element has a special situation for resource-lookup behavior because it has the inherent concepts of a definition scope and a usage scope. A **UserControl** that makes a XAML resource reference from its definition scope must be able to support the lookup of that resource within its own definition-scope lookup sequence—that is, it cannot access app resources. From a **UserControl** usage scope, a resource reference is treated as being within the lookup sequence towards its usage page root (just like any

other resource reference made from an object in a loaded object tree) and can access app resources.

ResourceDictionary and XamlReader.Load

You can use a ResourceDictionary as either the root or a part of the XAML input for the XamlReader.Load method. You can also include XAML resource references in that XAML if all such references are completely self-contained in the XAML submitted for loading. **XamlReader.Load** parses the XAML in a context that is not aware of any other **ResourceDictionary** objects, not even Application.Resources. Also, don't use `{ThemeResource}` from within XAML submitted to **XamlReader.Load**.

Using a ResourceDictionary from code

Most of the scenarios for a ResourceDictionary are handled exclusively in XAML. You declare the **ResourceDictionary** container and the resources within as a XAML file or set of XAML nodes in a UI definition file. And then you use XAML resource references to request those resources from other parts of XAML. Still, there are certain scenarios where your app might want to adjust the contents of a **ResourceDictionary** using code that executes while the app is running, or at least to query the contents of a **ResourceDictionary** to see if a resource is already defined. These code calls are made on a **ResourceDictionary** instance, so you must first retrieve one—either an immediate **ResourceDictionary** somewhere in the object tree by getting FrameworkElement.Resources, or `Application.Current.Resources`.

In C# or Microsoft Visual Basic code, you can reference a resource in a given ResourceDictionary by using the indexer (Item). A **ResourceDictionary** is a string-keyed dictionary, so the indexer uses the string key instead of an integer index. In Visual C++ component extensions (C++/CX) code, use Lookup.

When using code to examine or change a ResourceDictionary, the behavior for APIs like Lookup or Item does not traverse from immediate resources to app resources; that's a XAML parser behavior that only happens as XAML pages are loaded. At run time, scope for keys is self-contained to the **ResourceDictionary** instance that you are using at the time. However, that scope does extend into MergedDictionaries.

Also, if you request a key that does not exist in the ResourceDictionary, there may not be an error; the return value may simply be provided as **null**. You may still get an error, though, if you try to use the returned **null** as a value. The error would come from the property's setter, not your **ResourceDictionary** call. The only way you'd avoid an error is if the property accepted **null** as a valid value. Note how this behavior contrasts with

XAML lookup behavior at XAML parse time; a failure to resolve the provided key from XAML at parse time results in a XAML parse error, even in cases where the property could have accepted **null**.

Merged resource dictionaries are included into the index scope of the primary resource dictionary that references the merged dictionary at run time. In other words, you can use **Item** or Lookup of the primary dictionary to find any objects that were actually defined in the merged dictionary. In this case, the lookup behavior does resemble the parse-time XAML lookup behavior: if there are multiple objects in merged dictionaries that each have the same key, the object from the last-added dictionary is returned.

You are permitted to add items to an existing ResourceDictionary by calling **Add** (C# or Visual Basic) or Insert (C++/CX). You could add the items to either immediate resources or app resources. Either of these API calls requires a key, which satisfies the requirement that each item in a **ResourceDictionary** must have a key. However, items that you add to a **ResourceDictionary** at run time are not relevant to XAML resource references. The necessary lookup for XAML resource references happens when that XAML is first parsed as the app is loaded (or a theme change is detected). Resources added to collections at run time weren't available then, and altering the **ResourceDictionary** doesn't invalidate an already retrieved resource from it even if you change the value of that resource.

You also can remove items from a ResourceDictionary at run time, make copies of some or all items, or other operations. The members listing for **ResourceDictionary** indicates which APIs are available. Note that because **ResourceDictionary** has a projected API to support its underlying collection interfaces, your API options differ depending on whether you are using C# or Visual Basic versus C++/CX.

ResourceDictionary and localization

A XAML ResourceDictionary might initially contain strings that are to be localized. If so, store these strings as project resources instead of in a **ResourceDictionary**. Take the strings out of the XAML, and instead give the owning element an x:Uid directive value. Then, define a resource in a resources file. Provide a resource name in the form *XUIDValue.PropertyName* and a resource value of the string that should be localized.

Custom resource lookup

For advanced scenarios, you can implement a class that can have different behavior than the XAML resource reference lookup behavior described in this topic. To do this, you implement the class CustomXamlResourceLoader, and then you can access that behavior by using the CustomResource markup extension for resource references rather

than using StaticResource or ThemeResource. Most apps won't have scenarios that require this. For more info, see CustomXamlResourceLoader.

Related topics

- ResourceDictionary
- XAML overview
- StaticResource markup extension
- ThemeResource markup extension
- XAML theme resources
- Styling controls
- x:Key attribute

XAML theme resources

Article • 11/26/2024

Theme resources in XAML are a set of resources that apply different values depending on which system theme is active. There are 3 themes that the XAML framework supports: "Light", "Dark", and "HighContrast".

Prerequisites: This topic assumes that you have read ResourceDictionary and XAML resource references.

Theme resources v. static resources

There are two XAML markup extensions that can reference a XAML resource from an existing XAML resource dictionary: {StaticResource} markup extension and {ThemeResource} markup extension.

Evaluation of a {ThemeResource} markup extension occurs when the app loads and subsequently each time the theme changes at runtime. This is typically the result of the user changing their device settings or from a programmatic change within the app that alters its current theme.

In contrast, a {StaticResource} markup extension is evaluated only when the XAML is first loaded by the app. It does not update. It's similar to a find and replace in your XAML with the actual runtime value at app launch.

Theme resources in the resource dictionary structure

Each theme resource is part of the XAML file themeresources.xaml. For design purposes, themeresources.xaml is available in the \(Program Files)\Windows Kits\10\DesignTime\CommonConfiguration\Neutral\UAP\<SDK version>\Generic folder from a Windows Software Development Kit (SDK) installation. The resource dictionaries in themeresources.xaml are also reproduced in generic.xaml in the same directory.

The Windows Runtime doesn't use these physical files for runtime lookup. That's why they are specifically in a DesignTime folder, and they aren't copied into apps by default. Instead, these resource dictionaries exist in memory as part of the Windows Runtime itself, and your app's XAML resource references to theme resources (or system resources) resolve there at runtime.

Guidelines for custom theme resources

Follow these guidelines when you define and consume your own custom theme resources:

- Specify theme dictionaries for both "Light" and "Dark" in addition to your "HighContrast" dictionary. Although you can create a ResourceDictionary with "Default" as the key, it's preferred to be explicit and instead use "Light", "Dark", and "HighContrast".
- Use the {ThemeResource} markup extension in: Styles, Setters, Control templates, Property setters, and Animations.
- Don't use the {ThemeResource} markup extension in your resource definitions inside your ThemeDictionaries. Use {StaticResource} markup extension instead.

  EXCEPTION: You can use the {ThemeResource} markup extension to reference resources that are agnostic to the app theme in your ThemeDictionaries. Examples of these resources are accent color resources like `SystemAccentColor`, or system color resources, which are typically prefixed with "SystemColor" like `SystemColorButtonFaceColor`.

> **⊗ Caution**
>
> If you don't follow these guidelines, you might see unexpected behavior related to themes in your app. For more info, see the **Troubleshooting theme resources** section.

The XAML color ramp and theme-dependent brushes

The combined set of colors for "Light", "Dark", and "HighContrast" themes make up the *Windows color ramp* in XAML. Whether you want to modify the system themes, or apply a theme to your own XAML elements, it's important to understand how the color resources are structured.

For additional information about how to apply color in your Windows app, please see Color in Windows apps.

Light and Dark theme colors

The XAML framework provides a set of named Color resources with values that are tailored for the "Light" and "Dark" themes. For WinUI 2, the theme resources are defined in the Common theme resources Xaml file. The color names are very descriptive of their intended usage, and there's a corresponding SolidColorBrush resource for every Color resource.

> **Tip**

For a visual overview of these colors, see the WinUI 3 Gallery app: **Colors**

> The **WinUI 3 Gallery** app includes interactive examples of most WinUI 3 controls, features, and functionality. Get the app from the Microsoft Store or get the source code on GitHub

Windows system contrast theme colors

In addition to the set of resources provided by the XAML framework, there's a set of color values derived from the Windows system palette. These colors are not specific to the Windows Runtime or Windows apps. However, many of the XAML Brush resources consume these colors when the system is operating (and the app is running) using the "HighContrast" theme. The XAML framework provides these system-wide colors as keyed resources. The keys follow the naming format: `SystemColor[name]Color`.

For more information about supporting contrast themes, see Contrast themes.

System accent color

In addition to the system contrast theme colors, the system accent color is provided as a special color resource using the key `SystemAccentColor`. At runtime, this resource gets the color that the user has specified as the accent color in the Windows personalization settings.

> **Note**
>
> While it's possible to override the system color resources, it's a best practice to respect the user's color choices, especially for contrast theme settings.

Theme-dependent brushes

The color resources shown in the preceding sections are used to set the Color property of SolidColorBrush resources in the system theme resource dictionaries. You use the brush resources to apply the color to XAML elements.

Let's look at how the color value for this brush is determined at run-time. In the "Light" and "Dark" resource dictionaries, this brush is defined like this:

```
<SolidColorBrush x:Key="TextFillColorPrimaryBrush" Color="{StaticResource TextFillColorPrimary}"/>
```

In the "HighContrast" resource dictionary, this brush is defined like this:

```
<SolidColorBrush x:Key="TextFillColorPrimaryBrush" Color="{ThemeResource SystemColorWindowTextColor}"/>
```

When this brush is applied to a XAML element, its color is determined at run-time by the current theme, as shown in this table.

Expand table

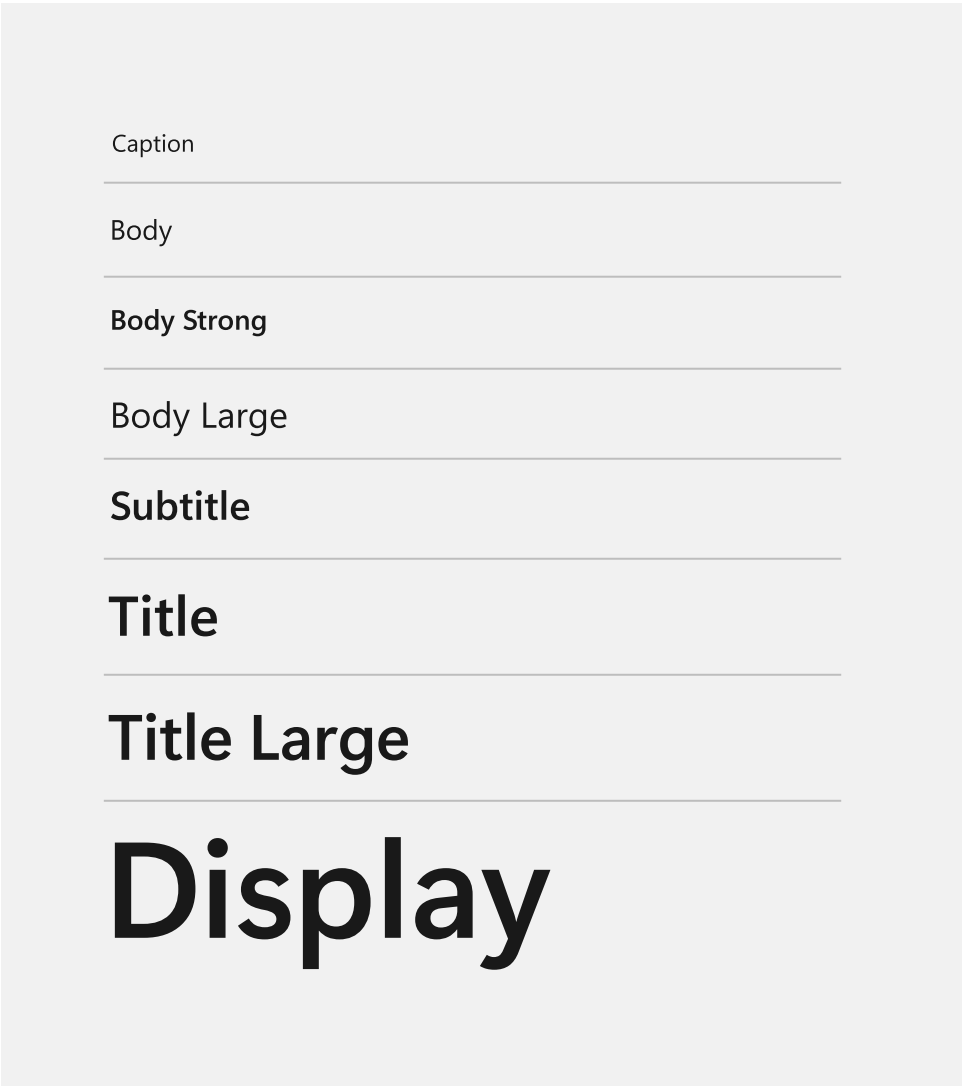
Theme	Color resource	Runtime value
Light	TextFillColorPrimary	#E4000000
Dark	TextFillColorPrimary	#FFFFFFFF
HighContrast	SystemColorWindowTextColor	The color specified in settings for Text.

The XAML type ramp

The themeresources.xaml file defines several resources that define a Style that you can apply to text containers in your UI, specifically for either TextBlock or RichTextBlock. These are not the default implicit styles. They are provided to make it easier for you to create XAML UI definitions that match the *Windows type ramp* documented in Guidelines for fonts.

These styles are for text attributes that you want applied to the whole text container. If you want styles applied just to sections of the text, set attributes on the text elements within the container, such as on a Run in TextBlock.Inlines or on a Paragraph in RichTextBlock.Blocks.

The styles look like this when applied to a TextBlock:



Expand table

Style	Weight	Size
Caption	Regular	12
Body	Regular	14
Body Strong	Semibold	14
Body Large	Regular	18
Subtitle	Semibold	20
Title	Semibold	28
Title Large	Semibold	40
Display	Semibold	68

XAML

```
<TextBlock Text="Caption" Style="{StaticResource CaptionTextBlockStyle}"/>
<TextBlock Text="Body" Style="{StaticResource BodyTextBlockStyle}"/>
<TextBlock Text="Body Strong" Style="{StaticResource
BodyStrongTextBlockStyle}"/>
<TextBlock Text="Body Large" Style="{StaticResource
BodyLargeTextBlockStyle}"/>
<TextBlock Text="Subtitle" Style="{StaticResource SubtitleTextBlockStyle}"/>
<TextBlock Text="Title" Style="{StaticResource TitleTextBlockStyle}"/>
<TextBlock Text="Title Large" Style="{StaticResource
TitleLargeTextBlockStyle}"/>
<TextBlock Text="Display" Style="{StaticResource DisplayTextBlockStyle}"/>
```

For guidance on how to use the Windows type ramp in your app, see Typography in Windows apps.

For details of the XAML styles, see WinUI on GitHub:

- Caption TextBlockStyle
- Body TextBlockStyle
- Body Strong TextBlockStyle

- Subtitle TextBlockStyle
- Title TextBlockStyle
- Title Large TextBlockStyle
- Display TextBlockStyle

Tip

For a visual overview of these styles, see the WinUI 3 Gallery app: Typography

BaseRichTextBlockStyle

TargetType: RichTextBlock

Supplies the common properties for all the other RichTextBlock container styles.

XAML

```
<!-- Usage -->
<RichTextBlock Style="{StaticResource BaseRichTextBlockStyle}">
    <Paragraph>Rich text.</Paragraph>
</RichTextBlock>

<!-- Style definition -->
<Style x:Key="BaseRichTextBlockStyle" TargetType="RichTextBlock">
```

```
    <Setter Property="FontFamily" Value="Segoe UI"/>
    <Setter Property="FontWeight" Value="SemiBold"/>
    <Setter Property="FontSize" Value="14"/>
    <Setter Property="TextTrimming" Value="None"/>
    <Setter Property="TextWrapping" Value="Wrap"/>
    <Setter Property="LineStackingStrategy" Value="MaxHeight"/>
    <Setter Property="TextLineBounds" Value="Full"/>
    <Setter Property="OpticalMarginAlignment" Value="TrimSideBearings"/>
</Style>
```

BodyRichTextBlockStyle

XAML

```
<!-- Usage -->
<RichTextBlock Style="{StaticResource BodyRichTextBlockStyle}">
    <Paragraph>Rich text.</Paragraph>
</RichTextBlock>

<!-- Style definition -->
<Style x:Key="BodyRichTextBlockStyle" TargetType="RichTextBlock" BasedOn="
{StaticResource BaseRichTextBlockStyle}">
    <Setter Property="FontWeight" Value="Normal"/>
</Style>
```

Note: The RichTextBlock styles don't have all the text ramp styles that TextBlock does, mainly because the block-based document object model for **RichTextBlock** makes it easier to set attributes on the individual text elements. Also, setting TextBlock.Text using the XAML content property introduces a situation where there is no text element to style and thus you'd have to style the container. That isn't an issue for **RichTextBlock** because its text content always has to be in specific text elements like Paragraph, which is where you might apply XAML styles for page header, page subheader and similar text ramp definitions.

Miscellaneous Named styles

There's an additional set of keyed Style definitions you can apply to style a Button differently than its default implicit style.

NavigationBackButtonNormalStyle

TargetType: Button

This Style provides a complete template for a Button that can be the navigation back button for a navigation app. The default dimensions are 40 x 40 pixels. To tailor the styling you can either explicitly set the Height, Width, FontSize, and other properties on your **Button** or create a derived style using BasedOn.

Here's a Button with the **NavigationBackButtonNormalStyle** resource applied to it.

XAML

```
<Button Style="{StaticResource NavigationBackButtonNormalStyle}" />
```

It looks like this:

NavigationBackButtonSmallStyle

TargetType: Button

This Style provides a complete template for a Button that can be the navigation back button for a navigation app. It's similar to **NavigationBackButtonNormalStyle**, but its dimensions are 30 x 30 pixels.

Here's a Button with the **NavigationBackButtonSmallStyle** resource applied to it.

XAML

```
<Button Style="{StaticResource NavigationBackButtonSmallStyle}" />
```

Troubleshooting theme resources

If you don't follow the guidelines for using theme resources, you might see unexpected behavior related to themes in your app.

For example, when you open a light-themed flyout, parts of your dark-themed app also change as if they were in the light theme. Or if you navigate to a light-themed page and then navigate back, the original dark-themed page (or parts of it) now looks as though it is in the light theme.

Typically, these types of issues occur when you provide a "Default" theme and a "HighContrast" theme to support high-contrast scenarios, and then use both "Light" and

"Dark" themes in different parts of your app.

For example, consider this theme dictionary definition:

XAML

```
<!-- DO NOT USE. THIS XAML DEMONSTRATES AN ERROR. -->
<ResourceDictionary>
  <ResourceDictionary.ThemeDictionaries>
    <ResourceDictionary x:Key="Default">
      <SolidColorBrush x:Key="myBrush" Color="{ThemeResource
ControlFillColorDefault}"/>
    </ResourceDictionary>
    <ResourceDictionary x:Key="HighContrast">
      <SolidColorBrush x:Key="myBrush" Color="{ThemeResource
SystemColorButtonFaceColor}"/>
    </ResourceDictionary>
  </ResourceDictionary.ThemeDictionaries>
</ResourceDictionary>
```

Intuitively, this looks correct. You want to change the color pointed to by `myBrush` when in high-contrast, but when not in high-contrast, you rely on the {ThemeResource} markup extension to make sure that `myBrush` points to the right color for your theme. If your app never has FrameworkElement.RequestedTheme set on elements within its visual tree, this will typically work as expected. However, you run into problems in your app as soon as you start to re-theme different parts of your visual tree.

The problem occurs because brushes are shared resources, unlike most other XAML types. If you have 2 elements in XAML sub-trees with different themes that reference the same brush resource, then as the framework walks each sub-tree to update its {ThemeResource} markup extension expressions, changes to the shared brush resource are reflected in the other sub-tree, which is not your intended result.

To fix this, replace the "Default" dictionary with separate theme dictionaries for both "Light" and "Dark" themes in addition to "HighContrast":

XAML

```
<!-- DO NOT USE. THIS XAML DEMONSTRATES AN ERROR. -->
<ResourceDictionary>
  <ResourceDictionary.ThemeDictionaries>
    <ResourceDictionary x:Key="Light">
      <SolidColorBrush x:Key="myBrush" Color="{ThemeResource
ControlFillColorDefault}"/>
    </ResourceDictionary>
    <ResourceDictionary x:Key="Dark">
      <SolidColorBrush x:Key="myBrush" Color="{ThemeResource
ControlFillColorDefault}"/>
```

```
    </ResourceDictionary>
    <ResourceDictionary x:Key="HighContrast">
      <SolidColorBrush x:Key="myBrush" Color="{ThemeResource
SystemColorButtonFaceColor}"/>
    </ResourceDictionary>
  </ResourceDictionary.ThemeDictionaries>
</ResourceDictionary>
```

However, problems still occur if any of these resources are referenced in inherited properties like Foreground. Your custom control template might specify the foreground color of an element using the {ThemeResource} markup extension, but when the framework propagates the inherited value to child elements, it provides a direct reference to the resource that was resolved by the {ThemeResource} markup extension expression. This causes problems when the framework processes theme changes as it walks your control's visual tree. It re-evaluates the {ThemeResource} markup extension expression to get a new brush resource but doesn't yet propagate this reference down to the children of your control; this happens later, such as during the next measure pass.

As a result, after walking the control visual tree in response to a theme change, the framework walks the children and updates any {ThemeResource} markup extension expressions set on them, or on objects set on their properties. This is where the problem occurs; the framework walks the brush resource and because it specifies its color using a {ThemeResource} markup extension, it's re-evaluated.

At this point, the framework appears to have polluted your theme dictionary because it now has a resource from one dictionary that has its color set from another dictionary.

To fix this problem, use the {StaticResource} markup extension instead of {ThemeResource} markup extension. With the guidelines applied, the theme dictionaries look like this:

XAML

```
<ResourceDictionary>
  <ResourceDictionary.ThemeDictionaries>
    <ResourceDictionary x:Key="Light">
      <SolidColorBrush x:Key="myBrush" Color="{StaticResource
ControlFillColorDefault}"/>
    </ResourceDictionary>
    <ResourceDictionary x:Key="Dark">
      <SolidColorBrush x:Key="myBrush" Color="{StaticResource
ControlFillColorDefault}"/>
    </ResourceDictionary>
    <ResourceDictionary x:Key="HighContrast">
      <SolidColorBrush x:Key="myBrush" Color="{ThemeResource
SystemColorButtonFaceColor}"/>
    </ResourceDictionary>
```

```
  </ResourceDictionary.ThemeDictionaries>
</ResourceDictionary>
```

Notice that the {ThemeResource} markup extension is still used in the "HighContrast" dictionary instead of {StaticResource} markup extension. This situation falls under the exception given earlier in the guidelines. Most of the brush values that are used for the "HighContrast" theme are using color choices that are globally controlled by the system, but exposed to XAML as a specially-named resource (those prefixed with 'SystemColor' in the name). The system enables the user to set the specific colors that should be used for their contrast theme settings through the Ease of Access Center. Those color choices are applied to the specially-named resources. The XAML framework uses the same theme changed event to also update these brushes when it detects they've changed at the system level. This is why the {ThemeResource} markup extension is used here.

API reference for Windows desktop apps

Article • 07/17/2024

The following collections of API frameworks can be used to build Windows desktop apps for Windows 11 and Windows 10.

Windows SDK

The Windows SDK ships with a corresponding release of the Windows OS and includes the latest headers, libraries, metadata, and tools for building Universal Windows Platform (UWP) and Win32 applications.

The following API frameworks ship with the Windows SDK:

- WinRT API reference
- WinUI 2 for UWP API reference
- Win32 API reference

Windows App SDK

The Windows App SDK provides a comprehensive set of Windows APIs with implementations that are decoupled from the operating system (OS) and the Windows SDK. The Windows App SDK ships independently as a NuGet package on a regular basis.

The following API frameworks ship with the Windows App SDK:

- WinRT API reference
- Win32 API reference
- COM interop APIs for WinUI reference
- C# Interop APIs for WinUI reference
- Bootstrapper C# APIs reference

.NET

.NET is a cross-platform, open source developer platform for building many different types of applications, including Windows desktop apps.

- .NET API reference

Schema specifications

Universal Windows Platform (UWP) apps include various file and XML schema specifications.

- Schema reference

> **Important**
>
> For inquiries about Windows data access or interoperability related to the Digital Markets Act (DMA), please use the **DMA Request Form** .

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see our contributor guide.

Windows developer feedback

Windows developer is an open source project. Select a link to provide feedback:

- Open a documentation issue
- Provide product feedback

Windows Runtime (WinRT) namespaces

Article03/22/2023

Find detailed information about Windows Runtime (WinRT) APIs.

Expand table

Namespace	Description
Windows.AI.MachineLearning	Enables apps to load machine learning models, bind features, and evaluate the results.
Windows.AI.MachineLearning.Preview	This namespace contains deprecated APIs. Please use Windows.AI.MachineLearning instead.
Windows.ApplicationModel	Provides an app with access to core system functionality and run-time information about its app package, and handles suspend operations.
Windows.ApplicationModel.Activation	Enables an app to handle the activation parameters for the activation contracts, appointments, and extensions supported by Windows.
Windows.ApplicationModel.AppExtensions	Enable your app to use content provided by other apps. Discover, enumerate, and access read-only content from those apps.
Windows.ApplicationModel.Appointments	Contains the Appointment class and related support classes. An Appointment represents an appointment in a calendar.
Windows.ApplicationModel.Appointments.AppointmentsProvider	Defines operations classes for add appointment and remove appointment requests through activations that an appointments provider interacts with.
Windows.ApplicationModel.Appointments.DataProvider	Contains the AppointmentDataProviderConnection class and support classes related to its events. An **AppointmentDataProviderConnection** allows data transfer between a Universal Windows app and the app or server that owns the calendar information of a user on the device.
Windows.ApplicationModel.AppService	Windows app services enable app-to-app communication by allowing your Universal Windows app to provide services to other Universal Windows app. App services let you create UI-less services that apps can call on the same device, and starting with Windows 10, version 1607, on remote devices.
Windows.ApplicationModel.Background	Enables an app to schedule background tasks to run app code even when the app is suspended.
Windows.ApplicationModel.Calls	Controls multiple aspects of how an application can interact with a phone device, including lock-screen

Namespace	Description
	interaction, the phone call history, and various telephony options and information.
Windows.ApplicationModel.Calls.Background	The classes in Windows.ApplicationModel.Calls.Background are used by device background tasks to handle changes to phone lines, call states, and voice mail counts for the device.
Windows.ApplicationModel.Calls.Provider	Enables an application to record and provide information about the origin of a phone call.
Windows.ApplicationModel.Chat	This namespace allows apps to implement messaging through SMS, MMS, and RCS protocols on devices that support it.
Windows.ApplicationModel.CommunicationBlocking	Enables applications to support blocking calls and messages.
Windows.ApplicationModel.Contacts	Enables apps to select and acquire info about contacts.
Windows.ApplicationModel.Contacts.DataProvider	This API is implemented by a server to provide contact information to client apps.
Windows.ApplicationModel.Contacts.Provider	Enables apps to create and remove contacts.
Windows.ApplicationModel.ConversationalAgent	Provides applications the ability to expose functionality through any digital assistant supported by the Windows Conversational Agent platform.
Windows.ApplicationModel.Core	Provides key application-level functionality for apps.
Windows.ApplicationModel.DataTransfer	Contains classes that are used to exchange data between a source app and a target app. These classes are used in share and clipboard operations.
Windows.ApplicationModel.DataTransfer.DragDrop	Manages dragging and dropping items.
Windows.ApplicationModel.DataTransfer.DragDrop.Core	Manages dragging and dropping items.
Windows.ApplicationModel.DataTransfer.ShareTarget	Contains the classes and events that a target app uses to complete share operations and create shortcuts, called QuickLinks, that the system can use to streamline future sharing operations to the app.
Windows.ApplicationModel.Email	Supports creating and interacting with email messages, recipients, and attachments.
Windows.ApplicationModel.Email.DataProvider	This API is implemented by an email server to provide email data to an email client app.
Windows.ApplicationModel.ExtendedExecution	Provides a mechanism for applications to execute when they are not in the foreground.
Windows.ApplicationModel.ExtendedExecution.Foreground	Provides a mechanism for applications to perform extended execution while they are in the

Namespace	Description
	foreground.
Windows.ApplicationModel.Holographic	Provides application model APIs for Windows Holographic apps.
Windows.ApplicationModel.LockScreen	Classes related to the Windows lock screen. Please note that creating a new lock screen is a feature **not** available to third party developers.
Windows.ApplicationModel.PackageExtensions	Enable your app to use content provided by other packages. Discover, enumerate, and access read-only content from those packages.
Windows.ApplicationModel.Payments	This namespace provides payment functionality equivalent to the W3C Payment Request API specification for Universal Windows Platform (UWP) apps. The W3C Payment Request API allows websites to collect payment information in a way that is easy for both the merchant and the user.
Windows.ApplicationModel.Payments.Provider	This namespace provides API that allow your app to facilitate payment transactions submitted by users.
Windows.ApplicationModel.Preview.Holographic	Contains classes that let apps interact with the holographic shell, allowing the user to see content in the world around them.
Windows.ApplicationModel.Preview.InkWorkspace	Provides support for specifying inking apps in the Ink Workspace.
Windows.ApplicationModel.Preview.Notes	Enables a note-taking app to implement a number of features for a faster and more effective note-taking experience in certain situations (see Remarks).
Windows.ApplicationModel.Resources	Provides simplified access to application resources, such as strings, that are defined using basic naming conventions for runtime apps.
Windows.ApplicationModel.Resources.Core	Provides classes that enable advanced resource loading.
Windows.ApplicationModel.Resources.Management	Enables the indexing of source resource files or containers, generating named resource candidates using particular file and folder naming conventions.
Windows.ApplicationModel.Search	Provides support for the search pane in apps that participate in the Search contract. The search pane opens when users activate the Search charm and provides a consistent, touch-friendly search box and optional search suggestions.
Windows.ApplicationModel.Search.Core	Provides access to search suggestions for the in-app SearchBox control and the search pane.
Windows.ApplicationModel.SocialInfo	Enables social media apps to supply feed items to Windows.

Namespace	Description
Windows.ApplicationModel.SocialInfo.Provider	Implements functionality for providing social data.
Windows.ApplicationModel.Store	Provides types and members for interacting with the Windows Store to add in-app purchases and trial functionality to your Universal Windows Platform (UWP) app.
Windows.ApplicationModel.Store.LicenseManagement	Provides APIs for managing license info for apps.
Windows.ApplicationModel.Store.Preview	Provides preview APIs for various Microsoft Store-related scenarios.
Windows.ApplicationModel.Store.Preview.InstallControl	Provides APIs for programmatically managing the installation of apps and app updates.
Windows.ApplicationModel.UserActivities	A User Activity encapsulates a user task that can be continued at a later time, and potentially on a different device.
Windows.ApplicationModel.UserActivities.Core	Manages the creation and deletion of user activities. A **UserActivity** encapsulates a user's task that can be continued at a later time, and potentially on a different device.
Windows.ApplicationModel.UserDataAccounts	Contains user data account information needed to access email, appointments, calendars, and so on.
Windows.ApplicationModel.UserDataAccounts.Provider	Enables an app to replace the existing account management and syncing functionality previously handled by the core operating system.
Windows.ApplicationModel.UserDataAccounts.SystemAccess	Provides functionality for apps to invoke system-level UI to manage sync partnerships on the device.
Windows.ApplicationModel.UserDataTasks	Enables apps to select and acquire info about tasks.
Windows.ApplicationModel.UserDataTasks.DataProvider	This API is implemented by a server to provide task information to client apps.
Windows.ApplicationModel.VoiceCommands	Provides support for handling voice commands in **Cortana**, entered by speech or text, to access features and functionality from a background app. When an app handles a voice command in the background it can display feedback on the **Cortana** canvas and communicate with the user using the **Cortana** voice.
Windows.ApplicationModel.Wallet	
Windows.ApplicationModel.Wallet.System	Defines a set of asynchronous actions and operations that manage the wallet items in the system wallet database. These can be used to read the existing items, or to add and remove items.
Windows.Data.Html	Provides utility classes for working with HTML.

Namespace	Description
Windows.Data.Json	Provides support for JSON arrays, values, and objects.
Windows.Data.Pdf	Contains classes for converting a page in a Portable Document Format (PDF) document to an image file.
Windows.Data.Text	Provides support for identifying text segments and for determining the properties of Unicode characters.
Windows.Data.Xml.Dom	Provides support for the XML Document Object Model (DOM).
Windows.Data.Xml.Xsl	Provides support for the XSLT processor.
Windows.Devices	This namespace provides access to the low level device providers, including ADC, GPIO, I^2C, PWM and SPI.
Windows.Devices.Adc	Analog-Digital Converter, or ADC, is an electrical component that converts an analog electrical signal (usually voltage) to a digital representation. ADCs usually interface a digital circuit that processes signals from analog world.
Windows.Devices.Adc.Provider	This API exposes a Provider interface for the user to implement for a specific controller. The ADC API needs a ADC Provider implementation to talk to the ADC controller.
Windows.Devices.AllJoyn	
Windows.Devices.Background	Enables the ability for device background tasks to get info provided by the app that triggered the task and to report progress back to the app.
Windows.Devices.Bluetooth	The **Windows.Devices.Bluetooth** namespace defines a set of Windows Runtime API that allows UWP app and desktop apps to interact with Bluetooth devices. For more information, see Bluetooth.
Windows.Devices.Bluetooth.Advertisement	Allow apps to send and receive Bluetooth Low Energy (LE) advertisements.
Windows.Devices.Bluetooth.Background	Provides class describing trigger details for Bluetooth triggers.
Windows.Devices.Bluetooth.GenericAttributeProfile	The **Windows.Devices.Bluetooth.GenericAttributeProfile** namespace defines Windows Runtime classes that a UWP app or a desktop application can use to communicate with Bluetooth LE devices.
Windows.Devices.Bluetooth.Rfcomm	The **Windows.Devices.Bluetooth.Rfcomm** namespace defines Windows Runtime classes that a UWP app and desktop applications can use to

Namespace	Description
	communicate with Bluetooth devices. For more information, see BluetoothTo learn more about how to use **Windows.Devices.Bluetooth.Rfcomm**, see the Bluetooth Rfcomm Chat Sample.
Windows.Devices.Custom	Opens a custom device asynchronously, and with the object returned from the open operation send asynchronous IOCTLs to the device.
Windows.Devices.Display	Contains components to determine aspects of a physical display.
Windows.Devices.Display.Core	Provides ways to manage and present content to the hardware display pipeline for custom compositor software.
Windows.Devices.Enumeration	Provides classes for enumerating devices.
Windows.Devices.Enumeration.Pnp	
Windows.Devices.Geolocation	Provides APIs for getting the current location or tracking the device's location over time. Location information may come from estimating a position from beacons like Wi-Fi access points and cell towers, from the device's IP address, or it may come from other sources such as a GNSS or GPS device. The Windows.Devices.Geolocation API provides the most appropriate geolocation data from all available sources.
Windows.Devices.Geolocation.Geofencing	A geofence is a virtual area around a geographical point. The geofencing APIs enable applications to provide geographically contextual experiences in a timely manner without the need for the app to be continuously running and consuming device resources. Geofencing enables scenarios like popping up a reminder for a user when they are leaving work or home or displaying coupons when the user gets within range of a store.
Windows.Devices.Geolocation.Provider	There are some scenarios in which the physical location of the devices isn't sufficient—either because it doesn't represent the user's real location, or the current positioning capabilities of the device aren't good enough to satisfy a particular scenario. In those cases, there's a need for a remote location source to *override* the current device location, and not only be a last resolution. The types in this namespace provide those facilities.
Windows.Devices.Gpio	Contains types for using general-purpose I/O (GPIO) pins in user mode.
Windows.Devices.Gpio.Provider	This API exposes a provider interface, and associated classes for you to implement for a specific controller.

Namespace	Description
	The GPIO API needs a GPIO provider implementation to talk to the GPIO controller.
Windows.Devices.Haptics	Provides support for basic, non-directional haptic feedback such as clicks, buzzes, and rumbles that can be tuned through characteristics such as intensity and timing.
Windows.Devices.HumanInterfaceDevice	This namespace gives your UWP app access to devices that support the Human Interface Device (HID) protocol.
Windows.Devices.I2c	Contains types that you can use to communicate with peripheral devices connected through a inter-integrated circuit (I^2C) bus from an application.
Windows.Devices.I2c.Provider	This API exposes a Provider interface and associated classes for the user to implement for a specific controller. The I2C API needs an I^2C Provider implementation to talk to the I^2C controller.
Windows.Devices.Input	Provides support for identifying the input devices available (pointer, touch, mouse, and keyboard) and retrieving information about those devices.
Windows.Devices.Input.Preview	Provide support for tracking a user's gaze, attention, and presence based on the location and movement of their eyes and head.
Windows.Devices.Lights	Provides APIs for accessing and controlling LampArrays and legacy Lamp devices.
Windows.Devices.Lights.Effects	Provides simple effects for common LampArray scenarios.
Windows.Devices.Midi	This namespace provides types for using MIDI devices in a Windows Runtime (WinRT) app.
Windows.Devices.Perception	Provides classes for handling image frames from color cameras, depth sensors, and infrared cameras.
Windows.Devices.Perception.Provider	Contains types for implementing providers of perception frames.
Windows.Devices.PointOfService	Enables application developers to access Point of Service (POS) peripheral devices. The namespace provides a vendor-neutral interface for accessing POS devices from various manufacturers from the UWP app.
Windows.Devices.PointOfService.Provider	Enables a server to provide barcode scanning functionality to client applications.
Windows.Devices.Portable	Windows Portable Devices (WPD) enables computers to communicate with attached media and storage devices. This namespace provides

Namespace	Description
	methods for identifying Windows Portable Device (WPD) services and storage.
Windows.Devices.Power	Provides APIs for detailed battery information. For more info, see Get battery information.
Windows.Devices.Printers	Provides APIs for working with 3D printers.
Windows.Devices.Printers.Extensions	Contains classes and interfaces that provide access to the Windows Runtime Components for print task configuration and print notifications.
Windows.Devices.Pwm	PWM stands for Pulse Width Modulation and describes the basic technique to create repeated square waves of user defined length and duty cycle. Most common uses include driving servo motors, dimming LEDs etc. This API provides users the ability to generate PWM wave forms using a PWM controller connected to the system.
Windows.Devices.Pwm.Provider	This API exposes a Provider interface for the user to implement for a specific controller. The PWM API needs a PWM Provider implementation to talk to the PWM controller.
Windows.Devices.Radios	The members of this namespace allow your app to find and control radio adapters, like Wi-Fi and Bluetooth, on the device where your app runs, and to query those radios for certain attributes.
Windows.Devices.Scanners	Contains the classes, interfaces, structures and other components that are built on top of the Windows Image Acquisition (WIA) COM APIs to provide scan functionality for UWP apps and desktop apps.
Windows.Devices.Sensors	Provides access to the supported types of sensors and related motion data.
Windows.Devices.Sensors.Custom	Provides access to a custom sensor and its related data.
Windows.Devices.SerialCommunication	The `Windows.Devices.SerialCommunication` namespace defines Windows Runtime classes that a UWP app can use to communicate with a device that exposes a serial port or some abstraction of a serial port. The classes provide functionality to discover such serial devices, read and write data, and control serial-specific properties for flow control, such as setting baud rate, signal states.
Windows.Devices.SmartCards	Gets info about smart card readers and smart cards; configures physical smart cards; and creates, configures, and deletes Trusted Platform Module (TPM) virtual smart cards.

Namespace	Description
Windows.Devices.Sms	Provides the ability to send and receive Short Message Service (SMS) messages, filter incoming messages, and send messages silently, as well as to query properties about a Mobile Broadband SMS Device. Use this API to implement Mobile Operator (MO) applications that use app-directed SMS to implement MO features such as visual voice mail apps.
Windows.Devices.Spi	Serial Peripheral Interface (SPI) is a high-speed interface used to communicate among integrated components on the same board.
Windows.Devices.Spi.Provider	This API exposes a Provider interface and associated classes for the user to implement for a specific controller. The SPI API needs an SPI Provider implementation to talk to the SPI controller.
Windows.Devices.Usb	This namespace defines Windows Runtime classes that a UWP app can use to communicate with WinUSB devices. These devices are handled by the inbox `winusb.sys` driver and are identified by a specific Microsoft OS Descriptor.
Windows.Devices.WiFi	
Windows.Devices.WiFiDirect	Contains classes that support connecting to associated Wi-Fi Direct devices and associated endpoints for PCs, tablets, and phones.
Windows.Devices.WiFiDirect.Services	Provides support for implementing your own Wi-Fi Direct Services.
Windows.Embedded.DeviceLockdown	Provides a simple and consistent interface for locking down the available applications and tiles on the device for specific user roles defined by the enterprise.
Windows.Foundation	Enables fundamental Windows Runtime functionality, including managing asynchronous operations and accessing property stores. This namespace also defines common value types that represent Uniform Resource Identifier (URI), dates and times, 2-D measurements, and other basic values.
Windows.Foundation.Collections	Defines standard, specialized, and generic collection objects. This includes the IIterable<T>, IVector<T> and IMap<K,V> generic interfaces, which are used as the pattern for basic collection support throughout the Windows Runtime API.
Windows.Foundation.Diagnostics	Provides classes for logging messages and tracing events.

Namespace	Description
Windows.Foundation.Metadata	Defines the attributes that indicate fundamental properties of Windows Runtime types and members.
Windows.Foundation.Numerics	Provides a set of graphics-oriented math types.
Windows.Gaming.Input	Provides access to game controller input for Windows applications.
Windows.Gaming.Input.Custom	Contains APIs enabling equipment manufacturers to write custom WinRT (Windows Runtime) class libraries for their devices.
Windows.Gaming.Input.ForceFeedback	Contains APIs for controlling the force feedback motor on a controller.
Windows.Gaming.Input.Preview	Provides APIs that support functionality for legacy gaming devices.
Windows.Gaming.Preview.GamesEnumeration	A set of APIs that provides querying installed games and their associated properties for launching and managing how a game runs in Game Mode.
Windows.Gaming.UI	Provides ways to use Game bar, game monitoring, and game chat.
Windows.Gaming.XboxLive.Storage	Contains GameSaveProvider and GameSaveContainer classes used to create, save, enumerate, and load Xbox Live game saves.
Windows.Globalization	Provides globalization support for language profiles, geographic regions, and international calendars.
Windows.Globalization.Collation	Provides read-only lists of character groupings.
Windows.Globalization.DateTimeFormatting	Enables formatting of dates and times.
Windows.Globalization.Fonts	Provides classes that allow apps to select language-appropriate and script-appropriate fonts for text display.
Windows.Globalization.NumberFormatting	Provides classes for formatting currencies, decimal numbers, percent values, and permille values, based on the user's languages and geographic region.
Windows.Globalization.PhoneNumberFormatting	Provides phone number formatting, validation, and matching with wide global support.
Windows.Graphics	Provides basic data types that contain info on how to draw graphics. These data structs are commonly used to define how large surfaces are drawn when using the CompositionVirtualDrawingSurface class.
Windows.Graphics.Capture	Enables apps to capture environments, application windows, and displays in a secure, easy to use way with the use of a system picker UI control.

Namespace	Description
Windows.Graphics.DirectX	Specifies pixel formats, and other enumerated constants, for use with Windows Runtime Direct3D 11 interop surfaces.
Windows.Graphics.DirectX.Direct3D11	Provides Windows Runtime projections of Microsoft Direct3D APIs to enable Windows Runtime applications and components to exchange Direct3D surfaces and devices.
Windows.Graphics.Display	Contains components to determine aspects of a physical display.
Windows.Graphics.Display.Core	Gets info about a display and configures their settings to support the display of your content.
Windows.Graphics.Effects	Windows.Graphics interfaces.
Windows.Graphics.Holographic	Contains classes that let apps display holograms in the world around you.
Windows.Graphics.Imaging	Enables the decoding, editing, and encoding of image files.
Windows.Graphics.Printing	Contains the classes, interfaces, structures and other components that are used by the print driver and by the Windows print system to create and manipulate documents.
Windows.Graphics.Printing.OptionDetails	Extends the basic print task options to enable you to have greater control over the printing experience that you present to your users.
Windows.Graphics.Printing.PrintSupport	Provides APIs for working with sessions of a print support settings UI.
Windows.Graphics.Printing.PrintTicket	This namespace provides Windows Runtime classes that serve as wrappers over the Print Schema COM interfaces. This allows for the easy configuration of print job instructions from within a UWP print workflow app.
Windows.Graphics.Printing.ProtectedPrint	Provides APIs related to Windows protected print mode. Windows protected print mode exclusively uses the Windows modern print stack which provides additional print security benefits on PCs. Enabling Windows protected print mode is highly recommended.
Windows.Graphics.Printing.Workflow	This namespace provides the functionality for writing a print workflow app. Print workflow apps are registered to print-capable source applications. They are launched in the background after the source application has elected to print something but before the print data is sent to the printer. Workflow apps are able to alter both the print ticket (printer device settings) and the actual content to be

Namespace	Description
	printed, and they can optionally expose this functionality to the user by launching a foreground app midway through the process. This allows for a greater degree of customization of the printing experience.
Windows.Graphics.Printing3D	Provides APIs for working with 3D printers.
Windows.Management	Provides capabilities to force a sync from an MDM device to the server. This MDM sync protocol is based on the Open Mobile Alliance - Device Management standard.
Windows.Management.Core	Provides core functionality for managing apps.
Windows.Management.Deployment	Provides functionality for managing app packages.
Windows.Management.Deployment.Preview	Provides functionality for finding installed desktop apps, and displaying their names and versions.
Windows.Management.Policies	Provides the ability for Universal Windows Platform (UWP) apps to query the current Mobile Device Management (MDM) policies on a device.
Windows.Management.Update	The types in this namespace allow users to opt in or out of receiving Windows Insider Preview builds on their device. The APIs are intended to be used by IoT original equipment manufacturers (OEMs).
Windows.Management.Workplace	Provides functionality for managing apps in an enterprise.
Windows.Media	Provides classes for creating and working with media such as photos, audio recordings and videos.
Windows.Media.AppBroadcasting	Provides APIs for getting the broadcasting status for the current app. This includes determining whether the app can currently begin broadcasting and, if not, details about why broadcasting can't be started as well as the ability to launch the app broadcasting system setup UI.
Windows.Media.AppRecording	Provides APIs for recording the audio and video content of an app to a file, saving screenshots of app content to image files, and querying the current availability of app recording and screenshot functionality.
Windows.Media.Audio	Provides APIs for routing, mixing, and processing audio.
Windows.Media.Capture	Provides classes for the capture of photos, audio recordings, and videos.
Windows.Media.Capture.Core	Provides classes for the capture of variable photo sequences.

Namespace	Description
Windows.Media.Capture.Frames	This namespace provides APIs for obtaining media frames from one or more MediaFrameSource objects representing capture devices.
Windows.Media.Casting	This namespace lets an app to cast or send particular media content to a second device and is meant to be device-protocol agnostic. Today, the APIs work with Miracast, DLNA, and Bluetooth, and support images, audio, and video from HTML Tags, ImageElement and MediaElement.
Windows.Media.ClosedCaptioning	Provides APIs for retrieving the closed caption formatting settings that the user can set through the system's closed captioning settings page.
Windows.Media.ContentRestrictions	Provides classes that define Family Safety settings for a Windows user.
Windows.Media.Control	Collection of APIs that allow control of different playback instances.
Windows.Media.Core	Contains core media APIs that are used by several different features related to media playback.
Windows.Media.Core.Preview	Provides media APIs that may be deprecated in future releases.
Windows.Media.Devices	Accesses and controls devices for audio, video, and communications.
Windows.Media.Devices.Core	Accesses and controls the device for variable photo sequences.
Windows.Media.DialProtocol	The DIAL protocol (**DI** scover **A** nd **L** aunch) lets people launches media apps from a primary device, like a phone or tablet, to a remote device like a television. See http://www.dial-multiscreen.org for more information in the industry-standard protocol.
Windows.Media.Editing	Provides functionality for editing, previewing, and saving edited media clips.
Windows.Media.Effects	Contains classes for discovering the audio processing chain on a device.
Windows.Media.FaceAnalysis	Provides APIs for face detection in bitmaps or video frames.
Windows.Media.Import	The Windows.Media.Import namespace provides APIs for discovering and importing photos and videos from devices including cameras, mass storage devices, phones, and other devices that support photo import.
Windows.Media.MediaProperties	Provides classes and other programming elements needed to create properties for media formats.

Namespace	Description
Windows.Media.Miracast	Namespace for Miracast-related functionality.
Windows.Media.Ocr	Provides optical character recognition (OCR) API for reading text from images.
Windows.Media.Playback	Provides functionality for playing media files from foreground and background tasks.
Windows.Media.Playlists	Provides classes for creating and saving media playlists for playback.
Windows.Media.PlayTo	Supports the streaming of audio, video, and images to remote devices.
Windows.Media.Protection	Provides types that provide content protection management for media content.
Windows.Media.Protection.PlayReady	Provides types that provide PlayReady content protection management for media content.
Windows.Media.Render	Provides classes for managing media rendering.
Windows.Media.SpeechRecognition	Enables speech recognition for command and control within Windows Runtime app.
Windows.Media.SpeechSynthesis	Provides support for initializing and configuring a speech synthesis engine (or voice) to convert a text string to an audio stream, also known as text-to-speech (TTS). Voice characteristics, pronunciation, volume, pitch, rate or speed, emphasis, and so on are customized through Speech Synthesis Markup Language (SSML) Version 1.1.
Windows.Media.Streaming.Adaptive	Provides the ability to playback multimedia content that is delivered using various adaptive streaming protocols. You can playback Http Live Streaming (HLS) or Dynamic Adaptive Streaming over HTTP (DASH) content.
Windows.Media.Transcoding	Contains classes for transcoding audio and video files.
Windows.Networking	Provides access to hostnames and endpoints used by network apps.
Windows.Networking.BackgroundTransfer	Enables advanced download and upload transfer capabilities within an app. For an overview of Background Transfer capabilities, see Transferring data in the background. Download the Background transfer sample for a code example.
Windows.Networking.Connectivity	Retrieves connection profiles that provide connectivity, usage, and data plan information for established network connections. This data can be used by a connected application to take appropriate action when operating in challenging network

Namespace	Description
	scenarios or abiding by data plan limits set by a network provider.
Windows.Networking.NetworkOperators	Contains classes that support Mobile Broadband Account management.
Windows.Networking.Proximity	Contains classes that support connections between devices that are within close range.
Windows.Networking.PushNotifications	Contains classes and interfaces that encapsulate push notifications.
Windows.Networking.ServiceDiscovery.Dnssd	Supports the registration and discovery of services that advertise themselves using DNS Service Discovery (DNS-SD).
Windows.Networking.Sockets	Provides sockets and WebSockets classes to use for network communications and classes for real-time network notifications received in the background for UWP apps.
Windows.Networking.Vpn	The classes in this namespace enable you to create, edit, and manage virtual private network (VPN) connections, and to write plugins for controlling a VPN connection using the Windows VPN platform.
Windows.Networking.XboxLive	This API is not available to all apps. Unless your developer account is specially provisioned by Microsoft, calls to these APIs will fail at runtime.
Windows.Perception	Contains classes for perceiving the user's surroundings, letting apps locate and reason about the device relative to the surfaces and holograms around the user.
Windows.Perception.Automation.Core	Defines support types for the Perception Automation infrastructure.
Windows.Perception.Automation.Monitoring	Defines support types for the Perception Automation Monitoring infrastructure.
Windows.Perception.People	Contains classes that describe people.
Windows.Perception.Spatial	Contains classes for reasoning about the spatial relationships within the user's surroundings.
Windows.Perception.Spatial.Preview	Contains classes that let apps track spatial nodes, allowing the user to reason about places and things in their surroundings.
Windows.Perception.Spatial.Surfaces	Contains classes that describe the surfaces observed in the user's surroundings and their triangle meshes.
Windows.Phone.ApplicationModel	Provides an app with access to core system functionality and run-time information about its app package.

Namespace	Description
Windows.Phone.Devices.Notification	The Windows.Phone.Devices.Notification namespace provides a class that you can use to notify the user by vibrating the phone.
Windows.Phone.Devices.Power	The Windows.Phone.Devices.Power namespace contains a class that provides information about the status of the phone's battery.
Windows.Phone.Management.Deployment	Provides classes to query the applications installed on a phone and manage the installation state of application packages.
Windows.Phone.Media.Devices	The Windows.Phone.Media.Devices namespace provides functionality that lets Voice over IP applications route audio to endpoints other than the speakerphone.
Windows.Phone.Networking.Voip	
Windows.Phone.Notification.Management	Provides members that allow accessories to interact with the phone.
Windows.Phone.PersonalInformation	Provides APIs for managing a custom contact store.
Windows.Phone.PersonalInformation.Provisioning	This namespace provides APIs that enable authorized entities to provision personal information.
Windows.Phone.Speech.Recognition	Enables speech recognition for Windows Phone apps.
Windows.Phone.StartScreen	Contains classes that encapsulate tile, toast, and badge notifications for dual sim devices.
Windows.Phone.System	Provides information related to system functionality.
Windows.Phone.System.Power	The Windows.Phone.System.Power namespace provides classes to indicate whether the phone's battery is in power-saving mode.
Windows.Phone.System.Profile	Contains profile information about the phone.
Windows.Phone.System.UserProfile.GameServices.Core	Provides access to game services.
Windows.Phone.UI.Core	Provides support for processing input events on the core window.
Windows.Phone.UI.Input	Provides support for user input mechanisms for the phone.
Windows.Security.Authentication.Identity	Contains classes for interacting with enterprise key credential registration.
Windows.Security.Authentication.Identity.Core	This API is for internal use only and should not be used in your code.

Namespace	Description
Windows.Security.Authentication.Identity.Provider	Supports unlocking Windows with a companion device.
Windows.Security.Authentication.OnlineId	Enables both apps and desktop programs to use Live to authenticate users by using their Microsoft account. After authentication, apps can get back a JSON token with an ID that can be used for that user with your application.
Windows.Security.Authentication.Web	Enables apps to integrate with online web services without exposing a user's credentials. The web authentication broker provides a set of APIs and infrastructure for apps to use Internet authentication and authorization protocols like OAuth and OpenID.
Windows.Security.Authentication.Web.Core	Contains core methods for obtaining tokens from web account providers.
Windows.Security.Authentication.Web.Provider	Contains classes for managing web accounts and authentication requests as an online identity provider.
Windows.Security.Authorization.AppCapabilityAccess	AppCapabilityAccess will allow for apps to request and monitor their access to Capability protected resources at runtime.
Windows.Security.Credentials	Provides a common way to securely store and manage your passcodes, passphrases, and other identification information.
Windows.Security.Credentials.UI	Enables you to use credentials without the complexity of the underlying operations, providing a uniform experience for credentials that reduces credential prompting to an absolute minimum. This namespace collects various types of credentials from users, stores them on the local computer, and presents them to web services and proxies by using existing authentication models.
Windows.Security.Cryptography	Contains a class that can be used for encoding and decoding data, generating random numbers, and converting between byte arrays and buffers.
Windows.Security.Cryptography.Certificates	Contains types that you can use to create certificate requests and install certificate responses.
Windows.Security.Cryptography.Core	Contains classes that enable you to create cryptographic keys, to hash and sign content, create message authentication codes, and to encrypt or decrypt data.
Windows.Security.Cryptography.DataProtection	Contains a class that enables you to easily encrypt and decrypt static data or a data stream.
Windows.Security.DataProtection	The Data Protection API enables data protection for all users and applications regardless of the presence

Namespace	Description
	of Windows Information Protection (WIP) policy.
Windows.Security.EnterpriseData	Contains classes that support Windows Information Protection (WIP). For the full developer picture of how WIP relates to files, buffers, the clipboard, networking, background tasks, and data protection under lock, see Windows Information Protection.
Windows.Security.ExchangeActiveSyncProvisioning	Enables an app for mail to evaluate and apply the Exchange Active Synchronization (EAS) policies.
Windows.Security.Isolation	**Deprecated.** Namespace for Windows Security Isolation APIs.
Windows.Services.Cortana	Provides types and members for accessing and managing the information an app shares with the Cortana service.
Windows.Services.Maps	The Windows.Services.Maps namespace contains classes that provide location, routing, and geocoding services.
Windows.Services.Maps.Guidance	Provides APIs for turn-by-turn guided navigation.
Windows.Services.Maps.LocalSearch	Provides APIs to search for business and places. Searches are performed against data that has been cached locally on the device.
Windows.Services.Maps.OfflineMaps	Contains classes that you can use to find and download map packages for offline use.
Windows.Services.Store	Provides types and members you can use to access and manage Microsoft Store-related data for the current app. For example, you can use this namespace to get Microsoft Store listing and license info for the current app, purchase the current app or products that are offered by the app, or download and install package updates for the app.
Windows.Services.TargetedContent	The Targeted Content namespace provides applications with the ability to subscribe to and access cloud-delivered, subscription-specific content and assets. This content is cached locally so that it is available while a device is offline. Subscription content can be updated periodically when new content is available.
Windows.Storage	Provides classes for managing files, folders, and application settings.
Windows.Storage.AccessCache	Enables and manages access to the most recently used (MRU) list of storage items, and to the list of storage items that the application has saved for efficient future access.

Namespace	Description
Windows.Storage.BulkAccess	Provides classes for searching, accessing, managing, and retrieving information about files and folders in the storage system.
Windows.Storage.Compression	Provides a unified interface to the compression features included in Windows that frees developers from responsibility for managing block sizes, compression parameters, and other details that the native compression API requires.
Windows.Storage.FileProperties	Provides access to the properties of a file.
Windows.Storage.Pickers	Provides classes for creating and managing user interface (UI) elements that let the user browse files, choose files to open, and to choose the name, extension, and location when storing files.
Windows.Storage.Pickers.Provider	Provides support for apps that offer files through the file picker by participating in the File Open Picker contract, and for apps that act as a save location for files through file picker by participating in the File Save Picker contract or the Cached File Updater contract.
Windows.Storage.Provider	Allows apps to provide real-time updates to files through the Cached File Updater contract.
Windows.Storage.Search	Provides classes for enumerating content in storage and performing Advanced Query Syntax (AQS) searches on the content.
Windows.Storage.Streams	Provides support for reading to and writing from sequential and random access streams.
Windows.System	Enables system functionality such as launching apps, obtaining information about a user, and memory profiling.
Windows.System.Diagnostics	Provides diagnostic information about the system and running processes.
Windows.System.Diagnostics.DevicePortal	Provides functionality for implementing a custom appx plugin for the Windows Device Portal.
Windows.System.Diagnostics.Telemetry	Provides functionality for registering clients with the Connected User Experience and Telemetry component.
Windows.System.Diagnostics.TraceReporting	This namespace represents the Connected User Experience and Telemetry component that manages the telemetry escalations.
Windows.System.Display	Enables an application to request that a device's display remain on even if there is no user activity for an extended period of time.

Namespace	Description
Windows.System.Implementation.FileExplorer	For internal use only.
Windows.System.Implementation.Holographic	Contains types used in the implementation of Windows components relating to the holographic shell.
Windows.System.Inventory	Provides a list of installed Win32 apps available to the current user. This includes apps installed for all users as well as only for the current user.
Windows.System.Power	Provides power and energy usage APIs.
Windows.System.Power.Diagnostics	Provides debugging APIs for computing your app's total energy usage in real time. We recommend using these APIs only for debugging.
Windows.System.Preview	Enables apps running on two panel devices to get information about the posture of the device.
Windows.System.Profile	Enables apps to get information about the system.
Windows.System.Profile.SystemManufacturers	Provides info for hardware developers and OEMs.
Windows.System.RemoteDesktop	Enables an application to obtain information about Remote Desktop sessions.
Windows.System.RemoteDesktop.Input	Provides the ability for Remote Desktop client applications to enable text input virtualization.
Windows.System.RemoteDesktop.Provider	Provides APIs for implementing remote desktop client apps.
Windows.System.RemoteSystems	Provides an app with the ability to discover and communicate with other devices. This API is part of Project Rome which allows you to create a multi-platform user experience.
Windows.System.Threading	Enables an application to use the thread pool to run work items.
Windows.System.Threading.Core	Creates work items that run in response to named events and semaphores. Also preallocates resources for work items that must be guaranteed the ability to run, even in circumstances of heavy (or full) resource allocation.
Windows.System.Update	Support for interactive control of system updates.
Windows.System.UserProfile	Enables apps get information about users, set globalization preferences, and control the lock screen image.
Windows.UI	Provides an app with access to core system functionality and run-time information about its UI.
Windows.UI.Accessibility	Provides support for working with accessibility

Namespace	Description
	devices such as screen readers and magnifiers.
Windows.UI.ApplicationSettings	Provides classes that allow developers to define the app settings that appear in the settings pane of the Windows shell. The settings pane provides a consistent place for users to access app settings.
Windows.UI.Composition	Provides APIs for user interface composition.
Windows.UI.Composition.Core	Provides core APIs for user interface composition.
Windows.UI.Composition.Desktop	Provides APIs for user interface composition.
Windows.UI.Composition.Diagnostics	Provides APIs for user interface composition diagnostics and debugging.
Windows.UI.Composition.Effects	Contains classes that are specific to creating effects for User Interactions. These effects are specific to composition.
Windows.UI.Composition.Interactions	Provides functionality that enables input driven animations in the visual layer.
Windows.UI.Composition.Scenes	Provides APIs for loading 3D models and composing 3D scenes.
Windows.UI.Core	Provides the basic user interface window components, including core input events and event dispatch.
Windows.UI.Core.AnimationMetrics	The classes and their properties that are exposed through this namespace retrieve the raw scaling, translation, and opacity parameters used in Windows animations
Windows.UI.Core.Preview	Provides preview APIs for various basic user interface window components.
Windows.UI.Core.Preview.Communications	Allows Teams or any other developer to write a plug-in VTC app for Surface Hub.
Windows.UI.Input	Provides support for the Windows input system. This includes:
Windows.UI.Input.Core	Provides support for processing input on a background thread.
Windows.UI.Input.Inking	Provides input, processing, and management support for Windows Ink apps.
Windows.UI.Input.Inking.Analysis	Provides analysis of ink stroke input for Windows Ink apps.
Windows.UI.Input.Inking.Core	Overrides default InkPresenter run-time behavior to provide advanced input, processing, and management support for Windows Ink apps.

Namespace	Description
Windows.UI.Input.Inking.Preview	Provides preview APIs for Windows Ink apps.
Windows.UI.Input.Preview	Provides Windows input preview APIs.
Windows.UI.Input.Preview.Injection	Provides support for programmatically generating and automating input from a variety of devices such as keyboard, mouse, touch, pen, and gamepad.
Windows.UI.Input.Spatial	Contains classes that let apps react naturally to your gaze, hand gestures, motion controllers and voice in the world around you.
Windows.UI.Notifications	Contains classes that encapsulate tile, toast, and badge notifications.
Windows.UI.Notifications.Management	Provides APIs for reading and managing notifications.
Windows.UI.Notifications.Preview	Provides preview APIs for Windows notification features.
Windows.UI.Popups	Provides support for context menu and message dialogs.
Windows.UI.Shell	Provides support for connecting apps with native elements of the Windows UI through the Windows Shell.
Windows.UI.StartScreen	Provides ways for apps to integrate with the Start screen, such as the creation of secondary tiles and the customization of jump lists.
Windows.UI.Text	Provides support for working with textual content.
Windows.UI.Text.Core	Provides types for accessing the Windows core text APIs and the text input server. Windows core text is a client-server system that centralizes the processing of keyboard input into a single server. You can use it to manipulate the edit buffer of your custom text input control; the text input server ensures that the contents of your text input control and the contents of its own edit buffer are always in sync via an asynchronous communication channel between the application and the server.
Windows.UI.UIAutomation	Provides support for WinRT automation clients.
Windows.UI.UIAutomation.Core	Provides types for accessing Windows core UI Automation APIs.
Windows.UI.ViewManagement	Provides support for handling and managing the various views associated with the active Universal Windows Platform (UWP) app.
Windows.UI.ViewManagement.Core	Provides support for handling and managing CoreInputView objects associated with the active

Namespace	Description
	foreground application.
Windows.UI.WebUI	Provides APIs that are specific to apps written in HTML, JavaScript, and CSS.
Windows.UI.WebUI.Core	Provides access to the command bar for apps written in HTML, JavaScript, and CSS.
Windows.UI.WindowManagement	Provides support for handling and managing the windows associated with the active app.
Windows.UI.WindowManagement.Preview	Provides preview APIs for handling and managing the windows associated with the active app.
Windows.UI.Xaml	Provides general framework APIs and application model APIs, and a variety of support classes that are commonly used by many different feature areas.
Windows.UI.Xaml.Automation	Defines the support types for Microsoft UI Automation infrastructure. These support types are for implementing Microsoft UI Automation support for a custom control. Also, AutomationProperties provides accessibility info as part of a XAML UI definition.
Windows.UI.Xaml.Automation.Peers	Contains automation peers for controls and base classes.
Windows.UI.Xaml.Automation.Provider	Defines provider patterns that are implemented by automation peers and recognized by automation clients.
Windows.UI.Xaml.Automation.Text	Contains enumerations that report specifics about text automation patterns.
Windows.UI.Xaml.Controls	Provides UI controls and classes that support existing and custom UWP controls.
Windows.UI.Xaml.Controls.Maps	Defines classes for the Map control. Also defines tile sources and elements that can be placed on a map.
Windows.UI.Xaml.Controls.Primitives	Defines classes that represent the component parts of UI controls, or otherwise support the control composition model. Also defines interfaces for control patterns such as snapping and selection.
Windows.UI.Xaml.Core.Direct	Provides XamlDirect APIs that allow middleware to access most of Xaml at a more primitive level, achieving better CPU and working set performance.
Windows.UI.Xaml.Data	Defines the data binding infrastructure for apps.
Windows.UI.Xaml.Documents	Provides basic text and document model classes.
Windows.UI.Xaml.Hosting	Provides services that are relevant for XAML design surfaces that are hosted in a larger application.

Namespace	Description
Windows.UI.Xaml.Input	Defines the input and input event infrastructure for apps and UI elements.
Windows.UI.Xaml.Interop	Provides basic types for cross-framework integration and interoperation. Notably, this includes interfaces and support types that enable binding patterns.
Windows.UI.Xaml.Markup	Provides XAML language support API for UWP app and the Windows Runtime.
Windows.UI.Xaml.Media	Provides basic media support, graphics primitives, and brush-drawing API.
Windows.UI.Xaml.Media.Animation	Provides animation and storyboard API for transition animations, visual states, or animated UI components.
Windows.UI.Xaml.Media.Imaging	Provides types related to imaging and obtaining resources and source files for bitmap images.
Windows.UI.Xaml.Media.Media3D	Contains types that support matrix/perspective transformation.
Windows.UI.Xaml.Navigation	Provides types that support navigation events as initiated by the Page and Frame classes.
Windows.UI.Xaml.Printing	Provides a basic printing support framework for applications.
Windows.UI.Xaml.Resources	Contains CustomXamlResourceLoader, which enables loading of custom resource dictionaries as sources.
Windows.UI.Xaml.Shapes	Defines basic shapes that are intended for decorative rendering or for compositing non-interactive parts of controls.
Windows.Web	Provides information on errors resulting from web service operations.
Windows.Web.AtomPub	Enables HTTP CRUD access to Web resources using the AtomPub protocol. The AtomPub sample demonstrates this implementation.
Windows.Web.Http	Provides a modern HTTP client API for UWP app.
Windows.Web.Http.Diagnostics	Provides a diagnostic facility for tracing and profiling web traffic for applications build on HTTP protocols.
Windows.Web.Http.Filters	Provides classes to send HTTP requests and an interface to create filters to target HTTP and REST services in UWP app.
Windows.Web.Http.Headers	Provides support for HTTP headers used by the Windows.Web.Http namespace for UWP app that target HTTP services.

Namespace	Description
Windows.Web.Syndication	Manages syndication feeds.
Windows.Web.UI	This namespace allows Win32 apps to host the UWP WebView control Out of Process and take full advantage of HTML5. Most of these APIs are a reflection of the Windows.UI.Xaml.Controls.WebView control.
Windows.Web.UI.Interop	This namespace allows Win32 apps to host the UWP WebView control Out of Process and take full advantage of HTML5.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get product support

Windows UWP Namespaces

Article10/03/2024

This section provides detailed information about the Windows Runtime (WinRT) APIs. These APIs are for UWP app development on Windows 10.

Expand table

Namespace	Description
Microsoft.UI.Xaml.Automation.Peers	Defines support types for the Microsoft UI Automation infrastructure.
Microsoft.UI.Xaml.Controls	Provides UI controls and classes for creating custom controls.
Microsoft.UI.Xaml.Controls.AnimatedVisuals	Defines the built-in animated visuals that can be played by an AnimatedIcon object.
Microsoft.UI.Xaml.Controls.Primitives	Defines the components that comprise WinUI controls, or otherwise support the control composition model.
Microsoft.UI.Xaml.Core.Direct	Provides a way for middleware authors to access low-level, high-performance XAML APIs and achieve better CPU and working set performance.
Microsoft.UI.Xaml.Media	Provides brushes, basic media support, and graphics primitives.
Microsoft.UI.Xaml.XamlTypeInfo	Provides support for the XAML schema.

Programming reference for the Win32 API

The Win32 API reference documentation is presented in several different views. You can browse a list of popular technologies on this page, or you can browse the full list of technologies in the table of contents. To browse all of the headers, see the list at the bottom of the table of contents.

Get started

OVERVIEW

Build desktop Windows apps using the Win32 API

DOWNLOAD

Development tools

Windows code samples

User Interface and desktop

REFERENCE

Windows controls

Windows and messages

Menus and other resources

Windows Shell

Accessibility features

Internationalization

Graphics and gaming

REFERENCE

Direct2D

Direct3D 11 Graphics

Direct3D 12 Graphics

DirectML

DXGI

Windows GDI

GDI+

Windows Imaging Component

Audio and video

REFERENCE

Microsoft Media Foundation

Windows Multimedia

XAudio2 APIs

Core Audio APIs

Audio Devices DDI Reference

Data access and storage

REFERENCE

Data access and storage

Backup

Background Intelligent Transfer Service

Cloud Filter API

Data Exchange

Structured storage

Virtual Storage

Devices

REFERENCE

Kernel-mode driver reference

Device and driver installation reference

Storage driver DDI reference

USB driver reference

Display devices reference

Human Interface Devices reference

Print DDI reference

Networking and internet

REFERENCE

Bluetooth

HTTP Server API

IP Helper

Network Management

Remote Procedure Call (RPC)

Windows HTTP Services (WinHTTP)

Windows Internet

Windows Sockets 2

Security and identity

REFERENCE

Security and identity

Network Access Protection

Network Policy Server

Parental controls

Windows Biometric Framework

TPM Base Services

Diagnostics

REFERENCE

Event tracing

Network Diagnostics Framework

Performance counters

Application Recovery and Restart

TraceLogging

Windows Event Collector

Windows Error Reporting

Application installation

REFERENCE

Application installation and servicing

Packaging and deployment Windows 10 apps

Developer licensing

Restart Manager

System services

REFERENCE

Component Object Model (COM)

COM+ (Component Services)

Microsoft Interface Definition Language (MIDL)

Compression API

Activity Coordinator

Hardware Requirement Evaluator (HWREQCHK)

Windows App SDK namespaces

Article03/19/2025

This section provides detailed information about the Windows Runtime (WinRT) APIs in the Windows App SDK.

Expand table

Namespace	Description
Microsoft.ApplicationModel.Resources	Provides simplified access to app resources, such as strings, that are defined using basic naming conventions.
Microsoft.Graphics.DirectX	Specifies pixel formats, and other enumerated constants, for use with Windows Runtime Direct3D 11 interop surfaces.
Microsoft.Graphics.Display	Contains components to determine aspects of a physical display.
Microsoft.Security.Authentication.OAuth	The `Microsoft.Security.Authentication.OAuth` namespace provides classes and methods for OAuth 2.0 authentication.
Microsoft.UI	Provides an app with access to core system functionality and run-time information about its UI.
Microsoft.UI.Composition	Provides APIs for user interface composition.
Microsoft.UI.Composition.Core	Provides core APIs for user interface composition.
Microsoft.UI.Composition.Diagnostics	Provides APIs for user interface composition diagnostics and debugging.
Microsoft.UI.Composition.Effects	Contains classes that are specific to creating effects for User Interactions. These effects are specific to composition.
Microsoft.UI.Composition.Interactions	Provides functionality that enables input driven animations in the visual layer.
Microsoft.UI.Composition.Scenes	Provides APIs for loading 3D models and composing 3D scenes.
Microsoft.UI.Composition.SystemBackdrops	Provides APIs for managing a CompositionBrush that defines a UI material such as Acrylic or Mica.
Microsoft.UI.Content	Provides support for defining the structure of interactive content in a Windows app through **ContentSite** (parent) and **ContentIsland** (child) objects using composition-based animations, rendering effects, user input and interactions, accessibility representation, and host state changes.
Microsoft.UI.Dispatching	Enables a queue for tasks to execute in a serial fashion on a thread.
Microsoft.UI.Input	Provides support for the Windows input system. This includes:
Microsoft.UI.Input.DragDrop	Provides support for the pointer-based, direct manipulation of an object (or data) that involves selecting that object and dragging it from one location to another location.
Microsoft.UI.Input.Interop	Provides a basic type for cross-framework input integration and inter-operation. At this time, this includes an interface and support type that expose pen device input.
Microsoft.UI.System	Windows user interface (UI) system APIs.
Microsoft.UI.Text	Provides support for working with textual content.

Namespace	Description
Microsoft.UI.Windowing	Provides support for creating and managing the windows associated with an app.
Microsoft.UI.Xaml	Provides general framework APIs for the Windows UI Library (WinUI).
Microsoft.UI.Xaml.Automation	Defines the support types for Microsoft UI Automation infrastructure. These support types are for implementing Microsoft UI Automation support for a custom control. Also, AutomationProperties provides accessibility info as part of a XAML UI definition.
Microsoft.UI.Xaml.Automation.Peers	Defines support types for the Microsoft UI Automation infrastructure.
Microsoft.UI.Xaml.Automation.Provider	Defines provider patterns that are implemented by automation peers and recognized by automation clients.
Microsoft.UI.Xaml.Automation.Text	Contains enumerations that report specifics about text automation patterns.
Microsoft.UI.Xaml.Controls	Provides UI controls and classes for creating custom controls.
Microsoft.UI.Xaml.Controls.AnimatedVisuals	Defines the built-in animated visuals that can be played by an AnimatedIcon object.
Microsoft.UI.Xaml.Controls.Primitives	Defines the components that comprise WinUI controls, or otherwise support the control composition model.
Microsoft.UI.Xaml.Data	Defines the data binding infrastructure for apps.
Microsoft.UI.Xaml.Documents	Provides basic text and document model classes.
Microsoft.UI.Xaml.Hosting	Provides services relevant to XAML design surfaces hosted in a larger application.
Microsoft.UI.Xaml.Input	Defines the input and input event infrastructure for apps and UI elements.
Microsoft.UI.Xaml.Interop	Provides basic types for cross-framework integration and interoperation. Notably, this includes interfaces and support types that enable binding patterns.
Microsoft.UI.Xaml.Markup	Provides XAML language support API for UWP app and the Windows Runtime.
Microsoft.UI.Xaml.Media	Provides brushes, basic media support, and graphics primitives.
Microsoft.UI.Xaml.Media.Animation	Provides animation and storyboard API for transition animations, visual states, or animated UI components.
Microsoft.UI.Xaml.Media.Imaging	Provides types related to imaging and obtaining resources and source files for bitmap images.
Microsoft.UI.Xaml.Media.Media3D	Contains types that support matrix/perspective transformation.
Microsoft.UI.Xaml.Navigation	Provides types that support navigation events as initiated by the Page and Frame classes.
Microsoft.UI.Xaml.Printing	Provides a basic printing support framework for applications.
Microsoft.UI.Xaml.Resources	Contains CustomXamlResourceLoader, which enables loading of custom resource dictionaries as sources.
Microsoft.UI.Xaml.Shapes	Defines basic shapes that are intended for decorative rendering or for compositing non-interactive parts of controls.

Namespace	Description
Microsoft.UI.Xaml.XamlTypeInfo	Provides support for the XAML schema.
Microsoft.Windows.ApplicationModel.Background	The **Microsoft.Windows.ApplicationModel.Background** APIs support background task registration for apps that leverage Windows App SDK.
Microsoft.Windows.ApplicationModel.Background.UniversalBGTask	The **Microsoft.Windows.ApplicationModel.Background.UniversalBGTask** APIs support background task registration for apps that leverage Windows App SDK.
Microsoft.Windows.ApplicationModel.DynamicDependency	Provides the ability for unpackaged desktop apps to load framework packages other than the Windows App SDK framework package. After the specified framework package is loaded by an unpackaged desktop app, the app can call APIs provided by the framework package.
Microsoft.Windows.ApplicationModel.Resources	Provides simplified access to app resources, such as strings, that are defined using basic naming conventions.
Microsoft.Windows.ApplicationModel.WindowsAppRuntime	Provides access to deployment information for the Windows App SDK runtime and the ability to register any missing packages for the runtime.
Microsoft.Windows.AppLifecycle	Provides APIs you can use to manage the lifecycle of your app. For example, you can control whether multiple instances of your app's process can run at the same time, and you can receive information about different kinds activations for your app.
Microsoft.Windows.AppNotifications	This namespace provides APIs for managing app notifications.
Microsoft.Windows.AppNotifications.Builder	Provides APIs for programmatically generating the XML payload for an AppNotification.
Microsoft.Windows.BadgeNotifications	Contains classes that support badge notifications.
Microsoft.Windows.Globalization	Provides globalization support (such as language profiles) for apps built with the Windows App SDK.
Microsoft.Windows.Management.Deployment	This namespace provides MSIX package management APIs comparable to those in the Windows.Management.Deployment namespace, but with additional functionality, improved developer experience, and performance optimizations.
Microsoft.Windows.Media.Capture	Provides APIs for configuring an launching the CameraCaptureUI which allows the user to capture photos, videos, or both.
Microsoft.Windows.PushNotifications	This namespace provides APIs for managing push notifications that use the Windows Push Notification Service (WNS).
Microsoft.Windows.Security.AccessControl	Contains APIs that ease and streamline named object sharing between packaged apps and Win32 applications.
Microsoft.Windows.Storage	Contains APIs in the Microsoft.Windows.Storage namespace that are available to access storage resources from Windows apps.
Microsoft.Windows.System	Contains APIs in the Microsoft.Windows.System namespace that are available to access system resources from Windows apps.
Microsoft.Windows.System.Power	Provides a set of power management APIs that allow visibility into how an app affects the device's power state, and enable the app to make intelligent decisions about resource usage. For example, an app might use this API to postpone resource-intensive background tasks while the device is running on battery power.

Namespace	Description
Microsoft.Windows.Widgets	Provides APIs related to Windows widgets.
Microsoft.Windows.Widgets.Feeds.Providers	Provides APIs for implementing a feed provider for Windows feeds.
Microsoft.Windows.Widgets.Notifications	
Microsoft.Windows.Widgets.Providers	Provides APIs for implementing a widget provider for Windows Widgets.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Programming reference for the Win32 APIs in the Windows App SDK

The Windows App SDK Win32 API reference documentation is presented in several different views. You can browse a list of popular technologies on this page, or you can browse the full list of technologies in the table of contents. To browse all of the headers, see the list at the bottom of the table of contents.

Text

OVERVIEW

Render text with DWriteCore

REFERENCE

Dwrite.h

Dwrite_1.h

Dwrite_2.h

Dwrite_3.h

Dwrite_core.h

Bootstrapper

REFERENCE

Mddbootstrap.h

Direct3D and DirectX

REFERENCE

winrtdirect3d11.h

winrtdirectxcommon.h

Dynamic Dependency

REFERENCE

Msixdynamicdependency.h

Insights

REFERENCE

windowsappruntimeinsights.h

Licensing

REFERENCE

Windowsappruntime-licensing.h

MRT Core

OVERVIEW

Manage resources with MRT Core

REFERENCE

Mrm.h

Security

REFERENCE

security.accesscontrol.h

UI Interop

REFERENCE

Microsoft.ui.interop.h

(winrt)Microsoft.ui.interop.h

microsoft.ui.input.inputcursor.interop.h

microsoft.ui.input.inputpretranslatesource.interop.h

microsoft.ui.dispatching.interop.h

Version

REFERENCE

windowsappsdk-versioninfo.h

Web content

REFERENCE

Microsoft Edge WebView2 Win32 C++ reference

WinUI COM Interop

OVERVIEW

Windows UI Library (WinUI)

REFERENCE

microsoft.ui.xaml.hosting.desktopwindowxamlsource.h

microsoft.ui.xaml.hosting.referencetracker.h

microsoft.ui.xaml.media.dxinterop.h

Microsoft.ui.xaml.window.h

xamlom.winui.h

Bootstrapper C++ API

Article • 06/03/2022

The Bootstrapper C++ API is a set of types and functions (callable only from C++) that offer an alternative way to call the Bootstrapper API.

Topics in this section

Expand table

Topic	Description
Constants for use with the Bootstrapper C++ API	The following constants (for use with the Bootstrapper C++ API) are declared in `WindowsAppSDK-VersionInfo.h`.
Microsoft::Windows::ApplicationModel namespace	Types and functions from the Bootstrapper C++ API that are in the (**Microsoft::Windows**) **ApplicationModel** namespace. For example, a class that represents a version of the Windows App SDK framework package.
Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap namespace	Types and functions from the Bootstrapper C++ API that are in the (**Microsoft::Windows::ApplicationModel::DynamicDependency**) **Bootstrap** namespace. For example, helper functions that wrap calls to the Bootstrapper API.

See also

- Bootstrapper API

Constants for use with the Bootstrapper C++ API

Article • 04/19/2023

The following constants (for use with the Bootstrapper C++ API) are declared in `WindowsAppSDK-VersionInfo.h`.

Microsoft::WindowsAppSDK::Release namespace

Expand table

Constant	Value	Description
constexpr PCWSTR Channel	[release-dependent]	The Windows App SDK release's channel; for example, L"preview", or empty string for stable.
constexpr PCWSTR FormattedVersionShortTag	[release-dependent]	The Windows App SDK release's short-form version tag, formatted for concatenation when constructing identifiers; for example, "-p2", or empty string for stable.
constexpr PCWSTR FormattedVersionTag	[release-dependent]	The Windows App SDK release's version tag, formatted for concatenation when constructing identifiers; for example, "-preview2", or empty string for stable.
constexpr uint16_t Major	[release-dependent]	The major version of the Windows App SDK release.
constexpr uint32_t MajorMinor	[release-dependent]	The major and minor version of the Windows App SDK release, encoded as a uint32_t (0xMMMMNNNN where M=major, N=minor).
constexpr uint16_t Minor	[release-dependent]	The minor version of the Windows App SDK release.
constexpr uint16_t Patch	[release-dependent]	The patch version of the Windows App SDK release.
constexpr PCWSTR VersionShortTag	[release-dependent]	The Windows App SDK release's short-form version tag; for example, L"p2", or empty string for stable.
constexpr PCWSTR VersionTag	[release-dependent]	The Windows App SDK release's version tag; for example, L"preview2", or empty string for stable.

Microsoft::WindowsAppSDK::Runtime::Identity namespace

Expand table

Constant	Value	Description
constexpr PCWSTR	L"CN=Microsoft Corporation, O=Microsoft	The Windows App SDK runtime's

Constant	Value	Description
Publisher	Corporation, L=Redmond, S=Washington, C=US"	package identity's Publisher.
constexpr PCWSTR PublisherId	L"8wekyb3d8bbwe"	The Windows App SDK runtime's package identity's PublisherId.

Microsoft::WindowsAppSDK::Runtime::Packages::Framewc namespace

Expand table

Constant	Value	Description
constexpr PCWSTR PackageFamilyName	"Microsoft.WindowsAppRuntime.1.1-preview1_8wekyb3d8bbwe"	The Windows App SDK runtime's Framework package's family name.

Microsoft::WindowsAppSDK::Runtime.Packages.DDLM.Arr namespace

Expand table

Constant	Value	Description
constexpr PCWSTR PackageFamilyName	[release-dependent]	The Windows App SDK runtime's Dynamic Dependency Lifetime Manager (DDLM) package's family name, for arm64.

Microsoft::WindowsAppSDK::Runtime.Packages.DDLM.X64 namespace

Expand table

Constant	Value	Description
constexpr PCWSTR PackageFamilyName	[release-dependent]	The Windows App SDK runtime's Dynamic Dependency Lifetime Manager (DDLM) package's family name, for x64.

Microsoft::WindowsAppSDK::Runtime.Packages.DDLM.X86 namespace

Expand table

Constant	Value	Description
constexpr PCWSTR PackageFamilyName	[release-dependent]	The Windows App SDK runtime's Dynamic Dependency Lifetime Manager (DDLM) package's family name, for x86.

Microsoft::WindowsAppSDK::Runtime::Packages::Main namespace

Expand table

Constant	Value	Description
constexpr PCWSTR PackageFamilyName	L"MicrosoftCorporationII.WinAppRuntime.Main.1.1-p1_8wekyb3d8bbwe"	The Windows App SDK runtime's Main package's family name.

Microsoft::WindowsAppSDK::Runtime::Packages::Singleton namespace

Expand table

Constant	Value	Description
constexpr PCWSTR PackageFamilyName	L"Microsoft.WindowsAppRuntime.Singleton-preview1_8wekyb3d8bbwe"	The Windows App SDK runtime's Singleton package's family name.

Microsoft::WindowsAppSDK::Runtime::Version namespace

Expand table

Constant	Value	Description
constexpr uint16_t Build	[release-dependent]	The build version of the Windows App SDK runtime; for example, 804.
constexpr PCWSTR DotQuadString	[release-dependent]	The version of the Windows App SDK runtime, as a string (const wchar_t*); for example, L"1000.446.804.0".
constexpr uint16_t Major	[release-dependent]	The major version of the Windows App SDK runtime; for example, 1000.
constexpr uint16_t Minor	[release-dependent]	The minor version of the Windows App SDK runtime; for example, 446.
constexpr uint16_t Revision	[release-dependent]	The revision version of the Windows App SDK runtime; for example, 0.
constexpr uint64_t UInt64	[release-dependent]	The version of the Windows App SDK runtime, as a uint64l for example, 0x03E801BE03240000.

Requirements

Minimum supported SDK: Windows App SDK version 1.1

Namespace: Microsoft::WindowsAppSDK

Header: WindowsAppSDK-VersionInfo.h

See also

- Bootstrapper C++ API

Microsoft::Windows::ApplicationModel namespace

Article • 06/03/2022

Types and functions from the Bootstrapper C++ API that are in the **Microsoft::Windows::ApplicationModel** namespace. For example, a class that represents a version of the Windows App SDK framework package.

Types in the Microsoft::Windows::ApplicationModel namespace

Expand table

Type	Description
PackageVersion class	Represents a version of the Windows App SDK framework package.

See also

Bootstrapper C++ API

PackageVersion class

Article • 04/19/2023

Represents a version of the Windows App SDK framework package.

Syntax

```
C++

class PackageVersion : public PACKAGE_VERSION;
```

Requirements

Minimum supported SDK: Windows App SDK version 1.1

Namespace: Microsoft::Windows::ApplicationModel

Header: MddBootstrap.h

Constructors

Expand table

Constructor	Description
PackageVersion constructor	Initializes a new instance of the **PackageVersion** class.

Member functions

Expand table

Function	Description
PackageVersion::ToString function	Retrieves the version as a **std::wstring**.
PackageVersion::ToVersion function	Retrieves the version as a **uint64_t**.

PackageVersion::PackageVersion constructor

Initializes a new instance of the **PackageVersion** class.

Syntax

```
C++
```

```
PackageVersion();
PackageVersion(uint16_t major, uint16_t minor = 0, uint16_t build = 0,
uint16_t revision = 0);
PackageVersion(uint64_t version);
```

Parameters

`major` A **uint16_t** value representing the `major` position of a `major.minor.build.revision` sequence.

`minor` An optional **uint16_t** value representing the `minor` position of a `major.minor.build.revision` sequence. Defaults to 0.

`build` An optional **uint16_t** value representing the `build` position of a `major.minor.build.revision` sequence. Defaults to 0.

`revision` An optional **uint16_t** value representing the `revision` position of a `major.minor.build.revision` sequence. Defaults to 0.

`version` A `major.minor.build.revision` sequence encoded as a (little-endian) **uint64_t** (so that the first 16 bits contain the revision, and so on).

PackageVersion::ToString function

Retrieves the version as a **std::wstring**.

Syntax

```
C++
```

```
std::wstring ToString() const;
```

Return value

The `major.minor.build.revision` sequence encoded as a string (**std::wstring**) in the format "12345.12345.12345.12345" + null-terminator.

PackageVersion::ToVersion function

Retrieves the version as a **uint64_t**.

Syntax

```
C++
```

```
uint64_t ToVersion() const;
```

Return value

The `major.minor.build.revision` sequence encoded as a (little-endian) **uint64_t** (so that the first 16 bits contain the revision, and so on).

See also

- Microsoft::Windows::ApplicationModel namespace

Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap namespace

Article • 06/03/2022

Types and functions from the Bootstrapper C++ API that are in the **Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap** namespace. For example, helper functions that wrap calls to the Bootstrapper API.

Functions in theMicrosoft::Windows::ApplicationModel::DynamicDependency:: namespace

Expand table

Function	Description
Initialize function	Calls MddBootstrapInitialize to initialize the calling process to use the specified version of the Windows App SDK's framework package. If the call fails, throws an exception.
InitializeFailFast function	Calls MddBootstrapInitialize to initialize the calling process to use the specified version of the Windows App SDK's framework package. If the call fails, aborts the process (via **std::abort**).
InitializeNoThrow function	Calls MddBootstrapInitialize to initialize the calling process to use the specified version of the Windows App SDK's framework package. If the call fails, returns a failure **HRESULT**.

See also

Bootstrapper C++ API

Initialize function

Article • 04/19/2023

Calls MddBootstrapInitialize to initialize the calling process to use the specified version of the Windows App SDK's framework package. If the call fails, throws an exception.

Syntax

C++

```
inline auto Initialize(
    uint32_t majorMinorVersion = WINDOWSAPPSDK_RELEASE_MAJORMINOR,
    PCWSTR versionTag = WINDOWSAPPSDK_RELEASE_VERSION_TAG_W,
    PackageVersion minVersion = WINDOWSAPPSDK_RUNTIME_VERSION_UINT64)
```

Parameters

`majorMinorVersion` See *majorMinorVersion* in MddBootstrapInitialize. Defaults to WINDOWSAPPSDK_RELEASE_MAJORMINOR.

`versionTag` See *versionTag* in MddBootstrapInitialize. Defaults to WINDOWSAPPSDK_RELEASE_VERSION_TAG_W.

`minVersion` See *minVersion* in MddBootstrapInitialize. Defaults to WINDOWSAPPSDK_RUNTIME_VERSION_UINT64.

Return value

On success, returns a resource acquisition is initialization (RAII) object which, when it goes out of scope, undoes the changes that were made by the call to MddBootstrapInitialize.

Example

C++

```
#include <windows.h>

#include <winrt\base.h>

#include <WindowsAppSDK-VersionInfo.h>
#include <MddBootstrap.h>
```

```
#include <iostream>

namespace MddBootstrap {using namespace
    ::Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap; }

int main()
{
    try
    {
        auto mddBootstrapCleanup{ MddBootstrap::Initialize() };
        // Do work here.
    }
    catch (const winrt::hresult_error& ex)
    {
        const auto hr{ ex.code() };
        std::cout << "Error 0x" << std::hex << hr << " in Bootstraper
initialization";
        return hr;
    }
    return 0;
}
```

Requirements

Minimum supported SDK: Windows App SDK version 1.1

Namespace: Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap

Header: MddBootstrap.h

See also

- Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap namespace
- MddBootstrapInitialize

InitializeFailFast function

Article • 04/19/2023

Calls MddBootstrapInitialize to initialize the calling process to use the specified version of the Windows App SDK's framework package. If the call fails, aborts the process (via **std::abort**).

Syntax

C++

```
inline auto InitializeFailFast(
    uint32_t majorMinorVersion = WINDOWSAPPSDK_RELEASE_MAJORMINOR,
    PCWSTR versionTag = WINDOWSAPPSDK_RELEASE_VERSION_TAG_W,
    PackageVersion minVersion = WINDOWSAPPSDK_RUNTIME_VERSION_UINT64)
```

Parameters

`majorMinorVersion` See *majorMinorVersion* in MddBootstrapInitialize. Defaults to WINDOWSAPPSDK_RELEASE_MAJORMINOR.

`versionTag` See *versionTag* in MddBootstrapInitialize. Defaults to WINDOWSAPPSDK_RELEASE_VERSION_TAG_W.

`minVersion` See *minVersion* in MddBootstrapInitialize. Defaults to WINDOWSAPPSDK_RUNTIME_VERSION_UINT64.

Return value

On success, returns a resource acquisition is initialization (RAII) object which, when it goes out of scope, undoes the changes that were made by the call to MddBootstrapInitialize.

Example

C++

```
#include <windows.h>

#include <WindowsAppSDK-VersionInfo.h>
#include <MddBootstrap.h>
```

```
#include <iostream>

namespace MddBootstrap {using namespace
    ::Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap; }

int main()
{
    auto mddBootstrapShutdown{ MddBootstrap::InitializeFailFast() };
    // Do work here.
    return 0;
}
```

Requirements

Minimum supported SDK: Windows App SDK version 1.1

Namespace: Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap

Header: MddBootstrap.h

See also

- Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap namespace
- MddBootstrapInitialize

InitializeNoThrow function

Article • 04/19/2023

Calls MddBootstrapInitialize to initialize the calling process to use the specified version of the Windows App SDK's framework package. If the call fails, returns a failure **HRESULT**.

Syntax

C++

```
inline auto Initialize(
    uint32_t majorMinorVersion = WINDOWSAPPSDK_RELEASE_MAJORMINOR,
    PCWSTR versionTag = WINDOWSAPPSDK_RELEASE_VERSION_TAG_W,
    PackageVersion minVersion = WINDOWSAPPSDK_RUNTIME_VERSION_UINT64)
```

Parameters

`majorMinorVersion` See *majorMinorVersion* in MddBootstrapInitialize. Defaults to WINDOWSAPPSDK_RELEASE_MAJORMINOR.

`versionTag` See *versionTag* in MddBootstrapInitialize. Defaults to WINDOWSAPPSDK_RELEASE_VERSION_TAG_W.

`minVersion` See *minVersion* in MddBootstrapInitialize. Defaults to WINDOWSAPPSDK_RUNTIME_VERSION_UINT64.

Return value

On success, returns **S_OK**. If the call to MddBootstrapInitialize fails, returns a failure **HRESULT**.

Example

C++

```
#include <windows.h>

#include <WindowsAppSDK-VersionInfo.h>
#include <MddBootstrap.h>

#include <iostream>
```

```
namespace MddBootstrap {using namespace
    ::Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap; }

int main()
{
    const auto hr{ MddBootstrap::InitializeNoThrow() };
    if (FAILED(hr))
    {
        std::cout << "Error 0x" << std::hex << hr << " in Bootstrap
initialization";
        return hr;
    }
    auto mddBootstrapShutdown{ MddBootstrap::unique_mddbootstrapshutdown(
            reinterpret_cast<MddBootstrap::details::mddbootstrapshutdown_t*>
(1)
        ) };
    // Do work here.
    return 0;
}
```

Requirements

Minimum supported SDK: Windows App SDK version 1.1

Namespace: Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap

Header: MddBootstrap.h

See also

- Microsoft::Windows::ApplicationModel::DynamicDependency::Bootstrap namespace
- MddBootstrapInitialize

WinUI COM Interop

Article07/14/2023

To develop with COM interop APIs for WinUI, you need these headers:

- microsoft.ui.xaml.hosting.desktopwindowxamlsource.h
- microsoft.ui.xaml.hosting.referencetracker.h
- microsoft.ui.xaml.media.dxinterop.h
- microsoft.ui.xaml.window.h
- xamlom.winui.h

For programming guidance for this technology, see WinUI 3.

Enumerations

XAML_REFERENCETRACKER_DISCONNECT

Functions

AddMemoryPressure Informs the host of increased memory allocations since the last notification.
AddRefFromReferenceTracker Indicates that the reference tracker is returning the target XAML object(s) from previous calls to FindTrackerTargets.
AddRefFromTrackerSource Indicates each time that a tracker source calls **IUnknown::AddRef** on the reference tracker (called after **AddRef**).
BeginDraw
BeginDraw

ConnectFromTrackerSource Indicates that a reference tracker source has created its first COM reference on a reference tracker object.
CreateTrackerHandle
DeleteTrackerHandle
DisconnectFromTrackerSource Indicates that a reference tracker source has stopped tracking a reference tracker.
DisconnectUnusedReferenceSources Requests that the host perform garbage collection and remove all unnecessary reference sources.
EndDraw
EndDraw
FindTrackerTargets Finds the reference tracker targets that are reachable from a reference tracker source.
FindTrackerTargetsCompleted Indicates that a garbage collection system has finished making all the calls it needs to IReferenceTracker::FindTrackerTargets (by this time, XAML has pegged all reference tracker targets that it wants to protect).
FlushAllSurfacesWithDevice
FoundTrackerTarget Called whenever a XAML object reference tracker target is found.
get_WindowHandle Retrieves the window handle (**HWND**) of the window represented by the object that implements IWindowNative.
GetReferenceTrackerManager

Gets an IReferenceTrackerManager interface from a XAML object.
GetTrackerTarget Requests the host provide a reference tracker target that references a reference tracker source. This tracker target then controls the lifetime of the tracker source.
GetUiLayerForXamlRoot Gets the visual diagnostics root that can be used to draw on for highlighting elements in the tree.
GetUpdateRectCount
GetUpdateRects
GetVisibleBounds
HitTestForXamlRoot Gets all elements in the visual tree that fall within the specified rectangle.
Invalidate
NotifyEndOfReferenceTrackingOnThread Notifies the host that reference tracking is no longer available on the calling thread; XAML calls this when the **FrameworkView** is uninitialized.
OnXamlRootChange Communicates when a XamlRoot root is added or removed.
Peg Marks that the reference tracker target is in use by the XAML framework, and should not be collected.
PegFromTrackerSource Indicates that a tracker source is unable to protected a reference tracker object.
ReferenceTrackingCompleted Indicates that a garbage collection system has finished with its collection process (at this point, XAML unblocks threads attempting to update tracked references).

ReferenceTrackingStarted Indicates that a garbage collector is performing a collection.
RegisterForUpdatesNeeded
ReleaseDisconnectedReferenceSources Requests that the host call **IUnknown::Release** on any reference tracker objects that have been disconnected by a reference source.
ReleaseFromReferenceTracker Releases the XAML object reference marked in a previous call to AddRefFromReferenceTracker.
ReleaseFromTrackerSource Indicates each time that a tracker source calls **IUnknown::Release** on the reference tracker (must be called before the **Release** call).
RemoveMemoryPressure Informs the host of reduced memory allocations since the last notification.
Resize
ResumeDraw
SetDevice
SetDevice
SetReferenceTrackerHost Registers an IReferenceTrackerHost interface with XAML.
SetSwapChain
SetSwapChain
SetSwapChainHandle

SetTrackerValue
SuspendDraw
TryGetSafeTrackerValue
Unpeg Marks that the reference tracker target is no longer in use by the XAML framework and can be collected.
UpdatesNeeded

Interfaces

IFindReferenceTargetsCallback Defines the interface for callbacks from FindTrackerTargets.
IReferenceTracker Defines the interface implemented by the XAML framework for managing XAML object references.
IReferenceTrackerExtension
IReferenceTrackerHost Defines an interface that provides the global services used by the garbage collection (GC) system used by the XAML framework.
IReferenceTrackerManager Defines the interface for a XAML object reference manager. Implement this interface to manage instances of IReferenceTracker on XAML objects.
IReferenceTrackerTarget Defines an interface implemented by a garbage collector object referenced from XAML.

ISurfaceImageSourceManagerNative
ISurfaceImageSourceNative
ISurfaceImageSourceNativeWithD2D
ISwapChainBackgroundPanelNative
ISwapChainPanelNative
ISwapChainPanelNative2
ITrackerOwner
IVirtualSurfaceImageSourceNative
IVirtualSurfaceUpdatesCallbackNative
IVisualTreeServiceCallback3 Represents additional capabilities of an IVisualTreeServiceCallback2 object.
IWindowNative Enables interoperability between XAML and a native window.
IXamlDiagnostics2 Represents additional capabilities of an IXamlDiagnostics object.

Structures

TrackerHandle__

Interop C# APIs for WinUI

Article • 10/24/2022

As a C# desktop application developer, in .NET you can make use of C# interop classes whose methods wrap and represent several interoperability functions. These include methods of C# classes representing the GetWindowIdFromWindow function, for example.

These C# interop classes are available in .NET as part of the Windows App SDK.

Topics in this section

Expand table

Topic	Description
Microsoft.UI namespace	C# interop classes in the **Microsoft.UI** namespace.

See also

- Call interop APIs from a .NET app

Microsoft.UI namespace

Article • 10/24/2022

Provides to desktop application developers C# interop classes whose methods wrap and represent several interoperability functions. These include methods of C# classes representing the GetWindowIdFromWindow function, for example.

These C# interop classes are available in .NET as part of the Windows App SDK.

Classes

- Win32Interop

See also

- Interop C# APIs for WinUI
- Call interop APIs from a .NET app

Win32Interop class

Article • 10/24/2022

As a C# desktop application developer, in .NET you can make use of the **Win32Interop** class, whose methods wrap and represent several interoperability functions including the GetWindowIdFromWindow function.

The **Win32Interop** class is available in .NET as part of the Windows App SDK.

Definition

Namespace: Microsoft.UI

```
C#
public class Win32Interop
```

Methods

- GetDisplayIdFromMonitor(IntPtr)
- GetIconFromIconId(IconId)
- GetIconIdFromIcon(IntPtr)
- GetMonitorFromDisplayId(DisplayId)
- GetWindowFromWindowId(WindowId)
- GetWindowIdFromWindow(IntPtr)

Applies to

Expand table

Product	Introduced in
WinUI 3	Windows App SDK 1.0

See also

- Interop C# APIs for WinUI
- Call interop APIs from a .NET app

Win32Interop.GetDisplayIdFromMonitor(IntPtr) method

Article • 10/24/2022

Reference

Definition

Namespace: Microsoft.UI

Gets the `DisplayId` that corresponds to the specified *hmonitor*, if the provided `hmonitor` is valid.

C#

```
public static DisplayId GetDisplayIdFromMonitor(IntPtr hmonitor);
```

Parameters

`hmonitor` IntPtr

The handle of the display monitor for which to get the `DisplayId`.

Returns

DisplayId

The display monitor identifier that corresponds to the specified *hmonitor*, if the provided *hmonitor* is valid. Otherwise, `null`.

Applies to

Expand table

Product	Introduced in
WinUI 3	Windows App SDK 1.0

See also

- Win32Interop class
- Manage app windows
- Call interop APIs from a .NET app

Win32Interop.GetIconFromIconId(IconId) method

Article • 10/24/2022

Reference

Definition

Namespace: Microsoft.UI

Gets the icon handle that corresponds to the specified *iconId*, if the provided *iconId* is valid and the system has an `HICON` that represents the icon.

C#

```
public static IntPtr GetIconFromIconId(IconId iconId);
```

Parameters

`iconId` IconId

The identifier for the icon.

Returns

IntPtr

The icon handle that corresponds to the specified *iconId*, if the provided *iconId* is valid and the system has an `HICON` that represents the icon. Otherwise, `null`.

Applies to

Expand table

Product	Introduced in
WinUI 3	Windows App SDK 1.0

See also

- Win32Interop class
- Manage app windows
- Call interop APIs from a .NET app

Win32Interop.GetIconIdFromIcon(IntPtr) method

Article • 10/24/2022

Reference

Definition

Namespace: Microsoft.UI

Gets the `IconId` that corresponds to the specified *hicon*, if the provided `hicon` is valid.

C#

```
public static IconId GetIconIdFromIcon(IntPtr hicon);
```

Parameters

`hicon` IntPtr

The handle of the icon for which to get the `IconId`.

Returns

IconId

The icon identifier that corresponds to the specified *hicon*, if the provided *hicon* is valid. Otherwise, `null`.

Applies to

Expand table

Product	Introduced in
WinUI 3	Windows App SDK 1.0

See also

- Win32Interop class
- Manage app windows
- Call interop APIs from a .NET app

Win32Interop.GetMonitorFromDisplayId(DisplayId) method

Article • 10/24/2022

Reference

Definition

Namespace: Microsoft.UI

Gets the display monitor handle that corresponds to the specified *displayId*, if the provided *displayId* is valid and the system has an `HMONITOR` that represents the display monitor.

```
C#
public static IntPtr GetMonitorFromDisplayId(DisplayId displayId);
```

Parameters

`displayId` DisplayId

The identifier for the display.

Returns

IntPtr

The display monitor handle that corresponds to the specified *displayId*, if the provided *displayId* is valid and the system has an `HMONITOR` that represents the display monitor. Otherwise, `null`.

Applies to

Expand table

Product	Introduced in
WinUI 3	Windows App SDK 1.0

See also

- Win32Interop class
- Manage app windows
- Call interop APIs from a .NET app

Win32Interop.GetWindowFromWindowId(WindowId) method

Article • 10/24/2022

Reference

Definition

Namespace: Microsoft.UI

Gets the window handle that corresponds to the specified *windowId*, if the provided *windowId* is valid and the system has an `HWND` that represents the window.

C#

```
public static IntPtr GetWindowFromWindowId(WindowId windowId);
```

Parameters

`windowId` WindowId

The identifier for the window.

Returns

IntPtr

The window handle that corresponds to the specified *windowId*, if the provided *windowId* is valid and the system has an `HWND` that represents the window. Otherwise, `null`.

Applies to

Expand table

Product	Introduced in
WinUI 3	Windows App SDK 1.0

See also

- Win32Interop class
- Manage app windows
- Call interop APIs from a .NET app

Win32Interop.GetWindowIdFromWindow(IntPtr) method

Article • 10/24/2022

Reference

Definition

Namespace: Microsoft.UI

Gets the `WindowId` that corresponds to the specified *hwnd*, if the provided `HWND` is a valid.

C#

```
public static WindowId GetWindowIdFromWindow(IntPtr hwnd);
```

Parameters

`hwnd` IntPtr

The handle of the window for which to get the `WindowId`.

Returns

WindowId

The identifier that corresponds to the specified *hwnd*, if the provided *hwnd* is valid. Otherwise, `null`.

Applies to

Expand table

Product	Introduced in
WinUI 3	Windows App SDK 1.0

See also

- Win32Interop class
- Manage app windows
- Call interop APIs from a .NET app

Bootstrapper C# APIs

Article • 10/24/2022

As a C# desktop application developer, in .NET you can make use of C# types that wrap and represent the Bootstrapper API.

These types are available in .NET as part of the Windows App SDK.

Topics in this section

Expand table

Topic	Description
Constants for use with the Bootstrapper C# APIs	The following constants are for use with the Bootstrapper C# APIs.
Microsoft.Windows.ApplicationModel.DynamicDependency namespace	C# Bootstrapper APIs in the **Microsoft.Windows.ApplicationModel.DynamicDependency** namespace.

See also

- Bootstrapper API

Constants for use with the Bootstrapper C# APIs

Article • 11/23/2022

The following constants are for use with the Bootstrapper C# APIs.

Definition

Namespace: **Microsoft.WindowsAppSDK**

Assembly: Microsoft.WindowsAppRuntime.Release.Net.dll

Microsoft.WindowsAppSDK.Release class

Expand table

Constant	Value	Description
public const string Channel	[release-dependent]	The Windows App SDK release's channel; for example, "preview", or empty string for stable.
public const string FormattedVersionShortTag	[release-dependent]	The Windows App SDK release's short-form version tag, formatted for concatenation when constructing identifiers; for example, "-p2", or empty string for stable.
public const string FormattedVersionTag	[release-dependent]	The Windows App SDK release's version tag, formatted for concatenation when constructing identifiers; for example, "-preview2", or empty string for stable.
public const ushort Major	[release-dependent]	The major version of the Windows App SDK release.
public const uint MajorMinor	[release-dependent]	The major and minor version of the Windows App SDK release, encoded as a uint32_t (0xMMMMNNNN where M=major, N=minor).
public const ushort Minor	[release-dependent]	The minor version of the Windows App SDK release.
public const ushort Patch	[release-dependent]	The patch version of the Windows App SDK release.
public const string VersionShortTag	[release-dependent]	The Windows App SDK release's short-form version tag; for example, "p2", or empty string for stable.
public const string VersionTag	[release-dependent]	The Windows App SDK release's version tag; for example, "preview2", or empty string for stable.

Microsoft.WindowsAppSDK.Runtime.Identity class

Expand table

Constant	Value	Description
public const string Publisher	"CN=Microsoft Corporation, O=Microsoft Corporation, L=Redmond, S=Washington, C=US"	The Windows App SDK runtime's package identity's Publisher.
public const string PublisherId	"8wekyb3d8bbwe"	The Windows App SDK runtime's package identity's PublisherId.

Microsoft.WindowsAppSDK.Runtime.Packages.DDLM.Arm class

Expand table

Constant	Value	Description
public const string PackageFamilyName	[release-dependent]	The Windows App SDK runtime's Dynamic Dependency Lifetime Manager (DDLM) package's family name, for arm64.

Microsoft.WindowsAppSDK.Runtime.Packages.DDLM.X64 class

Expand table

Constant	Value	Description
public const string PackageFamilyName	[release-dependent]	The Windows App SDK runtime's Dynamic Dependency Lifetime Manager (DDLM) package's family name, for x64.

Microsoft.WindowsAppSDK.Runtime.Packages.DDLM.X86 class

Expand table

Constant	Value	Description
public const string PackageFamilyName	[release-dependent]	The Windows App SDK runtime's Dynamic Dependency Lifetime Manager (DDLM) package's family name, for x86.

Microsoft.WindowsAppSDK.Runtime.Packages.Framework class

Expand table

Constant	Value	Description
public const string PackageFamilyName	"Microsoft.WindowsAppRuntime.1.1-preview1_8wekyb3d8bbwe"	The Windows App SDK runtime's Framework package's family name.

Microsoft.WindowsAppSDK.Runtime.Packages.Main class

Expand table

Constant	Value	Description
public const string PackageFamilyName	"MicrosoftCorporationII.WinAppRuntime.Main.1.1-p1_8wekyb3d8bbwe"	The Windows App SDK runtime's Main package's family name.

Microsoft.WindowsAppSDK.Runtime.Packages.Singleton class

Expand table

Constant	Value	Description
public const string PackageFamilyName	"Microsoft.WindowsAppRuntime.Singleton-preview1_8wekyb3d8bbwe"	The Windows App SDK runtime's Singleton package's family name.

Microsoft.WindowsAppSDK.Runtime.Version class

Expand table

Constant	Value	Description
public const ushort Major	[release-dependent]	The major version of the Windows App SDK runtime; for example, 1000.
public const ushort Minor	[release-dependent]	The minor version of the Windows App SDK runtime; for example, 446.
public const ushort Build	[release-dependent]	The build version of the Windows App SDK runtime; for example, 804.
public const ushort Revision	[release-dependent]	The revision version of the Windows App SDK runtime; for example, 0.
public const ulong UInt64	[release-dependent]	The version of the Windows App SDK runtime, as a uint64l for example, 0x03E801BE03240000.
public const string DotQuadString	[release-dependent]	The version of the Windows App SDK runtime, as a string (const wchar_t*); for example, "1000.446.804.0".

See also

- Bootstrapper C# APIs

Microsoft.Windows.ApplicationModel.DynamicDependency namespace

Article • 10/24/2022

Provides to desktop application developers C# types that wrap and represent the Bootstrapper API.

These types are available in .NET as part of the Windows App SDK.

Classes

- Bootstrap

Structs

- PackageVersion

See also

- Bootstrapper C# APIs

Bootstrap class

Article • 04/19/2023

The **Bootstrap** class contains static helper methods that conveniently wrap calls to the Bootstrapper API.

Definition

Namespace: Microsoft.Windows.ApplicationModel.DynamicDependency

Assembly: Microsoft.WindowsAppRuntime.Bootstrap.Net.dll

```
C#
```

```
public class Bootstrap
```

Methods

- Initialize methods
- Shutdown()
- TryInitialize methods

Initialize methods

Initializes the calling process to use Windows App SDK's framework package. Finds a Windows App SDK framework package meeting the criteria provided in the arguments, and makes it available for use by the current process. If multiple packages meet the criteria, then the best candidate is selected.

```
C#
```

```
public static void Initialize(uint majorMinorVersion);
public static void Initialize(uint majorMinorVersion, string versionTag);
public static void Initialize(uint majorMinorVersion, string versionTag,
PackageVersion minVersion);
```

Parameters

`majorMinorVersion` uint

The major and minor version of the Windows App SDK framework package to load. The version is encoded as `0xMMMMNNNN`, where `M` = Major and `N` = Minor (for example, version 1.2 should be encoded as `0x00010002`).

`versionTag` string

The version tag of the Windows App SDK framework package to load (if any). For example, `"prerelease"`. Defaults to `null`.

`minVersion` PackageVersion

The minimum version of the Windows App SDK framework package to use. Defaults to a new default instance of **PackageVersion**.

Shutdown method

Removes the changes made to the current process by Initialize or TryInitialize. After **Shutdown** is called, your app can no longer call Windows App SDK APIs, including the Dynamic dependency API.

C#

```
public static void Shutdown();
```

TryInitialize methods

Initializes the calling process to use Windows App SDK's framework package. Failure returns false with the failure **HRESULT** in the *hresult* parameter. Finds a Windows App SDK framework package meeting the criteria provided in the arguments, and makes it available for use by the current process. If multiple packages meet the criteria, then the best candidate is selected.

C#

```
public static bool TryInitialize(uint majorMinorVersion, out int hresult);
public static bool TryInitialize(uint majorMinorVersion, string versionTag,
out int hresult);
public static bool TryInitialize(uint majorMinorVersion, string versionTag,
PackageVersion minVersion, out int hresult);
```

Parameters

`majorMinorVersion` uint

The major and minor version of the Windows App SDK framework package to load. The version is encoded as `0xMMMMNNNN`, where `M` = Major and `N` = Minor (for example, version 1.2 should be encoded as `0x00010002`).

`hresult` uint

The failure **HRESULT**, if the initialization failed.

`versionTag` string

The version tag of the Windows App SDK framework package to load (if any). For example, `"prerelease"`. Defaults to `null`.

`minVersion` PackageVersion

The minimum version of the Windows App SDK framework package to use. Defaults to a new default instance of **PackageVersion**.

Applies to

Expand table

Product	Introduced in
Windows App SDK	Windows App SDK 1.0

See also

- Microsoft.Windows.ApplicationModel.DynamicDependency namespace

PackageVersion struct

Article • 04/19/2023

Represents a version of the Windows App SDK framework package.

Definition

Namespace: Microsoft.Windows.ApplicationModel.DynamicDependency

Assembly: Microsoft.WindowsAppRuntime.Bootstrap.Net.dll

C#

```
public struct PackageVersion
```

Constructors

- PackageVersion constructors

Fields

- Build
- Major
- Minor
- Revision

Methods

- ToString()
- ToVersion()

PackageVersion constructors

Initializes a new instance of the **PackageVersion** class.

C#

```
public PackageVersion(ushort major);
public PackageVersion(ushort major, ushort minor);
```

```
public PackageVersion(ushort major, ushort minor, ushort build);
public PackageVersion(ushort major, ushort minor, ushort build, ushort
revision);
public PackageVersion(ulong version);
```

Parameters

`major` ushort

The `major` position of a `major.minor.build.revision` sequence.

`minor` ushort

The `minor` position of a `major.minor.build.revision` sequence. Defaults to 0.

`build` ushort

The `build` position of a `major.minor.build.revision` sequence. Defaults to 0.

`revision` ushort

The `revision` position of a `major.minor.build.revision` sequence. Defaults to 0.

`version` ulong

A `major.minor.build.revision` sequence encoded as a (little-endian) **UInt64** (so that the first 16 bits contain the revision, and so on).

Build field

Gets or sets the `build` position of a `major.minor.build.revision` sequence.

```
C#
```

```
public ushort Build;
```

Major field

Gets or sets the `major` position of a `major.minor.build.revision` sequence.

```
C#
```

```
public ushort Major;
```

Minor field

Gets or sets the `minor` position of a `major.minor.build.revision` sequence.

```
C#

public ushort Minor;
```

Revision field

Gets or sets the `revision` position of a `major.minor.build.revision` sequence.

```
C#

public ushort Revision;
```

ToString method

Retrieves the version as a string.

```
C#

public override string ToString();
```

Returns

string

The `major.minor.build.revision` sequence encoded as a string.

ToVersion method

Retrieves the version as a **UInt64**.

```
C#

public ulong ToVersion();
```

Returns

ulong

The `major.minor.build.revision` sequence encoded as a (little-endian) **UInt64** (so that the first 16 bits contain the revision, and so on).

Applies to

Expand table

Product	Introduced in
Windows App SDK	Windows App SDK 1.0

See also

- Microsoft.Windows.ApplicationModel.DynamicDependency namespace

File and XML schema specifications for UWP apps

Article • 12/13/2019

This section provides information about file and XML schema specifications for Universal Windows Platform (UWP) apps.

Topic	Description
App Installer file	An App Installer file is an XML document (*.appinstaller) that contains app package and bundle information for defining the packages that are part of a related set.
App package block map	The package block map file is an XML document that contains a list of the app's files along with indexes and cryptographic hashes for each block of data that is stored in the package.
App package manifest	Includes reference topics that describe the schema used for packaging Windows apps.
Application content schema	The Application content, or appcontent-ms, schema for Windows apps enables developers to enhance in-app search by providing extra information about your app's content to the Windows Search Index.
Bundle manifest	The bundle manifest file is an XML document that contains a list of app packages and resource packages in the bundle. Includes reference topics that describe the schema used for packaging bundles.
Final content group map	The final content group map file is an XML document (AppxContentGroupMap.xml) that contains automatic content groups for UWP app streaming install.
Schema for Mobile Broadband	Includes reference topics that describe the schema used to provision a mobile broadband enabled device on a mobile broadband network, query subscriber data usage information, or query the results of the last provisioning attempt.
Source content group map	The source content group map file is an XML document (SourceAppxContentGroupMap.xml) that contains required and automatic content groups for UWP app streaming install.
StoreManifest schema	Includes reference topics that describe the schema used by developers to enable optional Store features that the AppxManifest.xml file does not cover.
Tiles, toast, and badge schemas	Includes reference topics that describe the underlying schemas used for tiles, toast notifications, and badges.

Topic	Description
Cortana Voice command (VCD) schemas	Reference topics that describe the XML markup elements and attributes used in Voice Command Definition (VCD) files to specify recognition constraints.